

An Illustrated Primer

by

Kendra Leigh Pugh

B.S., Massachusetts Institute of Technology (2009)

Submitted to the Department of Electrical Engineering and Computer
Science

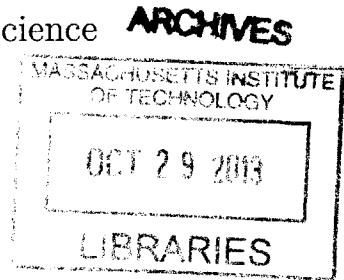
in partial fulfillment of the requirements for the degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2012



© Massachusetts Institute of Technology 2012. All rights reserved.

Author
Department of Electrical Engineering and Computer Science
August 7th, 2012

Certified by
Dennis M. Freeman
Professor of Electrical Engineering
Thesis Supervisor

Accepted by
Dennis M. Freeman
Chairman, Masters of Engineering Thesis Committee

An Illustrated Primer

by

Kendra Leigh Pugh

Submitted to the Department of Electrical Engineering and Computer Science
on August 7th, 2012, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

Educational technology has received unprecedented attention recently, as the efficacy of traditional education methods are coming into question. This thesis introduces the Primer - an e-textbook that initially customizes the presentation of a given course's content to the preferences of the reader, then gradually scales those customizations back as the reader progresses through the e-textbook. The Primer is designed to assist in the "two sigma" problem [1] of bringing the educational experience of a student closer to that of receiving individual instruction from a competent tutor that uses mastery learning techniques.

The Primer initially improves the accessibility of a new course to the student by customizing the representation of the course's content to the student's needs; additional instructional videos, dereferencing of definitions, and hyperlinks back to the defining information originally introducing a concept are provided given the students' preferences. These customizations are based both on the strengths and weaknesses in the student's background in the course's prerequisites, and the information presentation styles preferred by the student. The Primer reduces these customizations over the course of the textbook so that the e-textbook gradually and eventually reflects the delivery style intended by the e-textbook's author, and to give consistency across users' e-textbook experience. The goal is a method of information delivery that is personalized, yet standardized.

The Primer allows students to ease into an e-textbook without becoming discouraged with the presentation style native to the e-textbook's field. It can prevent the student from becoming discouraged by providing more support for areas of the e-textbook's content in which the student has unstable grounding. The Primer was implemented using readings from 6.01 and additional content from OCW Scholar, and feedback was received from students as well as professionals in the field of educational technology.

Thesis Supervisor: Dennis M. Freeman
Title: Professor of Electrical Engineering

Acknowledgments

I'd like to thank Dennis Freeman; primarily for guidance and support, ultimately for a large amount of patience, and firstly for allowing me the opportunity to work on the Primer.

I'd like to thank my parents, who simultaneously insisted on constraint satisfaction to meet the qualifications of a Real Person TM, and insisted that I could do whatever I wanted. I'm not sure I'm quite there yet with either, but I wouldn't have thought it possible without your persistence.

Thank you to my friends, who put up with me gracefully. In particular, to Caitlin Johnson, Sarina Canelake, Adam Hartz, and Alejandro Seden; Matt Malchano, Clara Rhee, Danielle Magrogan, Coral Ash, Andrew Rowe, John Innes. This wouldn't have been possible without you.

Thank you to Sal Khan, who helped me understand 18.06, and whose innovation legitimized my interest in "TA stuff"

Contents

- 1 Introduction 11**
 - 1.1 Educational Technology 11
 - 1.2 The Primer 12
 - 1.3 Evaluation 12
 - 1.4 Contents 13

- 2 Previous Work 15**
 - 2.1 Machines and the Study of Education 15
 - 2.2 WWW and Online Education 17
 - 2.3 Electronic and Interactive Textbooks 18

- 3 The Primer 21**
 - 3.1 User Profiling 22
 - 3.2 Approaching Standardization 22
 - 3.3 Disclaimer 22

- 4 Primer Interface 25**
 - 4.1 Admin 25
 - 4.1.1 User Profile 25
 - 4.1.2 Chapters 26
 - 4.1.3 Definition Elements 27
 - 4.2 User 28
 - 4.2.1 Login and Logout 28

4.2.2	Profile	29
4.2.3	Chapters	30
4.2.4	Definitions	31
5	Primer Architecture	37
5.1	Content	37
5.1.1	Course Readings	37
5.1.2	Definitions	38
5.1.3	Videos	38
5.2	Django	39
5.2.1	Models	40
5.2.2	Views	41
5.2.3	Templates	42
5.3	Server	45
5.3.1	Apache	45
5.3.2	mod_wsgi	45
6	Feedback	47
6.1	Primer	47
6.2	E-Textbooks	48
7	Contributions	51
7.1	The Primer	51
7.1.1	The Ideal	51
7.1.2	The Implementation	52
7.2	What's Next	52
A	Code	55

List of Figures

4-1	Administration Home	26
4-2	Admin User Profile Page	27
4-3	Example User Profile	28
4-4	Chapter Admin Page	29
4-5	Detailed view of Chapter Admin Page	30
4-6	Definition Elements Admin Page	31
4-7	Detailed view of Definition Element Admin Page	32
4-8	Primer Masthead	32
4-9	Primer Login Page	33
4-10	Primer User Profile	33
4-11	Primer Excerpt	34
4-12	A Triggered Definition Element	34
4-13	Primer Chapters	34
4-14	Primer Definition Elements	35
4-15	Primer Definition Element	36
5-1	Reference to the ‘voltage’ Definition Element	43
5-2	The Primer Template Hierarchy	44

Chapter 1

Introduction

This thesis explores the current state of educational technology, and attempts to address an outstanding problem in educational technology with a new combination of solutions. The outstanding problem in educational technology this thesis attempts to address is known as the two sigma problem [1]. The new combination of solutions is an e-textbook that initially adapts its presentation to the student's preferences, then gradually scales back the customizations over the course of the e-textbook such that by the end of the student's experience, the e-textbook is a more uniform resource across students. This customized content delivery system will be referred to throughout as The Primer.

1.1 Educational Technology

The Primer is introduced during an exciting time for the field of educational technology. Unprecedented interest in educational reform and technology-enabled learning sets the stage for a multitude of new approaches to a student's education, both in the classroom and at home [10] [6]. Simultaneously, the size of a given class is no longer necessarily limited by the size of a room, the number of desks or even the number of instructors available; web publications [7] and Massive Open Online Courses (MOOCs) [3] expand educational offerings to anyone willing to learn. The proliferation of tablets and popular adoption of e-books has lead to a race for domi-

nation among various e-textbook publishing formats. The Primer is introduced at a time where adaptation of its ideas could be beneficial to a great number of learners worldwide.

1.2 The Primer

In the midst of all this innovation, the Primer is an attempt to carefully improve a classic aspect of the higher education experience, the textbook, without sacrificing any of its inherent benefits. An e-textbook device has the benefit of being lighter and easier to carry than a traditional textbook, and an e-textbook can theoretically be transferred to a new device when normal wear and tear affects the previous device. A textbook with presentation style personalized to the reader can improve the rate and quality of information transfer between the textbook and the reader.

But what of standardization? What if every economics student read a slightly different version of The Wealth of Nations [9]? The downside of MOOCs is that professional oversight of a student's experience may be diluted to the extent that it is not guaranteed to be sufficiently rigorous or comparable to that of other students.

The Primer attempts to address all these issues. Too much customization and an e-textbook experience isn't relatable to any other student in the course. Too little and the experience provides little advantage over a traditional textbook. The Primer aims to provide improvement over the traditional textbook experience without sacrificing the benefit that comes from a common source of information.

1.3 Evaluation

The Primer was evaluated by several experts both familiar with the course content upon which the Primer was based and expressing interest in the field of educational technology. These experts are participants in educational technology efforts such as OCW, the iCampus project, and edX. The experts were asked to read a sample of the Primer under several different profiles, create a dynamic element for use within

the Primer, edit the Primer to include this dynamic element, and give feedback on the experience.

1.4 Contents

Chapter 2 discusses both the historical and current related work relevant to the Primer.

Chapter 3 is an overview of the goals of the Primer and the specifics of which problems it does and does not solve.

Chapter 4 provides a detailed description of the Primer's user interface. It establishes both the educator and student's interface configurations, and indicates the decisions that went into establishing the user interface as provided.

Chapter 5 presents the architecture of the Primer.

Chapter 6 reviews the evaluations of the Primer.

Chapter 7 concludes the thesis and indicates desirable possible extensions to the Primer.

Chapter 2

Previous Work

Educational technology has a rich history; traditional technologies include those items classically associated with a traditional instructional setting, such as blackboards, chalk, textbooks, and lecture halls. Although these elements aren't typically indicated by the contemporary use of the term 'Educational Technology', they are the standard against which emerging technologies are measured. The past century, and in particular the past ten years, have seen incredible advances and a myriad of options proposed to improve upon these traditional technologies. The Primer is comparable to or influenced by these advances.

2.1 Machines and the Study of Education

The first significant advancements in educational technology in the 20th century were the introduction of training films and teaching machines. Thomas Edison's 1908 film *Flypaper* represents the first educational film in the United States, and Sidney Pressey's teaching machine represents the first in a long effort towards an automated tutor [8]. The Primer represents an element of educational technology that derives both from educational film and machines that respond to student input; it would be impossible without these two pieces of groundbreaking technology. Edison was inspired by the blossoming industry of film; Pressey by the goal of providing teachers with more individual time with students. Both goals are relevant to the Primer; it

intends to make use of the ubiquity of recording technologies and the ability of anyone with a computer, a webcam and the Internet to make an instructional video, and to improve personal learning experience so that more time with instructors can be spent refining understanding and clarifying those difficulties a student encounters that can't be approached by automated means.

Training film use boomed during World War II as soldiers needed basic instruction before entering the field. This legitimized the presence of video in the instructional setting. Simultaneously, the advances in behavior psychology by Skinner and others legitimized the quantitative study of human experience in relation to learning. These historical developments are relevant to the Primer in that the relevance of video to the learner was not solidified until World War II, and that a quantitative approach to measuring and changing students' experiences, rather than exclusively test scores to measure performance, had not been legitimized. These advances solidified those concepts.

The Primer makes heavy use of hyperlinks and hypermedia. Precursing the use of hypertext and hypermedia, Vannevar Bush wrote of the utility of a device that tracked the history of a person through research materials, called a memex [2], in 1945. Much later, Ted Nelson wrote and researched hypertext, hypermedia, and transclusion in an effort to manifest a system similar to the World Wide Web.

Though the behavioral psychology movement focused on human learning, little effort to relate the results of such experiments to the system of education was recognized before the ideas promoted by Edward Bloom. Bloom's famous "two sigma" problem highlights the basics of the disparity between individual professional tutoring and the experience inside a typical classroom [1]. The Primer attempts to bridge the two sigma gap by personalizing the presentation of textbook information to the student. This personalization is expected to improve retention of new information and automatically reinforce information with which the student has difficulty.

2.2 WWW and Online Education

The World Wide Web contributes greatly to the idea of the Primer; the Primer is designed to be accessed from a web server, and viewed on a personal computer, laptop, mobile device, or tablet device. The advances in educational technology following the proliferation of the World Wide Web provide a backdrop against which the Primer can be evaluated.

One of the most ambitious efforts to introduce educational technologies that take advantage of the web was the Microsoft iCampus Project [4]. Innovation such as Technology Enhanced Active Learning and online tutors used in MIT courses came out of the iCampus project. These technologies center around human-computer interaction, and much of the coursework associated with the technologies could be completed at any time from any computer with a connection to the internet. The iCampus project represents a step in the direction towards technologies such as the Primer.

Another step in the right direction is MIT's OpenCourseWare (OCW) [7]. OCW is an archive of materials used during previous terms of MIT courses for instruction and learning. The Primer's content is derived in part from OCW offerings; the OCW offerings in turn were developed in response to student input on information not readily absorbed through a traditional textbook. In addition, the Primer would be an excellent resource for OCW Scholar. OCW Scholar are traditional OCW offerings augmented with additional content and designed for independent learners to use in absence of additional resources for learning, such as an instructor or course staff.

Similar to OCW scholar is The Khan Academy [6]. Started by Salman Khan as an effort to educate his cousins over the internet, Khan Academy features over 3,200 instructional videos as well as exercises designed to teach anyone with an internet connection the basics in a wide variety of disciplines. The success of Khan Academy has led to experimentation in California and Colorado K-12 schools with the concept of an "Inverted" or "flipped" classroom. In such a classroom, students watch videos and absorb instructional materials at home, then practice skills with one another and under the instructional guidance of teachers or tutors. The Primer is an example of

an educational technology that is both influenced by Khan Academy - educational videos are available and practice sessions are tailored to the student - and would augment the Khan Academy experience by providing a textbook.

Yet another emerging educational technology that makes use of the web is the Massive Open Online Course, or MOOC. Similarly to sites such as OCW and The Khan Academy, the Primer is an educational technology that could see exemplary use in a MOOC. MOOCs are different from OCW in that they are designed to replace the classroom experience, not archive it; a greater variety of materials are available, typically including contact with an instructor for the course. MOOCs are a relatively new concept; some MOOCs under development include MIT and Harvard's edX, Udacity, and Stanford's Coursera.

2.3 Electronic and Interactive Textbooks

Though previously mentioned technologies have contributed to the emergence of The Primer, The Primer is more related to electronic textbooks than any previously discussed technology. Electronic textbooks were originally developed in the 1960's by Doug Engelbart at Stanford Research Institute. The advent of the web allowed massive distribution of electronic textbooks, and the development of electronic textbook readers made electronic textbooks a viable alternative to paper media.

Today, competition for the electronic textbook market is still strong, and several formats for electronic textbooks exist, including SCORM, HTML5, ePub and KF8. The different formats are a result of different companies developing for the electronic book market at the same time, as well as an independent effort to come up with an open standard. The Primer is published in HTML5 to ensure the greatest compatibility with the largest number of devices. Closest in technological achievement to The Primer is Nature Publishing's *Principles of Biology*, an interactive, modular textbook that is customizable by the instructor for a course. Additional companies are also introducing interactive electronic textbooks [5]. The difference between the Primer and these technologies is that the Primer's presentation is customizable at the user

level, by the user. In this way, the Primer represents a new direction for e-textbooks.

Chapter 3

The Primer

The Primer is an electronic textbook that customizes the presentation of information to the stylistic preferences and educational strengths and weaknesses of the student reader. The Primer is designed to do this in attempt to reduce the “two sigma” gap - the performance difference between a student individually tutored using mastery techniques and a student receiving traditional instruction. In some ways, the Primer represents an inversion of Vannevar Bush’s Memex [2] - instead of recording the trail a student takes through a particular set of standard resources, the Primer tailors one of many paths through a set of standard resources for the student and provides the student with that path. The standard set of resources is always available.

The change the Primer induces is the number and depth of exposures to definitions and assistive videos the student automatically receives. The student, as well as any administrator, can change the student’s preferences at any time - if a student’s proficiency in a particular area increases, or if their preference for video lectures decreases, those changes can be reflected by changing their corresponding setting in the Primer.

The Primer is available anywhere there is an internet connection and a web browser. The Primer can be viewed on a personal computer or a mobile or tablet device. The profile a student develops with the Primer persists between a student’s sessions with the Primer.

3.1 User Profiling

The Primer engages in user profiling through explicit references to data stored in a user profile. User profiles keep track of student proficiency in a number of sub-fields relevant to the textbook's domain, as well as general student preferences for definition availability.

3.2 Approaching Standardization

The Primer gradually scales back the customizations it makes by dividing a textbook into chapters. Each chapter can have a different level of allowed customizations and the level of customization can be adjusted across users. The Primer's customizations can be adjusted by chapter; definitions and videos will be less or more likely to dereference on the page based on these adjustments. Since adjustments to chapters affect all users, the Primer can be standardized by manipulating chapter customizations. In particular, customization by chapter allows later chapters to have fewer dereferences, making those chapters more closely resemble the baseline layout of the textbook. In this way, a more uniform textbook can slowly emerge over the administration of a course, such that by the end of a course, all students' Primers look the same.

3.3 Disclaimer

The Primer is not an electronic tutor; it does not record students' interactions in an effort to establish their skill levels. The Primer simply adjusts to the decisions for information delivery that students make. The Primer is not a complete learning system. Learning management systems also handle evaluation and administrative tasks.

The Primer is not a Learning Content Management System (LCMS). An LCMS typically involves the ability to create and manipulate entire learning courses; the Primer is just an e-textbook. One similarity between LCMSes and the Primer is that the Primer's definition elements could be construed as learning objects. The Primer

adjusts the learning path of a student to the students' preferences, but the Primer stands-in for a textbook in a course, not all course materials.

The Primer is different from existing e-textbook solutions in the following ways: although electronic textbooks are in existence today, none attempt to adjust presentation of information in the same ways and at the same level as The Primer. E-textbooks that customize their presentation to the reader exist, but these customizations are determined on the course level; all textbooks for students in a particular course look the same. The Khan Academy website most closely approximates the pursuit of experiences that differentiate The Primer from another e-textbook; however, Khan Academy does not publish e-textbooks.

Chapter 4

Primer Interface

The front end of the Primer is the most important aspect of its functionality; customized user experience means very little without discussion of user interface. This section walks through the user interface functionality of the Primer.

4.1 Admin

The Primer has a few essential administrative components. Administrative components in the Primer can be used to add, change, or delete users; add, edit, or delete a user's profile preferences; add, change or delete chapter difficulty coefficients; and add, change, or delete definition elements. Additional changes to the Primer, such as changing chapter content, must be done by editing Django templates, discussed in Chapter 5. The Primer's admin interface makes use of Django's automatic admin interface, `django.contrib.admin`. The main administration page is shown in figure 4-1.

4.1.1 User Profile

The user profiles page allows administrators to add or make changes to a user's user profile. The main page to access user profiles is shown in figure 4-2. An example of a user profile is shown in figure 4-3. Note that user profiles contain five values; four of

Site administration

Auth	
Groups	+ Add Change
Users	+ Add Change

Book	
Chapters	+ Add Change
Definition elements	+ Add Change
User profiles	+ Add Change

Recent Actions
My Actions
kpughProfile User profile
kpughProfile User profile
+ kpughProfile User profile
+ bloomProfile User profile
+ freemanProfile User profile
+ hartzProfile User profile
+ hartzProfile User profile
+ hartz User
freeman User
basic_search Definition element

Figure 4-1: Administration Home

the values indicate a user’s self-described proficiency in one of four areas that are the focus of 6.01, the course upon which this implementation of the Primer is based. The fifth value indicates a user’s general preferences regarding definition dereferencing.

4.1.2 Chapters

The chapters page allows administrators to add, delete, and make changes to the difficulty coefficient associated with each chapter. The main page to access chapters is show in figure 4-4. An example of a chapter admin page is shown in figure 4-5. Chapter coefficients are used to modify the overall levels of definition dereferencing that happens in a given chapter; the lower the value, the less likely it is that a definition will be dereferenced. This enables the standardization of the presentation of the Primer as users progress to later chapters.

Select user profile to change

Action:	<input type="text" value="-----"/>	<input type="button" value="Go"/>	0 of 4 selected
<input type="checkbox"/>	User profile		
<input type="checkbox"/>	hartzProfile		
<input type="checkbox"/>	freemanProfile		
<input type="checkbox"/>	bloomProfile		
<input type="checkbox"/>	kpughProfile		
4 user profiles			

Figure 4-2: Admin User Profile Page

4.1.3 Definition Elements

The definition elements page allows administrators to add, edit, and delete definition elements. The main page to access definition elements is shown in figure 4-6, and an example definition element is shown in figure 4-7. Definition elements are the definitions and videos whose presentations are customized based on user preferences. In this implementation of the Primer, they are vocabulary word definitions and videos that are supplementary to the main e-textbook. Each definition element has a difficulty rating for each of the four main areas of focus in the e-textbook course. In addition, each definition element has an overall difficulty rating associated with definition elements. If these difficulty ratings, when multiplied by a chapter definition coefficient, exceed the ratings specified by the user in their preferences, then the element in the 'triggered' field follows the element in the 'baseline' field in-line within a chapter of the Primer. If not, then the 'baseline' field appears as a hyperlink to a page featuring both the 'baseline' and 'triggered' fields. This functionality is detailed and figures are available in the 'Definitions' section below.

Change user profile

User:	<input type="text" value="kpugh"/>	<input type="button" value="⊕"/>
DefinitionRating:	<input type="text" value="3"/>	
ProgrammingBac	<input type="text" value="5"/>	
SystemsBackgrou	<input type="text" value="1"/>	
CircuitsBackgrou	<input type="text" value="1"/>	
ProbabilityBackg	<input type="text" value="1"/>	

Figure 4-3: Example User Profile

4.2 User

The Primer's user components are those aspects of the Primer meant for student interaction. The chapters of the Primer are the main focus of user interaction, but additional pages are required to make the Primer most useful. User pages feature a masthead with links to the login, profile, chapter, and definition aspects of the Primer, shown in figure 4-8. The user pages are described below.

4.2.1 Login and Logout

The Primer's login page is shown in figure 4-9. Users log into the Primer in order to edit their profile preferences and allow the chapters of the Primer to adjust to their preferences. The login page will also display one of several error messages associated with logging in as appropriate. If a user is already logged in, the user's name will appear in the error message and a link to the logout page will appear. If the user

Select chapter to change

Action:	<input type="text" value="-----"/>	<input type="button" value="Go"/>	0 of 8 selected
<input type="checkbox"/>	Chapter		
<input type="checkbox"/>	Chapter 1		
<input type="checkbox"/>	Chapter 2		
<input type="checkbox"/>	Chapter 3		
<input type="checkbox"/>	Chapter 4		
<input type="checkbox"/>	Chapter 5		
<input type="checkbox"/>	Chapter 6		
<input type="checkbox"/>	Chapter 7		
<input type="checkbox"/>	Chapter 8		
8 chapters			

Figure 4-4: Chapter Admin Page

login attempted is invalid or inactive, a message to that effect will also appear. If the user selects logout, a short logout message appears along with a link to log in again.

4.2.2 Profile

The user's profile page is shown in figure 4-10. Users can view and edit their profile from the user profile page. The profile page allows a user to control the way the Primer appears based on their personal preferences; if their proficiency ratings exceed the difficulty rating of a given Primer definition element in a given chapter multiplied by the chapter's standardization coefficient, than the baseline version of the definition element will appear. This calculation is performed for all areas of proficiency in the course book as well as for a general definition difficulty. If any of the difficulty ratings exceed the proficiency level of a user, than the triggered version of the definition element will be displayed. Users can update their profile preferences and the change

Change chapter

AdvanceMultiply	<input type="text" value="0.7"/>	
✖ Delete	Save and add another	Save and continue e

Figure 4-5: Detailed view of Chapter Admin Page

will immediately appear upon completion and submission of the HTML form.

4.2.3 Chapters

The chapter pages are the heart of the Primer. These pages display the basic e-textbook content as well as the definition elements in their appropriate state given the definition element and the logged-in user's preferences. An example of the Primer is shown in figure 4-11. Note that images in the Primer are static and appear regardless of user preferences; the Primer could be easily adapted to make this a user preference, however. Figure 4-11 shows a definition element that has not been triggered; the hyperlink will direct the user to a definition element page. Figure 4-12 shows the same definition element triggered; the hyperlink is no longer available, and the definition appears immediately after the baseline appearance of the definition element.

Links to all chapters are available on a separate chapters page, shown in figure 4-13.

Select definition element to change

Add defin

Action:	-----	Go	0 of 100 selected
<input type="checkbox"/>	Definition element		
<input type="checkbox"/>	state_machine		
<input type="checkbox"/>	belief_state		
<input type="checkbox"/>	state_estimation		
<input type="checkbox"/>	PCAP		
<input type="checkbox"/>	state_estimation_video		
<input type="checkbox"/>	interpreter		
<input type="checkbox"/>	expression		
<input type="checkbox"/>	variable		
<input type="checkbox"/>	mutation		
<input type="checkbox"/>	aliasing		
<input type="checkbox"/>	environment		
<input type="checkbox"/>	object_oriented_programming		
<input type="checkbox"/>	class		
<input type="checkbox"/>	instance		

Figure 4-6: Definition Elements Admin Page

4.2.4 Definitions

The definition elements are what make the Primer different from a regular e-textbook. A list of all definition elements is available at the definitions webpage, shown in figure 4-14. Each definition element has its own page with both the baseline name of the definition and the triggered content of the definition displayed. An example of this page is shown in figure 4-15. Definition elements are also displayed within the chapter pages, and their appearance changes based on the settings for the definition element, the chapter, and the user coming together to create the particular definition reference.

Change definition element

Name:	<input type="text" value="state_machine"/>
Chapter:	<div style="border: 1px solid black; padding: 2px;"><div style="background-color: #f0f0f0; padding: 2px;">Chapter 1</div><div style="background-color: #f0f0f0; padding: 2px;">Chapter 2</div><div style="background-color: #f0f0f0; padding: 2px;">Chapter 3</div><div style="background-color: #f0f0f0; padding: 2px;">Chapter 4</div></div> <p>Hold down "Control", or "Command" on a Mac, to select more than one.</p>
ProgrammingThr	<input type="text" value="2"/>
SystemsThreshold	<input type="text" value="0"/>
CircuitsThreshold	<input type="text" value="0"/>
ProbabilityThresh	<input type="text" value="0"/>
Baseline:	<input type="text" value="state machine"/>
Triggered:	<input type="text" value="a method of modeling systems whose ou"/>
DefinitionThresh	<input type="text" value="2"/>

Figure 4-7: Detailed view of Definition Element Admin Page

Login Profile Chapters Definitions

Figure 4-8: Primer Masthead

[Login](#) [Profile](#) [Chapters](#) [Definitions](#)

Login Page

kpugh is already logged in [logout](#) [first chapter](#)

Username:
Password:

Figure 4-9: Primer Login Page

[Login](#) [Profile](#) [Chapters](#) [Definitions](#)

Welcome to the Primer

You are logged in as kpugh

Edit your profile below. 0 is a novice, 10 is an expert.

Definition Rating:
Programming Background:
Systems Background:
Circuits Background:
Probability Background:

[logout here](#)

[here is the first chapter](#)

Figure 4-10: Primer User Profile

The laws for the flow of electrical current are similar to those for the flow of an incompressible fluid. The net flow of electrical current into a **node** must be zero.

The following circuit has three elements, each represented with a box.

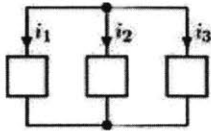


Figure 4-11: Primer Excerpt

The laws for the flow of electrical current (the flow of electric charge through a path in a circuit) are similar to those for the flow of an incompressible fluid. The net flow of electrical current into a **node** must be zero.

The following circuit has three elements, each represented with a box.

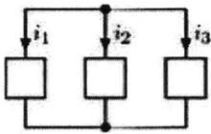


Figure 4-12: A Triggered Definition Element

Login Profile Chapters Definitions

- Chapter 1
- Chapter 2
- Chapter 3
- Chapter 4
- Chapter 5
- Chapter 6
- Chapter 7
- Chapter 8

Figure 4-13: Primer Chapters

Login Profile Chapters Definitions

- state machine
- belief state
- State Estimation
- PCAP
- State Estimation Video
- interpreter
- expression
- variable
- mutation
- aliasing
- environment
- object-oriented programming
- class
- instance
- method
- inhertiance
- recursion
- list comprehension
- lambda
- self
- state transition diagram
- transition table
- controller
- plant
- signal

Figure 4-14: Primer Definition Elements

[Login](#) [Profile](#) [Chapters](#) [Definitions](#)

mutation

the action of changing the values associated with a particular data structure without also overwriting that data structure. In particular, when a variable is mutated the variable name points to the changed structure

[Back to Chapter 2](#)

[Back to Chapter 3](#)

Figure 4-15: Primer Definition Element

Chapter 5

Primer Architecture

5.1 Content

The innovative aspect of the Primer is its customizable interface, but in order to demonstrate the Primer, a suitable textbook had to be adapted to the Primer platform. The Primer is most useful when content suitable for definition elements are also incorporated; definitions and videos make the Primer different from a standard e-textbook. The Primer was made from a collection of source materials that I had access to; the 6.01 course readings and 6.01 OCW Scholar offerings.

5.1.1 Course Readings

I used the course readings for the MIT class *6.01: Introduction to EECS I* from spring term 2012. This textbook provided the optimum platform to demonstrate the power of the Primer; the stylistic aspects of the authors in the 6.01 course readings are preserved, yet the definitions and videos designed to present repetition of information and information in a different way are readily accessible. The original \LaTeX files for the 6.01 course readings were carefully adapted to HTML5, including transformation of PDF figures into .jpg images. As a consequence of the adaptation, some small features of the textbook changed; figure references were changed from the standard \LaTeX method to simply “above” or “below” as the image reference appeared in the

source document. In addition, exercises in the 6.01 readings were visually distinct from regular text; this distinction was removed for ease of conversion to HTML5 and to ensure that the Primer rendered correctly on more browsers. These distinctions could be added back in using CSS styling.

5.1.2 Definitions

Once the base of the textbook was established, material for the definition elements in the Primer had to be established as well. While working for 6.01 OCW Scholar, I developed “Weekly Summaries” of 6.01 course content. The purpose of OCW Scholar over a traditional OCW offering is to increase the amount of support material available in order to enable independent learners. These summaries were based on my experiences as a teaching assistant in 6.01 during spring semester 2011; students had difficulty absorbing course material as presented, and these summaries attempted to ameliorate those difficulties.

In particular, the “Vocabulary” section of the summaries provide those terms a student would have great difficulty passing the course without understanding. These terms were used as the basis for definition elements in the Primer. These terms are exactly the kind of information that would be useful as definition elements - they are chosen based on student feedback on what information is essential to the course, but possibly missed on the individual efforts of a student through the course material. Definitions for the vocabulary words were taken from the 6.01 course readings.

5.1.3 Videos

In addition to weekly summaries, I developed a number of short video presentations for the 6.01 OCW Scholar website. These videos were also based on student feedback as to the more challenging or unintuitive aspects of the course. The videos were also organized to cover a short section of the 6.01 material at a time, breaking the material into conceptual chunks that are also meant to be viewed about once per week. The videos were distributed throughout the 6.01 readings in a manner that made sense

given the context of the chapters.

5.2 Django

The Primer was developed using the Django development framework. Django is an open source Web 2.0 application framework, written in Python, which follows the model-view-controller architectural pattern. This pattern is established through the use of files; models, views, and controllers all live in separate files. Additional files manage application settings. A single Django installation can run multiple websites, each with their own content and applications. This means that multiple Primers can be established using a single Django installation.

The "model" file, known as `models.py` in a Django application, allows a programmer to set up database tables automatically by writing Python classes. These classes are then accessible through a Python database-abstraction API that can be used to create, retrieve, update and delete objects.

The type of database can be specified in a separate settings file. Django has built-in support for PostgreSQL, MySQL, SQLite, and Oracle. Django will also accept other database engines. I used SQLite for developing this application, but did not have to manipulate the database tables directly, as the Django database abstraction API made it easy to manage information in the database through Python.

The "controller" aspect of the model-view-controller architectural pattern is established, somewhat confusingly, by a file called `views.py`. `views.py` controls the use of database elements in webpages, and processes incoming HTML requests to create or change the model data.

The "view" aspect of the model-view-controller architectural pattern is implemented using Django templates. Django templates are HTML files containing a Django-specific template language that allows hierarchical organization and extensibility of different portions of HTML. Templates receive database and other information as passed on from `views.py` and renders it in HTML, completing the MVC architecture.

Two additional files complete the essentials in a Django website application. Each Django project has a `settings.py` file, which specifies a variety of settings for a given Django project. Settings include which database engine and database to use, the location of static and media files, which applications and middleware are included, logging configurations, and the location of the URL configuration.

The final file used in a given Django application is `urls.py`, the URL configuration itself. URLs are matched through regular expressions to specific controller logic in `views.py`, creating the final connection between website user and application.

The specific data models, view-rendering functions, and html templates used together describe the basic functionality of the Primer, and the specifics of three components are described in further detail in the following sections of this chapter. The code for the Django application is included in the appendices.

5.2.1 Models

The Primer uses several model classes to store database information associated with its administration. Users, definition elements, and chapters all require database storage and transactions in order for the Primer to function.

The `UserProfile` model class is the model class used to represent a given user's profile. A user's profile contains his or her preferences regarding the appearance of the Primer. In particular, the `UserProfile` model class contains proficiency ratings in the four main areas of 6.01, as well as a general rating of proficiency with definitions. These values are used in calculations of the appearance of a particular Primer definition object in the `chapters` view.

Since the Primer uses Django's default settings for administration of users, user profiles must be initiated separately from users. For users and groups, the Primer uses Django's `django.contrib.auth.models` classes.

Since Django includes a Python database-abstraction API, both users and user profiles could be initiated in batches using a short Python script. Scaling the Primer up to deal with an MIT class's worth of users is as simple as providing a file of usernames and passwords.

The `Chapter` model class represents a given Chapter of the Primer. Chapter models are necessary to retain the coefficient associated with the advanced nature of a given chapter; these coefficients are used to determine whether a given Primer chapter will have more or fewer dereferenced definitions.

The `PrimerElement` model class represents a Primer element, a learning object whose appearance changes based on the comparison of a given user's preferences with the `PrimerElement`'s ratings. This class stores the ratings associated with a given element, which are multiplied by the chapter's advancement coefficient to determine the overall difficulty associated with a given `PrimerElement`. These ratings are compared to the preferences specified by a given user's `UserProfile` attributes when deciding whether to dereference the definition of a Primer definition object in the text of the Primer.

The `DefinitionElement` model class is a subclass of the `PrimerElement` model class. `PrimerElement` model objects have difficulty ratings in the four main areas of 6.01; `DefinitionElements` have an additional rating for general definition difficulty. `DefinitionElements` are subclassed to allow for extensibility to the Primer; additional `UserProfile` preferences could be added for comparison with different subclasses of `PrimerElement`. Different logic could be used to determine the rendering of different subclasses.

5.2.2 Views

Views are where the control logic for the Primer resides. Views query the database and receive user input, then perform any specified operations on that information, then pass the results on to templates for rendering.

The `definitions` view controls the display of individual definition elements on the Primer website. It queries the database for the term and definition or video associated with that term and sends them to a generic template that is used to display any individual definition.

The `chapter` view handles the majority of logic that goes into determining the custom appearance of the Primer. For a given chapter, a query of the definition

elements is done to determine which definition elements are relevant to that chapter. Then, for each definition element, a comparison is done of that definition element's thresholds to the user's background as specified by the user's profile. The definition element's thresholds are multiplied by a coefficient determined by the chapter; this is where the standardization through gradual regression to hidden definitions comes in. If the threshold exceeds the background, then the definition element is dereferenced and the definition or video will appear alongside the definition element's baseline name in the text of the Primer website. If not, a hyperlink to the definition or video will appear in the text of the definition's baseline name.

The `site_login` view handles users logging in. If a user attempts to log in while another user is logged in, or with an invalid or expired username, he will be brought back to the login screen with an error message.

The `logging_in` view handles the requests associated with user login. It queries the database with the given username and password, checks to see if the username is valid, checks to see if the user is active, and then logs the user in. Django uses the PBKDF2 algorithm with a SHA256 hash for passwords.

The `site_logout` view handles site logouts. If a user is logged in, it logs the current user out and provides a link back to the login page.

The `profile` view retrieves the user that is logged in's profile values for display in a template. The difficulty rating preferences for the four main areas of 6.01 are retrieved, as well as the general definition difficulty rating.

The `submit_profile` handles requests to change a user profile. It receives an HTML POST request and updates the values associated with a user profile accordingly, then passes the new values to a template to render immediately.

5.2.3 Templates

Djangos template engine provides a powerful mini-language for defining the user-facing layer of an application. Templates are the "view" portion of the MVC architecture; they are the presentation logic. Unlike other parts of the Django development framework, templates can be maintained by anyone with an understanding of HTML;

no knowledge of Python is required. This means that the requirements for authoring a textbook using the Primer system is access to the admin pages, access to the templates, and knowledge of HTML. The Primer uses several templates in a hierarchical arrangement to determine its presentation.

Templates are the place where user interface design for the Primer occurs. Changing the Django templates associated with the Primer changes the appearance of the Primer itself. Stylistically, the Primer has been left with no CSS styling - its appearance adapts to the preferences of the browser and device and/or browser used to access the Primer.

Templates are also the place where Primer chapter content is changed. In order to add references to definition elements to the text of the Primer, a variable reference to the Primer `DefinitionElement` must be added to the appropriate chapter template. An example of this variable reference appears in figure 5-1; `voltage` is the variable.

```
<p>Voltage is a difference in electrical potential between two different
points in a circuit. We will often pick some point in
a circuit and say that it is "ground" or has voltage 0. Now, every
other point has a {{voltage}} defined with respect to ground. Because
voltage is a relative concept, we could pick <em>any</em> point in the
circuit and call it ground, and we would still get the same results. </p>
```

Figure 5-1: Reference to the ‘voltage’ Definition Element

`base.html` is the base of the Primer template hierarchy. It contains the basic layout of all templates that inherit from it. Django’s template language allows template inheritance - templates can inherit structure from other templates, and modify sections of templates called blocks in those inherited templates. The Primer’s template hierarchy is shown in figure 5-2.

`login.html` inherits from `base.html`. It determines the layout of the login page. It includes a `csrf_token`, a protection against Cross Site Request Forgeries. A CSRF attack occurs when a malicious Web site contains a link, a form button or some javascript that is intended to perform some action on a Web site, using the credentials of a logged-in user who visits the malicious site in their browser. In Django, for all incoming requests that are not using HTTP GET, HEAD, OPTIONS or TRACE, a CSRF cookie must be present. Using the `csrf_token` in `login.html` provides this

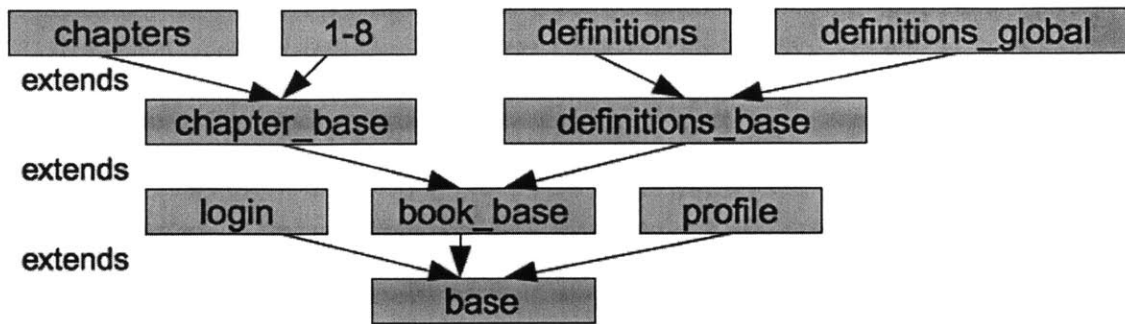


Figure 5-2: The Primer Template Hierarchy

CSRF cookie. The `csrf_token` is used to provide secure login and must be used since login is conducted using an HTML POST request.

`profile.html` inherits from `base.html`. It simultaneously displays the user profile and provides the ability to edit it in an HTML form.

`book_base.html` inherits from `base.html`. It provides the template structure for the chapters and definition templates.

`chapter_base.html` inherits from `book_base.html`. It provides layout specific to the chapters page and individual chapters.

`chapters.html` inherits from `chapter_base.html`. It provides layout for the page used to view all chapters.

Individual chapter templates inherit from `chapter_base.html`. These templates control the layout of the Primer chapters. Definition elements are inserted into HTML as variables; these variables are populated by the `chapter` view before being passed to the template. These templates contain the 6.01 course readings, including images, adapted to HTML5.

`definitions_base.html` inherits from `book_base.html`. It provides layout specific to the definition templates.

`definitions.html` inherits from `definitions_base.html`. It controls the appearance of the page where all definitions and videos featured in the Primer are available.

`definitions_global.html` inherits from `definitions_base.html`. It specifies the

appearance of a single definitions as viewed when selecting them from the `definitions.html` page or from a chapter page.

5.3 Server

The Primer is hosted at a static IP address on the MIT network; this makes the Primer accessible from any device with an internet connection. In particular, the Primer has a simple design to make it readily usable from mobile or tablet devices.

5.3.1 Apache

The site is served by Apache 2.2. Although Django delivers with its own Python development server, the amount of media demanded by an e-textbook requires a more robust HTTP server. Apache supports a variety of features, many implemented as compiled modules which extend the core functionality (`mod_wsgi`, discussed below, is one such module). Apache also features virtual hosting, which allows one Apache installation to serve many different websites. The Primer is hosted using a virtual host; multiple Primers with different addresses could be hosted from the same server using virtual hosting.

5.3.2 `mod_wsgi`

I used `mod_wsgi`, a popular mod for Apache, to host the Primer. `mod_wsgi` is an Apache mod that can host any Python application which supports the Python WSGI interface. `mod_wsgi` for the Primer is used in ‘daemon mode’, which means the application is being run in its own process. This reduces the impact on the normal Apache child processes used to serve up static files and host applications.

Chapter 6

Feedback

The Primer was reviewed by several professionals in the educational technology domain, both for specific feedback and to instigate a general discussion on the future of textbooks. The Primer was also reviewed by students that had and had not taken 6.01 in previous semesters to gauge the utility of the Primer in comparison to the standard 6.01 textbook. The feedback from resulting discussions is reviewed below.

6.1 Primer

Overall, the Primer is well-received. Experts approved of the basic idea of the Primer and of the implementation details associated with the 6.01 e-textbook. The admin user interface and template language was deemed intuitive enough to use to constitute an e-textbook authoring tool. Students regarded the Primer as an improvement upon the standard 6.01 textbook and expressed a desire for more courses, especially introductory courses, to have e-textbooks like the Primer.

Several comments on how to improve the Primer were received. The current implementation of the Primer provides no way to gradually decrease the rating associated with a definition element over the course of a given chapter. This means that if a user becomes familiar with a definition before the chapter ends, the only way to turn the triggered definition element off is to increase their proficiency rating in their user profile in the middle of a chapter. Associating a particular difficulty rating with a

given location of a given definition element is a good idea; the same definition element may have different difficulty ratings in different chapters due to different chapter coefficients. Associating a unique difficulty rating with every instance of a particular definition element in the Primer may constitute micromanaging; nevertheless, a granularity between every single instance and any instance within a chapter would present an improvement. A short comprehension quiz or profile dialog box to adjust profile settings within a chapter was suggested. On the other hand, some experts were happy with the persistence of definition dereferencing throughout a chapter. If in particular profile settings were adjusted automatically by some other extension, some persistence of definition dereferencing was deemed desirable.

Users suggested a way to see the difficulty ratings associated with a particular definition element. It would be beneficial to some users to understand why certain definitions were dereferencing with certain profile settings. In addition, the difficulty ratings could provide feedback to a user on his proficiency in the course relative to the difficulty ratings associated with concepts in the Primer. Users also wanted to be able to save progress through a given chapter, or the ability to use 'bookmarks' - this could be achieved using HTML anchorpoints. The ability to highlight or annotate chapters, and to toggle on and off highlights and annotations, was also deemed desirable.

In addition to lecture videos, users desired animation videos, such as of the progression of an environment as an object was instantiated or of state estimation. These short videos could be cropped from the longer lecture videos or created separately, increasing the frequency of video referencing within the Primer.

6.2 E-Textbooks

E-textbook discussion centered around the differences between the Primer and current e-textbooks, and what the optimal e-textbook would look like. In general, the customized presentation style of the Primer was deemed desirable in e-textbooks, and tailoring e-textbook presentation to a given login or username was a feature left to be desired in the state-of-the-art. E-textbooks for introductory courses would gain the

most from functionality like that of the Primer; e-textbooks for later courses would have an audience with a more uniform background and more familiarity with the language associated with a given domain.

Chapter 7

Contributions

The Primer represents a new advancement in educational technology. Both traditional learning environments and MOOCs could benefit from the ideas and implementation of the Primer. In this section I review the achievements of and possible extensions to the Primer.

7.1 The Primer

7.1.1 The Ideal

The Primer represents a step towards a more personalized web-delivered user experience in either a traditional or open classroom. The Primer functions in all popular web browsers and even from mobile and tablet devices. It incorporates the result of student feedback on the difficulties associated with using available resources into new resources that are designed to meet the needs of the individual student. It allows students to control the way information is presented to them, and to preserve that control in-between learning sessions. Unlike other e-textbooks, the Primer provides an e-textbook with customizations at the user level.

7.1.2 The Implementation

This implementation of the Primer, using 6.01 and 6.01 OCW Scholar materials, represents a usable alternative to the 6.01 course readings. The implementation of the Primer in this thesis merely needs user and user profile setup for students in order to be useful to the administration of 6.01. This implementation also has a simple enough interface to administrators to constitute an e-textbook authoring tool.

7.2 What's Next

Several small extensions to the Primer were conceived but not implemented before the completion of this thesis. A script to automate the creation of usernames and passwords would further prepare the thesis implementation of the Primer for use with a semester of 6.01. Primer elements that behave similar to definition elements but have different properties or difficulty coefficients could be created easily. CSS styling could be added to stylize the Primer or specific subsections of the Primer, such as the exercises, to visually distinguish them from the main content of the e-textbook. A self-serve account and profile creation page would reduce the amount of administrative hassle associated with the Primer. Giving users the opportunity to save their progress through the Primer could further reduce the setup required between learning sessions.

Larger extensions to the Primer could change the fundamental nature of the Primer, but are still worth considering. As pointed out during feedback collection, the ability to specify different difficulty ratings for a given definition element in a given chapter would allow changes to the dereferencing of a given term or video within a given chapter. This would represent an improvement as students are likely to become familiar with a definition element after a limited amount of exposures, and being able to decrease the amount of exposures in a given chapter without changing user profile settings is desirable. This could be accomplished by a quiz or reminder to adjust user profile settings.

Another extension worth consideration is the incorporation of communication between student and educator using the Primer. Primer administrators could leave

reading recommendations for struggling students, reading assignments for the entire class, and annotations regarding adjustments to students' profiles. These features would allow a more personalized interaction surrounding the Primer without incorporating a separate communication medium.

One advantage a traditional textbook still harbors over the Primer is the ability to highlight and annotate text. The Primer would benefit from the inclusion of these features; however, they present a significant enough design challenge to constitute a large extension. Highlights and annotations could persist between sessions, and could be toggled on and off as a user profile setting.

Appendix A

Code

settings.py

```
1 # Django settings for primer project.
2
3 DEBUG = False
4 TEMPLATE_DEBUG = DEBUG
5
6 ADMINS = (
7     # ('Kendra Pugh', 'kpugh@mit.edu'),
8 )
9
10 MANAGERS = ADMINS
11
12 DATABASES = {
13     'default': {
14         'ENGINE': 'django.db.backends.sqlite3', # Add '
15             postgresql_psycopg2', 'mysql', 'sqlite3' or 'oracle'.
16         'NAME': '/srv/www/primer/sqlite3/primer.db',
17             # Or path to database file if using
18             sqlite3.
19         'USER': '', # Not used with sqlite3.
20         'PASSWORD': '', # Not used with sqlite3.
21         'HOST': '', # Set to empty string for
22             localhost. Not used with sqlite3.
```

```

19         'PORT': '',                                     # Set to empty string for
           default. Not used with sqlite3.
20     }
21 }
22
23 # Local time zone for this installation. Choices can be found here:
24 # http://en.wikipedia.org/wiki/List\_of\_tz\_zones\_by\_name
25 # although not all choices may be available on all operating systems.
26 # On Unix systems, a value of None will cause Django to use the same
27 # timezone as the operating system.
28 # If running in a Windows environment this must be set to the same as
   your
29 # system time zone.
30 TIMEZONE = 'America/New_York'
31
32 # Language code for this installation. All choices can be found here:
33 # http://www.i18nguy.com/unicode/language-identifiers.html
34 LANGUAGECODE = 'en-us'
35
36 SITE_ID = 1
37
38 # If you set this to False, Django will make some optimizations so as
   not
39 # to load the internationalization machinery.
40 USE_I18N = True
41
42 # If you set this to False, Django will not format dates, numbers and
43 # calendars according to the current locale.
44 USE_L10N = True
45
46 # If you set this to False, Django will not use timezone-aware datetimes
   .
47 USE_TZ = True
48
49 # Absolute filesystem path to the directory that will hold user-uploaded
   files.

```



```

50 # Example: "/home/media/media.lawrence.com/media/"
51 MEDIA_ROOT = ''
52
53 # URL that handles the media served from MEDIA_ROOT. Make sure to use a
54 # trailing slash.
55 # Examples: "http://media.lawrence.com/media/", "http://example.com/
    media/"
56 MEDIA_URL = ''
57
58 # Absolute path to the directory static files should be collected to.
59 # Don't put anything in this directory yourself; store your static files
60 # in apps' "static/" subdirectories and in STATICFILES_DIRS.
61 # Example: "/home/media/media.lawrence.com/static/"
62 STATIC_ROOT = ''
63
64 # URL prefix for static files.
65 # Example: "http://media.lawrence.com/static/"
66 STATIC_URL = '/static/'
67
68 # Additional locations of static files
69 STATICFILES_DIRS = (
70     # Put strings here, like "/home/html/static" or "C:/www/django/
    static".
71     # Always use forward slashes, even on Windows.
72     # Don't forget to use absolute paths, not relative paths.
73 )
74
75 # List of finder classes that know how to find static files in
76 # various locations.
77 STATICFILES_FINDERS = (
78     'django.contrib.staticfiles.finders.FileSystemFinder',
79     'django.contrib.staticfiles.finders.AppDirectoriesFinder',
80     # 'django.contrib.staticfiles.finders.DefaultStorageFinder',
81 )
82
83 # Make this unique, and don't share it with anybody.

```

```

84 SECRET_KEY = omitted
85
86 # List of callables that know how to import templates from various
      sources.
87 TEMPLATeloadERS = (
88     'django.template.loaders.filesystem.Loader',
89     'django.template.loaders.app_directories.Loader',
90 #     'django.template.loaders.eggs.Loader',
91 )
92
93 MIDDLEWARE_CLASSES = (
94     'django.middleware.common.CommonMiddleware',
95     'django.contrib.sessions.middleware.SessionMiddleware',
96     'django.middleware.csrf.CsrfViewMiddleware',
97     'django.contrib.auth.middleware.AuthenticationMiddleware',
98     'django.contrib.messages.middleware.MessageMiddleware',
99     # Uncomment the next line for simple clickjacking protection:
100    # 'django.middleware.clickjacking.XFrameOptionsMiddleware',
101 )
102
103 ROOT_URLCONF = 'primer.urls'
104
105 # Python dotted path to the WSGI application used by Django's runserver.
106 WSGI_APPLICATION = 'primer.wsgi.application'
107
108 TEMPLATE_DIRS = (
109     # Put strings here, like "/home/html/django_templates" or "C:/www/
      django/templates".
110     # Always use forward slashes, even on Windows.
111     # Don't forget to use absolute paths, not relative paths.
112     '/srv/www/primer/templates'
113 )
114
115 INSTALLED_APPS = (
116     'django.contrib.auth',
117     'django.contrib.contenttypes',

```

```

118     'django.contrib.sessions',
119     #'django.contrib.sites',
120     'django.contrib.messages',
121     'django.contrib.staticfiles',
122     # Uncomment the next line to enable the admin:
123     'django.contrib.admin',
124     # Uncomment the next line to enable admin documentation:
125     # 'django.contrib.admindocs',
126     'book',
127 )
128
129 AUTHPROFILEMODULE = 'book.UserProfile'
130
131 # A sample logging configuration. The only tangible logging
132 # performed by this configuration is to send an email to
133 # the site admins on every HTTP 500 error when DEBUG=False.
134 # See http://docs.djangoproject.com/en/dev/topics/logging for
135 # more details on how to customize your logging configuration.
136 LOGGING = {
137     'version': 1,
138     'disable_existing_loggers': False,
139     'filters': {
140         'require_debug_false': {
141             '()': 'django.utils.log.RequireDebugFalse'
142         }
143     },
144     'handlers': {
145         'mail_admins': {
146             'level': 'ERROR',
147             'filters': ['require_debug_false'],
148             'class': 'django.utils.log.AdminEmailHandler'
149         }
150     },
151     'loggers': {
152         'django.request': {
153             'handlers': ['mail_admins'],

```

```

154         'level': 'ERROR',
155         'propagate': True,
156     },
157 }
158 }

```

urls.py

```

1 from django.conf.urls import patterns, include, url
2
3 # Uncomment the next two lines to enable the admin:
4 from django.contrib import admin
5 admin.autodiscover()
6
7 urlpatterns = patterns('http://flahp.mit.edu/',
8
9                     url(r'^book/', include('book.urls')),
10                    url(r'^admin/', include(admin.site.urls)),
11 )

```

models.py

```

1 from django.db import models
2 from django.db.models.signals import post_save
3 from django.contrib.auth.models import User
4
5
6 class UserProfile(models.Model):
7     user = models.OneToOneField(User)
8
9     definitionRating = models.IntegerField()
10    programmingBackground = models.IntegerField()
11    systemsBackground = models.IntegerField()
12    circuitsBackground = models.IntegerField()
13    probabilityBackground = models.IntegerField()
14
15    def __unicode__(self):
16        return str(self.user) + " Profile"

```

```

17
18 def create_user_profile(sender, instance, created, **kwargs):
19     if created:
20         UserProfile.objects.create(user=instance)
21
22         post_save.connect(create_user_profile, sender=User)
23
24 class Chapter(models.Model):
25     def __unicode__(self):
26         return "Chapter " + str(self.id)
27     advanceMultiplier = models.DecimalField(max_digits=4, decimal_places
28         =3)
29
30 class PrimerElement(models.Model):
31     name = models.CharField(max_length=100)
32     chapter = models.ManyToManyField(Chapter)
33     programmingThreshold = models.IntegerField()
34     systemsThreshold = models.IntegerField()
35     circuitsThreshold = models.IntegerField()
36     probabilityThreshold = models.IntegerField()
37     baseline = models.CharField(max_length=1000)
38     triggered = models.CharField(max_length=1000)
39
40     def __unicode__(self):
41         return self.name
42
43 class DefinitionElement(PrimerElement):
44     definitionThreshold = models.IntegerField()

```

views.py

```

1
2 from django.http import HttpResponseRedirect, HttpResponse
3 from django.core.urlresolvers import reverse
4 from book.models import Chapter, UserProfile, PrimerElement,
5     DefinitionElement
6 from django.shortcuts import get_object_or_404, render_to_response

```

```

6 from django.template import RequestContext
7 from django.contrib.auth import authenticate, login, logout
8
9 def index(request):
10     return render_to_response('book/index.html')
11
12 def definitions(request, def_element):
13     primerElement=DefinitionElement.objects.get(name=def_element)
14     chapters = [x.pk for x in primerElement.chapter.all()]
15     name = primerElement.baseline
16     definition = primerElement.triggered
17     return render_to_response('book/definitions/definitions_global.html',
18         {
19             'chapters' : chapters,
20             'name' : name,
21             'definition' : definition})
22
23 def chapter(request, chapter_id):
24     if not request.user.is_authenticated():
25         return render_to_response('book/login.html', {
26             'error_message': 'you must be logged in to use the
27             Primer'
28         }, context_instance=(RequestContext(request)))
29     else:
30         primerDictionary = {}
31         primerChapter = Chapter.objects.get(pk=chapter_id)
32         currentProfile = UserProfile.objects.get(user=request.
33             user)
34         mul = primerChapter.advanceMultiplier
35         for element in DefinitionElement.objects.filter(chapter=
36             primerChapter):
37             if ((element.definitionThreshold * mul > \
38                 currentProfile.definitionRating) or
39                 (element.programmingThreshold * mul > \
40                 currentProfile.programmingBackground) or

```

```

38     (element.systemsThreshold * mul > \
39         currentProfile.systemsBackground) or
40     (element.circuitsThreshold * mul > \
41         currentProfile.circuitsBackground)
42         or
43     (element.probabilityThreshold * mul > \
44         currentProfile.probabilityBackground
45         )):
46         primerDictionary[element.name] = \
47             element.baseline + " (" + \
48             element.triggered + ") "
49         else:
50             primerDictionary[element.name] = \
51                 '<a href=" ../.. / definitions /'+ \
52                 element.name + '/'>' + element.baseline + \
53                 '</a>'
54
55         return render_to_response('book/chapter/'+ \
56             str(chapter_id)+' .html',
57             primerDictionary)
58
59 def site_login(request):
60     if request.user.is_authenticated():
61         return render_to_response('book/login.html', {
62             'error_message': str(request.user) + ' is already logged
63             in <a href=" ../logout/">' +
64             'logout</a> <a href=" ../chapter/1/">first
65             chapter<a>'
66             }, context_instance=RequestContext(request))
67     else:
68         return render_to_response('book/login.html',
69             context_instance=RequestContext(request))
70
71 def logging_in(request):
72     if request.user.is_authenticated():
73         return render_to_response('book/login.html', {
74             'error_message': str(request.user) + ' is already logged
75             in <a href=" ../logout/">logout</a>'

```

```

68         }, context_instance=(RequestContext(request)))
69     username = request.POST['username']
70     password = request.POST['password']
71     user = authenticate(username=username, password=password)
72     if user is not None:
73         if user.is_active:
74             login(request, user)
75             return HttpResponseRedirect(reverse('book.views.
76                 profile', ))
77         else:
78             return render_to_response('book/login.html', {
79                 'error_message' : 'inactive user'
80             }, context_instance=RequestContext(request))
81     else:
82         return render_to_response('book/login.html', {
83             'error_message': "Invalid Login",
84         }, context_instance=RequestContext(request))
85 def site_logout(request):
86     logout(request)
87     return HttpResponseRedirect("Congratulations! You are logged out. <a
88         href='../login/'>login</a>")
89 def profile(request):
90     user = request.user
91     currentProfile = UserProfile.objects.get(user=request.user)
92     return render_to_response('book/profile.html', {
93         'user' : user,
94         'currentProfile' : currentProfile,
95     }, context_instance=RequestContext(request))
96
97 def submit_profile(request):
98     user = request.user
99     currentProfile = UserProfile.objects.get(user=user)
100
101     currentProfile.definitionRating = request.POST['definitionRating']

```



```

102     currentProfile.programmingBackground = request.POST[ '
        programmingBackground ' ]
103         currentProfile.systemsBackground = request.POST[ '
            systemsBackground ' ]
104         currentProfile.circuitsBackground = request.POST[ '
            circuitsBackground ' ]
105     currentProfile.probabilityBackground = request.POST[ '
        probabilityBackground ' ]
106     currentProfile.save()
107     return render_to_response( 'book/profile.html', {
108         'user' : user,
109         'currentProfile' : currentProfile,
110     }, context_instance=RequestContext(request))

```

book/urls.py

```

1 from django.conf.urls import patterns, include, url
2 from django.views.generic import ListView
3 from book.models import Chapter, DefinitionElement
4
5 # Uncomment the next two lines to enable the admin:
6 from django.contrib import admin
7 admin.autodiscover()
8
9 urlpatterns = patterns( 'book.views',
10                         url( r'^$', 'index' ),
11                         url( r'^login/$', 'site_login' ),
12                         url( r'^logging_in/$', 'logging_in' ),
13                         url( r'^logout/$', 'site_logout' ),
14                         url( r'^chapter/(?P<chapter_id>\d+)/$', 'chapter' )
15                         ,
16                         url( r'^chapters/$',
17                             ListView.as_view( queryset=Chapter.objects.all
18                                             ( ),
19                                             context_object_name='
20                                             chapter_list',

```

```

18         template_name='book/chapter/
19             chapters.html')),
20     url(r'^profile/$', 'profile'),
21     url(r'^submit_profile/$', 'submit_profile'),
22     url(r'^definitions/(?P<def_element>\w+)/$', '
23         definitions'),
24     url(r'^definitions/$',
25         ListView.as_view(queryset=DefinitionElement.
26             objects.all(),
27             context_object_name='
28                 definitions_list',
29             template_name='book/
30                 definitions/definitions.
31                 html')),
32     url(r'^admin/', include(admin.site.urls)),
33 )

```

admin.py

```

1 from book.models import Chapter, DefinitionElement, UserProfile
2 from django.contrib import admin
3
4 admin.site.register(Chapter)
5 admin.site.register(DefinitionElement)
6 admin.site.register(UserProfile)

```

base.html

```

1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4     <link rel="stylesheet" href="style.css" />
5     <title>{% block title %}The Primer{% endblock title %}</title>
6 </head>
7
8 <body>
9     <div id="masthead">

```

```

10     <a href="/book/login/">Login</a> <a href="/book/profile/">Profile<
      /a> <a href="/book/chapters/">Chapters</a> <a href="/book/
      definitions/">Definitions</a>
11 </div>
12 <div id="content">
13     {% autoescape off %}
14     {% block content %}{% endblock %}
15     {% endautoescape %}
16 </div>
17 </body>
18 </html>

```

book_base.html

```

1 {% extends "base.html" %}

```

login.html

```

1 {% extends "base.html" %}
2
3 {% block content %}
4 <h1>Login Page</h1>
5
6 {% if error_message %}<p><strong>{{ error_message }}</strong></p>{%
  endif %}
7
8 <form action="/book/logging_in/" method="post">
9   {% csrf_token %}
10  Username: <input type="text" name="username"><br />
11  Password: <input type="password" name="password"><br />
12  <input type="submit" value="Login" />
13 </form>
14 {% endblock %}

```

profile.html

```

1 {% extends "base.html" %}
2
3 {% block content %}

```

```

4 <h1>Welcome to the Primer</h1>
5 <p>You are logged in as {{user.username}}</p>
6 Edit your profile below. 0 is a novice, 10 is an expert.
7 <form action="/book/submit_profile/" method="post">
8 {% csrf_token %}
9
10 Definition Rating: <input type="text" name="definitionRating" value={{
    currentProfile.definitionRating}} /><br />
11 Programming Background: <input type="text" name="programmingBackground"
    value={{currentProfile.programmingBackground}} /><br />
12 Systems Background: <input type="text" name="systemsBackground" value={{
    currentProfile.systemsBackground}} /><br />
13 Circuits Background: <input type="text" name="circuitsBackground" value
    ={{currentProfile.circuitsBackground}} /><br />
14 Probability Background: <input type="text" name="probabilityBackground"
    value={{currentProfile.probabilityBackground}} /><br />
15 <input type="submit" value="Submit" />
16 </form>
17
18 <p><a href=" ../logout/">logout here</a></p>
19 <p><a href=" ../chapter/1/">here is the first chapter</a></p>
20 {% endblock %}

```

definitions_base.html

```

1 {% extends "book/book_base.html" %}
2
3 {% block title %}The Primer{% endblock %}
4
5 {% block content %}
6 <h1>{{block word}}Your Word goes here {% endblock %}</h1>
7
8 <p>{{block definition}}Your definition goes here{% endblock %}</p>
9
10 <br \>
11 {% for chapter in chapters %}

```

```

12 <a href='../.. / chapter / {{chapter}} /'>Back to Chapter {{chapter}}</a><br
    />
13 {% endfor %}
14
15 {% endblock %}

```

definitions.html

```

1 {% extends "book/book_base.html" %}
2
3 {% block title %}Primer Definitions{% endblock %}
4
5 {% block content %}
6
7 {% if definitions_list %}
8 <ul>
9   {% for definition in definitions_list %}
10     <li><a href="/book/definitions / {{ definition.name }} /">{{ definition
        .baseline }}</a></li>
11   {% endfor %}
12 </ul>
13 {% else %}
14 <p>No definitions are available.</p>
15 {% endif %}
16
17 {% endblock %}

```

definitions_global.html

```

1 {% extends "book/definitions / definitions_base.html" %}
2 {% block word %}
3   {{name}}
4 {% endblock %}
5 {% block definition %}
6   {{definition}}
7 {% endblock %}

```

chapter_base.html

```

1 {% extends "book/book_base.html" %}
2
3 {% block title %}Primer Chapter{% endblock %}

```

chapters.html

```

1 {% extends "book/chapter/chapter_base.html" %}
2
3 {% block title %}Primer Chapters{% endblock %}
4
5 {% block content %}
6
7 {% if chapter_list %}
8 <ul>
9   {% for chapter in chapter_list %}
10    <li><a href="/book/chapter/{{ chapter.id }}/">Chapter {{ chapter.id
11      }}</a></li>
12  {% endfor %}
13 </ul>
14 {% else %}
15 <p>No chapters are available.</p>
16 {% endif %}
17 {% endblock %}

```

Example chapter - 8.html

```

1 {% extends "book/chapter/chapter_base.html" %}
2 {% block content %}
3
4 <h1>Long-term decision-making and search</h1>
5
6 <p>In the lab exercises of this course, we have implemented several
7 brains for our robots. We used wall-following to navigate through the
8 world and we used various linear {{controller}}s to drive down the hall
9 and to control the robot head. In the first case, we just wrote a
10 program that we hoped would do a good job. When we were studying

```

11 linear controllers , we, as designers , made models of the controller 's
12 behavior in the world and tried to prove whether it would behave in
13 the way we wanted it to, taking into account a longer-term pattern of
14 behavior.</p>

15

16 <p>Often , we will want a system to generate complex long-term patterns
of

17 behavior , but we will not be able to write a simple control rule to
18 generate those behavior patterns. In that case , we'd like the system
19 to evaluate alternatives for itself , but instead of evaluating single
20 actions , it will have to evaluate whole sequences of actions , deciding
21 whether they're a good thing to do given the current state of the
22 world. </p>

23

24 <p>Let 's think of the problem of navigating through a city , given a road
25 map, and knowing where we are. We can't usually decide whether to
26 turn left at the next intersection without deciding on a whole path.</p>

27

28 <p>As always , the first step in the process will be to come up with a
29 formal model of a real-world problem that abstracts away the
30 irrelevant detail. So, what, exactly , is a path? The car we're
31 driving will actually follow a trajectory through continuous
32 space(time) , but if we tried to plan at that level of detail we would
33 fail miserably. Why? First , because the space of possible
34 trajectories through two-dimensional space is just too enormous.
35 Second , because when we're trying to decide which roads to take , we
36 don't have the information about where the other cars will be on
37 those roads , which will end up having a huge effect on the detailed
38 trajectory we'll end up taking. </p>

39

40 <p>So, we can divide the problem into two levels: planning in advance
41 which turns we'll make at which intersections , but deciding 'on-line' ,
42 while we're driving , exactly how to control the steering wheel and the
43 gas to best move from intersection to intersection , given the current
44 circumstances (other cars , stop-lights , etc.)</p>

45

46 <p>We can make an abstraction of the driving problem to include
47 road intersections and the way they're connected by roads. Then,
48 given a start and a goal intersection, we could consider all possible
49 paths between them, and choose the one that is best.</p>
50
51 <p>What criteria might we use to evaluate a path? There are all sorts
of
52 reasonable ones: distance, time, gas mileage, traffic-related
53 frustration, scenery, etc. Generally speaking, the approach we'll
54 outline below can be extended to any criterion that is additive: that
55 is, your happiness with the whole path is the sum of your happiness
56 with each of the segments. We'll start with the simple
57 criterion of wanting to find a path with the fewest "steps"; in
58 this case, it will be the path that traverses the fewest
59 intersections. Then in the Uniform Cost section we will
60 generalize our methods to handle problems where different actions have
61 different costs.</p>
62
63 <p>One possible algorithm for deciding on the best path through a map of
the
64 road intersections in this (very small) world
65 <p>
66 <p></p>
67
68 <p>
69 would be to enumerate all the paths, evaluate each
70 one according to our criterion, and then return the best one. The
71 problem is that there are lots of paths. Even in our little
72 domain, with 9 intersections, there are 210 paths from the
73 intersection labeled S to the one labeled G. </p>
74
75 <p>We can get a much better handle on this problem, by formulating it as
76 an instance of a graph search (or a "state-space search") problem,
77 for which there are simple algorithms that perform well. </p>
78
79 <p><h2>State-space search</h2>

80 We'll model a state-space search problem formally as

- 81
- 82 a (possibly infinite) set of states the system can be in;
- 83 a starting state, which is an element of the set of states
- ;
- 84 a goal test, which is a procedure that can be applied to
- 85 any state, and returns <tt>True</tt> if that state can serve as a
- 86 goal;(Although in many cases we have a particular goal state
- 87 (such as the intersection in front of my house), in other cases,
- 88 we may have the goal of going to any gas station, which can be
- 89 satisfied by many different intersections.)
- 90 a successor function, which takes a state and an action as
- 91 input, and returns the new state that will result from taking the
- 92 action in the state; and
- 93 a legal action list,
- 94 which is just a list of actions that can be legally executed in this
- domain.
- 95

96 The decision about what constitutes an action is a

97 modeling decision. It could be to drive to the next intersection, or

98 to drive a meter, or a variety of other things, depending on the

99 domain. The only requirement is that it terminate in a well-defined

100 next state (and that, when it is time to execute the plan, we will

101 know how to execute the action.) </p>

102

103 <p>We can think of this model as specifying a <i>labeled graph</i> (in

the

104 computer scientist's sense), in which the states are the {{node}}s,

105 action specifies which of the arcs leading out of a node is to be

106 selected, and the successor function specifies the node at the end of

107 each of the arcs.</p>

108

109 <p>So, for the little world above, we might make a model in

110 which

111

112 The set of states is the intersections <tt>{'S', 'A', 'B', 'C', 'D'
', 'E', 'F', 'G', 'H'}</tt>.

113 The starting state is <tt>'S'</tt>.

114 The {{goal}} test is something like:

115 <pre>
116 lambda x: x == 'H'
117 </pre>

118

119 The {{legal_action_list}} in this domain are the numbers <tt>0</tt>
, <tt>1</tt>,
120 <tt>...</tt>, <tt>n-1</tt>, where <tt>n</tt> is the maximum number of
successors
121 in any of the states.

122 The map can be defined using a dictionary:

123 <pre>
124 map1 = {'S' : ['A', 'B'],
125 'A' : ['S', 'C', 'D'],
126 'B' : ['S', 'D', 'E'],
127 'C' : ['A', 'F'],
128 'D' : ['A', 'B', 'F', 'H'],
129 'E' : ['B', 'H'],
130 'F' : ['C', 'D', 'G'],
131 'H' : ['D', 'E', 'G'],
132 'G' : ['F', 'H']}
133 </pre>

134 where each key is a state, and its value is a list of states that can
135 be reached from it in one step.</p>

136

137 <p>Now we can define the {{successor_function}} as

138 <pre>
139 def map1successors(s, a):
140 return map1[s][a]
141 </pre>

142 but with an additional test to be sure that if we attempt to take an
action
143 that doesn't exist in <tt>s</tt>, it just results in state <tt>s</tt>.

144 So, for example, the successor reached from state `'A'` by taking
145 action `l` is state `'C'`.

146
147 `</p>`

148
149 `<p>`We can think of this structure as defining a *search tree*, like
150 this:

151 `<p></p>`

152
153 It has the starting state, `S`, at the
154 root node, and then each node has its successor states as children.
155 Layer `k` of this tree contains all possible paths
156 of length `k` through the graph. `</p>`

157
158 `<p><h3>`Representing search trees`</h3>`

159 We will need a way to represent the tree as a Python data structure as
160 we construct it during the search process. We will start by defining
161 a class to represent a *search node*, which is one of the circles
162 in the tree.`</p>`

163
164 `<p>`Each search node represents:

165 ``
166 `` The state of the node;
167 `` the action that was taken to arrive at the node; and
168 `` the search node *from which* this node can be reached.
169 ``

170 We will call the node from which a node can be reached its *parent*
171 node. So, for example, in the figure below

172 `<p></p>`

173
174 we will represent the node with double circles around it with its
175 state, `'D'`, the action that reached it, `l`, and its
parent

176 node, which is the node labeled `'B'` above it.`</p>`

177
178 `<p>`Note that *states* and *nodes* are not the same thing!`` In this

179 tree, there are many nodes labeled by the same state; they represent
180 different paths to and through the state.</p>

181

182 <p>Here is a Python class representing a search {{node}}. It's pretty
183 straightforward.

184 <pre>

```
185 class SearchNode:
186     def __init__(self, action, state, parent):
187         self.state = state
188         self.action = action
189         self.parent = parent
```

190 </pre></p>

191

192 <p>There are a couple of other useful methods for this class. First,
the

193 <tt>path</tt> method, returns a list of pairs <tt>(a, s)</tt>
corresponding

194 to the path starting at the top (root) of the tree, going down to this
195 node. It works its way up the tree, until it reaches a node whose
196 parent is <tt>None</tt>.

197 <pre>

```
198     def path(self):
199         if self.parent == None:
200             return [(self.action, self.state)]
201         else:
202             return self.parent.path() + [(self.action, self.state)]
```

203 </pre>

204 The path corresponding to our double-circled node is <tt>((None, 'S'),
(1, 'B'), (1, 'D'))</tt>.</p>

205

206 <p>Another helper method that we will find useful is the <tt>inPath</tt>
207 method, which takes a state, and returns <tt>True</tt> if the state
208 occurs anywhere in the path from the root to the node.

209 <pre>

```
210     def inPath(self, s):
211         if s == self.state:
```

```

212         return True
213     elif self.parent == None:
214         return False
215     else:
216         return self.parent.inPath(s)

```

217 </pre></p>

218

219 <p><h3>Basic search algorithm</h3></p>

220

221 <p>We'll describe a sequence of search algorithms of increasing
222 sophistication and efficiency. An ideal algorithm will take a problem
223 description as input and return a path from the start to a goal state,
224 if one exists, and return None, if it does not. Some algorithms will
225 not be capable of finding a path in all cases.</p>

226

227 <p>How can we systematically search for a path to the goal? There are
228 two plausible strategies:

229

230 Start down a path, keep trying to extend it until you get stuck, in
231 which case, go back to the last choice you had, and go a different
232 way. This is how kids often solve mazes. We'll call it depth-first
 search.

233 Go layer by layer through the tree, first considering all paths
234 of length 1, then all of length 2, etc. We'll call this breadth-
 first search.

235 </p>

236

237 <p>Both of the search strategies described above can be implemented
 using

238 a procedure with this basic structure:

239 <pre>

```

240 def search(initialState, goalTest, actions, successor):
241     if goalTest(initialState):
242         return [(None, initialState)]
243     agenda = EmptyAgenda()
244     add(SearchNode(None, initialState, None), agenda)

```

```

245 while not empty(agenda):
246     parent = getElement(agenda)
247     for a in actions:
248         newS = successor(parent.state, a)
249         newN = SearchNode(a, newS, parent)
250         if goalTest(newS):
251             return newN.path()
252         else:
253             add(newN, agenda)
254     return None

```

255 </pre>

256 We start by checking to see if the initial state is a {{goal}} state.

 If

257 so, we just return a path consisting of the initial state.</p>

258

259 <p>Otherwise, we have real work to do. We make the root node
 of

260 the tree. It has no parent, and there was no action leading to it, so

261 all it needs to specify is its state, which is <tt>initialState</tt>, so

262 it is created with

263 <pre>

```

264 SearchNode(None, initialState, None)

```

265 </pre></p>

266

267 <p>During the process of constructing the search tree, we will use a
 data

268 structure, called an agenda, to keep track of which {{node}}s
 in the

269 partially-constructed tree are on the fringe, ready to be expanded, by

270 adding their children to the tree.

271 We initialize the agenda to contain the root note. Now, we enter a

272 loop that will run until the agenda is empty (we have no more paths to

273 consider), but could stop sooner.</p>

274

275 <p>Inside the loop, we select a node from the {{agenda}} (more on how we
276 decide which one to take out in a bit) and <i>expand it</i>. To expand

277 a node, we determine which actions can be taken from the state that is
278 stored in the node, and ~~visit~~ the ~~successor~~ states
that
279 can be reached via the actions. ~~</p>~~
280
281 ~~<p>~~When we visit a state, we make a new search node (~~newN~~, in
the
282 code) that has the node we are in the process of expanding as the
283 parent, and that remembers the state being visited and the action that
284 brought us here from the parent.~~</p>~~
285
286 ~~<p>~~Next, we check to see if the new state satisfies the goal test. If
it
287 does, we're done! We return the path associated with the new node.~~</p>~~
288
289 ~~<p>~~If this state it doesn't satisfy the ~~{{goal}}~~ test, then we add the
new
290 ~~{{node}}~~ to the ~~{{agenda}}~~. We continue this process until we find a
goal
291 state or the agenda becomes empty. This is not quite yet an
292 algorithm, though, because we haven't said anything about what it
293 means to add and extract nodes from the agenda. And, we'll find, that
294 it will do some very stupid things, in its current form.~~</p>~~
295
296 ~~<p>~~We'll start by curing the stupidities, and then return to the
question
297 of how best to select nodes from the agenda.~~</p>~~
298
299 ~~<p><h3>~~Basic pruning, or How not to be completely stupid~~</h3></p>~~
300
301 ~~<p>~~If you examine the full search tree, you can see that some of the
302 paths it contains are completely ridiculous. It can never be
303 reasonable, if we're trying to find the shortest path between two
304 states, to go back to a state we have previously visited on that same
305 path. So, to avoid trivial infinite loops, we can adopt the following
rule:~~</p>~~

306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339

<p>

Don't consider any path that visits the same state twice.

<p>If we can apply this rule, then we will be able to remove a number of branches from the tree, as shown here:

<p></p>

It is relatively straightforward to modify our code to implement this rule:

<pre>

```
def search(initialState, goalTest, actions, successor):
    if goalTest(initialState):
        return [(None, initialState)]
    agenda = [SearchNode(None, initialState, None)]
    while agenda != []:
        parent = getElement(agenda)
        for a in actions:
            newS = successor(parent.state, a)
            newN = SearchNode(a, newS, parent)
            if goalTest(newS):
                return newN.path()
            elif parent.inPath(newS):} /ETEX
            pass
        else:
            add(newN, agenda)
    return None
```

</pre>

We've added code to our basic algorithm.

It just checks to see whether the current state already exists on the path to the node we're expanding and, if so, it doesn't do anything with it. </p>

340 <p>The next pruning rule doesn't make a difference in the current domain
 341 but can have a big effect in other domains:
 342 <p>
 343 If there are multiple actions that lead from a state <tt>r</tt> to a
 state
 344 <tt>s</tt>, consider only one of them.
 345 <p>
 346 To handle this in the code, we have to keep track of which new states
 347 we have reached in expanding this node, and if we find another way to
 348 reach one of those states, we just ignore it. The changes to the code
 349 for implementing this rule are shown:
 350 <pre>
 351 def search(initialState, goalTest, actions, successor):
 352 if goalTest(initialState):
 353 return [(None, initialState)]
 354 agenda = [SearchNode(None, initialState, None)]
 355 while agenda != []:
 356 parent = getElement(agenda)
 357 newChildStates = []
 358 for a in actions:
 359 newS = successor(parent.state, a)
 360 newN = SearchNode(a, newS, parent)
 361 if goalTest(newS):
 362 return newN.path()
 363 elif newS in newChildStates:
 364 pass
 365 elif parent.inPath(newS):
 366 pass
 367 else:
 368 newChildStates.append(newS)
 369 add(newN, agenda)
 370 return None
 371 </pre>
 372 Each time we pick a new node to expand, we make a new empty list, <tt>
 newChildStates</tt>, and keep track of all of the new states we have

373 reached from this node.</p>
374
375 <p>Now, we have to think about how to extract nodes from the agenda.</p>
376
377 <p><h3>Stacks and Queues</h3>
378 In designing algorithms, we frequently make use of two simple data
379 structures: stacks and queues. You can think of them both as
380 abstract data types that support two operations: <tt>push</tt> and <tt>
pop</tt>. The <tt>push</tt> operation adds an element to the stack
or queue,
381 and the <tt>pop</tt> operation removes an element. The difference
382 between a stack and a queue is what element you get back when you do a
383 <tt>pop</tt>.
384
385 stack: When you <tt>pop</tt> a stack, you get back the
element
386 that you most recently put in. A stack is also called a LIFO, for
387 last in, first out.
388 queue: When you <tt>pop</tt> a queue, you get back the
element
389 that you put in earliest. A queue is also called a FIFO, for
390 first in, first out.
391 </p>
392
393 <p>In Python, we can use lists to represent both stacks and queues. If
394 <tt>data</tt> is a list, then <tt>data.pop(0)</tt> removes the first
element
395 from the list and returns it, and <tt>data.pop()</tt> removes the last
396 element and returns it. </p>
397
398 <p>Here is a class representing stacks as lists. It always adds new
399 elements to the end of the list, and pops items off of the same end,
400 ensuring that the most recent items get popped off first.
401 <pre>
402 class Stack:
403 def __init__(self):

```

404     self.data = []
405     def push(self, item):
406         self.data.append(item)
407     def pop(self):
408         return self.data.pop()
409     def isEmpty(self):
410         return self.data is []
411 </pre> </p>

```

412
413 <p>Here is a class representing stacks as lists. It always adds new
414 elements to the end of the list, and pops items off of the front,
415 ensuring that the oldest items get popped off first.

```

416 <pre>
417 class Queue:
418     def __init__(self):
419         self.data = []
420     def push(self, item):
421         self.data.append(item)
422     def pop(self):
423         return self.data.pop(0)
424     def isEmpty(self):
425         return self.data is []
426 </pre></p>

```

427
428 <p>We will use stacks and queues to implement our search algorithms.</p>
429

430 <p><h3>Depth-First Search</h3>

431 Now we can easily describe `{{depth_first_search}}` by saying that it's
432 an instance of the generic search procedure described above, but in
433 which the agenda is a **stack**: that is, we always expand the
434 node we most recently put into the agenda.</p>

435
436 <p>The code listing below shows our implementation of depth-first
437 search.

```

438 <pre>
439 def depthFirstSearch(initialState, goalTest, actions, successor):

```

```

440     agenda = Stack()
441     if goalTest(initialState):
442         return [(None, initialState)]
443     agenda.push(SearchNode(None, initialState, None))
444     while not agenda.isEmpty():
445         parent = agenda.pop()
446         newChildStates = []
447         for a in actions:
448             newS = successor(parent.state, a)
449             newN = SearchNode(a, newS, parent)
450             if goalTest(newS):
451                 return newN.path()
452             elif newS in newChildStates:
453                 pass
454             elif parent.inPath(newS):
455                 pass
456             else:
457                 newChildStates.append(newS)
458                 agenda.push(newN)
459     return None

```

460 </pre>

461 You can see several operations on the `agenda`. We:

462

463 Create an empty `Stack` instance, and let that be the agenda

464 Push the initial node onto the agenda.

465 Test to see if the agenda is empty.

466 Pop the node to be expanded off of the agenda.

467 Push newly visited nodes onto the agenda.

468

469 Because the agenda is an instance of the `Stack` class,
subsequent

470 operations on the agenda ensure that it will act like a stack, and
471 guarantee that children of the most recently expanded node will be
472 chosen for expansion next.</p>

473

```

474 <p>So, let's see how this search method behaves on our city map, with
      start
475 state <TT>S</TT> and goal state <TT>F</TT>. Here is a trace of the
      algorithm (you
476 can get this in the code we distribute by setting <tt>verbose = True</tt>
      >
477 before you run it.)
478 <pre>
479 depthFirst('S', lambda x: x == 'F', map1LegalActions, map1successors)
480 agenda: Stack([S])
481     expanding: S
482 agenda: Stack([S-0->A, S-1->B])
483     expanding: S-1->B
484 agenda: Stack([S-0->A, S-1->B-1->D, S-1->B-2->E])
485     expanding: S-1->B-2->E
486 agenda: Stack([S-0->A, S-1->B-1->D, S-1->B-2->E-1->H])
487     expanding: S-1->B-2->E-1->H
488 agenda: Stack([S-0->A, S-1->B-1->D, S-1->B-2->E-1->H-0->D, S-1->B-2->E
      -1->H-2->G])
489     expanding: S-1->B-2->E-1->H-2->G
490 8 states visited
491 [(None, 'S'), (1, 'B'), (2, 'E'), (1, 'H'), (2, 'G'), (0, 'F')]
492 </pre></p>
493
494 <p>You can see that in this world, the search never needs to
495 "backtrack", that is, to go back and try expanding an older path on
496 its agenda. It is always able to push the current path forward until
497 it reaches the goal.
498 Here is the search tree generated during the depth-first search process.
499 <p></p>
500
501
502 <p>Here is another city map (it's a weird city, we know,
503 but maybe a bit like trying to drive in Boston):
504 <p></p>
505

```

506 In this city ,
507 depth-first search behaves a bit differently (trying to go from `S`/
`T` to
508 `D` this time):
509

```
  

510 depthFirst('S', lambda x: x == 'D', map2LegalActions, map2successors)  

511 agenda: Stack([S])  

512     expanding: S  

513 agenda: Stack([S-0->A, S-1->B])  

514     expanding: S-1->B  

515 agenda: Stack([S-0->A, S-1->B-1->E, S-1->B-2->F])  

516     expanding: S-1->B-2->F  

517 agenda: Stack([S-0->A, S-1->B-1->E, S-1->B-2->F-1->G])  

518     expanding: S-1->B-2->F-1->G  

519 agenda: Stack([S-0->A, S-1->B-1->E])  

520     expanding: S-1->B-1->E  

521 agenda: Stack([S-0->A])  

522     expanding: S-0->A  

523 7 states visited  

524 [(None, 'S'), (0, 'A'), (2, 'D')]  

525 
```

526 In this case, it explores all possible paths down in the right branch
527 of the world, and then has to backtrack up and over to the left
528 branch. `</p>`
529

530 `<p>`Here are some important properties of `{{depth_first_search}}`:
531 ``
532 `` It will run forever if we don't apply pruning rule 1,
533 potentially going back and forth from one state to another,
534 forever.
535 `` It may run forever in an infinite domain (as long as the path
536 it's on has a new successor that hasn't been previously visited, it
537 can go down that path forever; we'll see an example of this in the
538 last section).
539 `` It doesn't necessarily find the shortest path (as we can see
540 from the very first example).

541 **li** It is generally efficient in the amount of space it requires to
542 store the agenda,
543 which will be a constant factor times the depth of the path it is
544 currently considering (we'll explore this in more detail later).

545 **ul**

546

547 **h3**Breadth-First Search**h3**

548 To change to `{{breadth_first_search}}`, we need to choose the oldest,
549 rather than the newest paths from the agenda to expand. All we have
550 to do is change the agenda to be a queue instead of a stack, and
551 everything else stays the same, in the code.

552 **pre**

```
553 def breadthFirstSearch(initialState, goalTest, actions, successor):
554     agenda = Queue()
555     if goalTest(initialState):
556         return [(None, initialState)]
557     agenda.push(SearchNode(None, initialState, None))
558     while not agenda.isEmpty():
559         parent = agenda.pop()
560         newChildStates = []
561         for a in actions:
562             newS = successor(parent.state, a)
563             newN = SearchNode(a, newS, parent)
564             if goalTest(newS):
565                 return newN.path()
566             elif newS in newChildStates:
567                 pass
568             elif parent.inPath(newS):
569                 pass
570             else:
571                 newChildStates.append(newS)
572                 agenda.push(newN)
573     return None
```

574 **pre**

575

```

576 <p>Here is how breadth_first_search works, looking for a path from S to
      S to
577 F in our first city:
578 <pre>
579 >>> breadthFirst('S', lambda x: x == 'F', mapLegalActions,
      mapSuccessors)
580 agenda: Queue([S])
581     expanding: S
582 agenda: Queue([S-0->A, S-1->B])
583     expanding: S-0->A
584 agenda: Queue([S-1->B, S-0->A-1->C, S-0->A-2->D])
585     expanding: S-1->B
586 agenda: Queue([S-0->A-1->C, S-0->A-2->D, S-1->B-1->D, S-1->B-2->E])
587     expanding: S-0->A-1->C
588 7 states visited
589 [(None, 'S'), (0, 'A'), (1, 'C'), (1, 'F')]
590 </pre>
591 We can see it proceeding systematically through paths of length two,
592 then length three, finding the goal among the
593 length-three paths. </p>
594
595 <p>Here are some important properties of breadth-first search:
596 <ul>
597 <li> Always returns a shortest (least number of steps) path to a goal
598 state, if a goal state exists in the set of states reachable from
599 the start state.
600 <li> It may run forever if there is no solution and the domain is
      infinite.
601 <li> It requires more space than depth-first search.
602 </ul></p>
603
604 <p><h3>Dynamic programming</h3></p>
605
606 <p>Let's look at breadth-first search in the first city map example, but
      this
607 time with goal state G:
```



```

608 <pre>
609 >>> breadthFirst('S', lambda x: x == 'G', map1LegalActions,
        map1successors)
610 agenda: Queue([S])
611     expanding: S
612 agenda: Queue([S-0->A, S-1->B])
613     expanding: S-0->A
614 agenda: Queue([S-1->B, S-0->A-1->C, S-0->A-2->D])
615     expanding: S-1->B
616 agenda: Queue([S-0->A-1->C, S-0->A-2->D, S-1->B-1->D, S-1->B-2->E])
617     expanding: S-0->A-1->C
618 agenda: Queue([S-0->A-2->D, S-1->B-1->D, S-1->B-2->E, S-0->A-1->C-1->F
        ])
619     expanding: S-0->A-2->D
620 agenda: Queue([S-1->B-1->D, S-1->B-2->E, S-0->A-1->C-1->F, S-0->A-2->D
        -1->B, S-0->A-2->D-2->F, S-0->A-2->D-3->H])
621     expanding: S-1->B-1->D
622 agenda: Queue([S-1->B-2->E, S-0->A-1->C-1->F, S-0->A-2->D-1->B, S-0->A
        -2->D-2->F, S-0->A-2->D-3->H, S-1->B-1->D-0->A, S-1->B-1->D-2->F, S
        -1->B-1->D-3->H])
623     expanding: S-1->B-2->E
624 agenda: Queue([S-0->A-1->C-1->F, S-0->A-2->D-1->B, S-0->A-2->D-2->F, S
        -0->A-2->D-3->H, S-1->B-1->D-0->A, S-1->B-1->D-2->F, S-1->B-1->D-3->
        H, S-1->B-2->E-1->H])
625     expanding: S-0->A-1->C-1->F
626 16 states visited
627 [(None, 'S'), (0, 'A'), (1, 'C'), (1, 'F'), (2, 'G')]
628 </pre>
629 The first thing that is notable about this trace is that it ends up
630 visiting 16 states in a domain with 9 different states. The issue is
631 that it is exploring multiple paths to the same state. For instance,
632 it has both <tt>S-0->A-2->D</tt> and <tt>S-1->B-1->D</tt> in the agenda.
633 Even worse, it has both <tt>S-0->A</tt> and <tt>S-1->B-1->D-0->A</tt> in
634 there! We really don't need to consider all of these paths. We can
635 make use of the following example of the dynamic programming
636 principle:</p>

```

637
638 *The shortest path from `X` to `Z` that goes through
 `Y` is made*
639 up of the shortest path from `X` to `Y` and the shortest
 path from
640 `Y` to `Z`.

641
642

So, as long as we find the shortest path from the start state to some
643 intermediate state, we don't need to consider any other paths between
644 those two states; there is no way that they can be part of the
645 shortest path between the start and the goal. This insight is the
646 basis of a new pruning principle:

647

Don't consider any path that visits a state that you have already
648 visited via some other path.

649
650

651
652

In `breadth_first_search`, because of the orderliness of the
 expansion
653 of the layers of the search tree, we can guarantee that the first time
654 we visit a state, we do so along the shortest path. So, we'll keep
655 track of the states that we have visited so far, by using a
656 dictionary, called `visited` that has an entry for every state we
657 have visited. (An alternative representation would be just to
658 keep a Python `set` of visited `node`s. Then, if we are
659 considering adding a new node to the tree that goes to a state we have
660 already visited, we just ignore it. This test can take the place of
661 the test we used to have for pruning rule 1; it's clear that if the
662 path we are considering already contains this state, then the state
663 has been visited before. Finally, we have to remember, whenever we
664 add a node to the agenda, to add the corresponding state to the
665 visited list.

666
667

Here is our `breadth_first_search` code, modified to take advantage
 of
668 dynamic programming.

```

669 <pre>
670 def breadthFirstDP(initialState, goalTest, actions, successor):
671     agenda = Queue()
672     if goalTest(initialState):
673         return [(None, initialState)]
674     agenda.push(SearchNode(None, initialState, None))
675     visited = {initialState: True}
676     while not agenda.isEmpty():
677         parent = agenda.pop()
678         for a in actions:
679             newS = successor(parent.state, a)
680             newN = SearchNode(a, newS, parent)
681             if goalTest(newS):
682                 return newN.path()
683             elif visited.has_key(newS):
684                 pass
685             else:
686                 visited[newS] = True:
687                 agenda.push(newN)
688     return None
689 </pre>
690 So, let's see how this performs on the task of going from <TT>S</TT> to
        <TT>G</TT>
691 in the first city map:
692 <pre>
693 >>> breadthFirstDP('S', lambda x: x == 'G', map1LegalActions,
        map1successors)
694 agenda: Queue([S])
695     expanding: S
696 agenda: Queue([S-0->A, S-1->B])
697     expanding: S-0->A
698 agenda: Queue([S-1->B, S-0->A-1->C, S-0->A-2->D])
699     expanding: S-1->B
700 agenda: Queue([S-0->A-1->C, S-0->A-2->D, S-1->B-2->E])
701     expanding: S-0->A-1->C
702 agenda: Queue([S-0->A-2->D, S-1->B-2->E, S-0->A-1->C-1->F])

```

```

703     expanding:  S-0->A-2->D
704 agenda:  Queue([S-1->B-2->E, S-0->A-1->C-1->F, S-0->A-2->D-3->H])
705     expanding:  S-1->B-2->E
706 agenda:  Queue([S-0->A-1->C-1->F, S-0->A-2->D-3->H])
707     expanding:  S-0->A-1->C-1->F
708 8 states visited
709 [(None, 'S'), (0, 'A'), (1, 'C'), (1, 'F'), (2, 'G')]
710 </pre>
711 As you can see, this results in visiting significantly fewer states.
712 Here is the tree generated by this process:
713 <p></p>
714
715 In bigger problems, this effect will be amplified hugely, and will
716 make the difference between whether the algorithm can run in a
717 reasonable amount of time, and not.</p>
718
719 <p>We can make the same improvement to depth-first-search; we just
       need
720 to use a stack instead of a queue in the algorithm above. It still will
721 not guarantee that the shortest path will be found, but will guarantee
722 that we never visit more paths than the actual number of states. The
723 only change to breadth-first search with dynamic programming is that
724 the new states are added to the beginning of the agenda.</p>
725
726 <p><h3>Configurable search code</h3>
727 Because all of our search algorithms (breadth-first and depth-first,
728 with and without dynamic programming) are all so similar, and we don't
729 like to repeat code, we provide (in file search.py) a single,
730 configurable search procedure. It also prints out some information
731 as it goes, if you have the verbose or somewhatVerbose
732 variables set to True, and has a limit on the maximum number of
733 nodes it will expand (to keep from going into an infinite loop).
734 <pre>
735 def search(initialState, goalTest, actions, successor,
736           depthFirst = False, DP = True, maxNodes = 10000):
737     if depthFirst:

```

```

738     agenda = Stack()
739 else:
740     agenda = Queue()
741
742 startNode = SearchNode(None, initialState, None)
743 if goalTest(initialState):
744     return startNode.path()
745 agenda.push(startNode)
746 if DP: visited = {initialState: True}
747 count = 1
748 while not agenda.isEmpty() and maxNodes > count:
749     n = agenda.pop()
750     newStates = []
751     for a in actions:
752         newS = successor(n.state, a)
753         newN = SearchNode(a, newS, n)
754         if goalTest(newS):
755             return newN.path()
756         elif newS in newStates:
757             pass
758         elif ((not DP) and n.inPath(newS)) or \
759             (DP and visited.has_key(newS)):
760             pass
761         else:
762             count += 1
763             if DP: visited[newS] = True
764             newStates.append(newS)
765             agenda.push(newN)
766 return None

```

767 </pre></p>

768

769 <p><h2>Connection to state machines</h2>

770 We can use state machines as a convenient representation of

771 state-space search

772 problems. Given a `{{state_machine}}`, in its initial state, what
sequence

773 of inputs can we feed to it to get it to enter a done state? This is
774 a search problem, analogous to determining the sequence of actions
775 that can be taken to reach a goal state.</p>

776

777 <p>The <tt>getNextValues</tt> method of a state machine can serve as the
 <tt> successor</tt> function in a search (the inputs to the machine
 are the

778 actions). Our standard machines do not have a notion of legal actions;

779 but we will add an attribute called <tt> legalInputs</tt>, which is a
 list of values that are legal inputs to the

780 machine (these are the actions, from

781 the planning perspective) to machines that we want to use with a

782 search.</p>

783

784 <p>The <tt>startState</tt> attribute can serve as the initial state in
 the

785 search and the <tt>done</tt> method of the machine can serve as the goal

786 test function.</p>

787

788 <p>Then, we can plan a sequence of actions to go from the start state to
789 one of the done states using this function, where <tt>smToSearch</tt> is
790 an instance of <tt>sm.SM</tt>.

791 <pre>

```
792 def smSearch(smToSearch, initialState = None, goalTest = None, maxNodes  
    = 10000,
```

```
793            depthFirst = False, DP = True):
```

```
794   if initialState == None:
```

```
795       initialState = smToSearch.startState
```

```
796   if goalTest == None:
```

```
797       goalTest = smToSearch.done
```

```
798   return search(initialState, goalTest, smToSearch.legalInputs,
```

```
799                  # This returns the next state
```

```
800                  lambda s, a: smToSearch.getNextValues(s, a)[0],
```

```
801                  maxNodes = maxNodes,
```

```
802                  depthFirst=depthFirst, DP=DP)
```

```
803 </pre>
```

804 It is mostly clerical: it allows us to specify a different initial
805 state or `{{goal}}` test if we want to, and it extracts the appropriate
806 functions out of the `{{state_machine}}` and passes them into the search
807 procedure. Also, because `<tt>getNextValues</tt>` returns both a state
and
808 an output, we have to wrap it inside a function that just selects out
809 the next state and returns it.</p>

810

811 <p><h2>Numeric search domain</h2>

812

813 <p>Many different kinds of problems can be formulated in terms of
finding

814 the shortest path through a space of states. A famous one, which is
815 very appealing to beginning calculus students, is to take a derivative
816 of a complex equation by finding a sequence of operations that takes
817 you from the starting expression to one that doesn't contain any
818 derivative operations. We'll explore a different simple one here:

819

820 The states are the integers.

821 The initial state is some integer; let's say 1.

822 The legal actions are to apply the following operations: `<tt>{2n,
n+1, n-1, n^2, -n}</tt>`.

823 The goal test is `<tt>lambda x: x == 10</tt>` .

824

825 So, the idea would be to find a short sequence of operations to move
826 from 1 to 10.</p>

827

828 <p>Here it is, formalized as state machine in Python:

829 <pre>

830 class NumberTestSM(sm.SM):

831 startState = 1

832 legalInputs = ['x*2', 'x+1', 'x-1', 'x**2', '-x']

833 def __init__(self, goal):

834 self.goal = goal

835 def nextState(self, state, action):

836 if action == 'x*2':

```

837         return state*2
838     elif action == 'x+1':
839         return state+1
840     elif action == 'x-1':
841         return state-1
842     elif action == 'x**2':
843         return state**2
844     elif action == '-x':
845         return -state
846     def getNextValues(self, state, action):
847         nextState = self.nextState(state, action)
848         return (nextState, nextState)
849     def done(self, state):
850         return state == self.goal

```

851 </pre></p>

852

853 <p>First of all, this is a bad domain for applying `{{depth_first_search}}`.

854 Why? Because it will go off on a gigantic chain of doubling the
855 starting state, and never find the goal. We can run `{{`

`breadth_first_search}}`,

856 though. Without dynamic programming, here is what happens (we

857 have set `<tt>verbose = False</tt>` and `<tt>somewhatVerbose = True</tt>` in
the

858 search file):

859 <pre>

```

860 >>> smSearch(NumberTestSM(10), initialState = 1, depthFirst = False, DP
      = False)

```

```

861     expanding:  1

```

```

862     expanding:  1-x*2->2

```

```

863     expanding:  1-x-1->0

```

```

864     expanding:  1--x->-1

```

```

865     expanding:  1-x*2->2-x*2->4

```

```

866     expanding:  1-x*2->2-x+1->3

```

```

867     expanding:  1-x*2->2--x->-2

```

```

868     expanding:  1-x-1->0-x-1->-1

```



```

869     expanding: 1--x->-1-x*2->-2
870     expanding: 1--x->-1-x+1->0
871     expanding: 1-x*2->2-x*2->4-x*2->8
872     expanding: 1-x*2->2-x*2->4-x+1->5
873 33 states visited
874 [(None, 1), ('x*2', 2), ('x*2', 4), ('x+1', 5), ('x*2', 10)]
875 </pre>
876 We find a nice short path, but visit 33 states. Let's try it with DP:
877 <pre>
878 >>> smSearch(NumberTestSM(10), initialState = 1, depthFirst = False, DP
      = True)
879     expanding: 1
880     expanding: 1-x*2->2
881     expanding: 1-x-1->0
882     expanding: 1--x->-1
883     expanding: 1-x*2->2-x*2->4
884     expanding: 1-x*2->2-x+1->3
885     expanding: 1-x*2->2--x->-2
886     expanding: 1-x*2->2-x*2->4-x*2->8
887     expanding: 1-x*2->2-x*2->4-x+1->5
888 17 states visited
889 [(None, 1), ('x*2', 2), ('x*2', 4), ('x+1', 5), ('x*2', 10)]
890 </pre>
891 We find the same path, but visit noticeably fewer states. If we change
892 the goal to 27, we find that we visit 564 states without DP and 119,
893 with. If the goal is 1027, then we visit 12710 states without DP and
894 1150 with DP, which is getting to be a very big difference.</p>
895
896 <p>To experiment with {{depth_first_search}}, we can make a version of
      the
897 problem where the state space is limited to the integers in some
898 range. We do this by making a subclass of the NumberTestSM,
899 which remembers the maximum legal value, and uses it to restrict the
900 set of legal inputs for a state (any input that would cause the
901 successor state to go out of bounds just results in staying at the
902 same state, and it will be pruned.)

```

```

903 <pre>
904 class NumberTestFiniteSM(NumberTestSM):
905     def __init__(self, goal, maxVal):
906         self.goal = goal
907         self.maxVal = maxVal
908     def getNextValues(self, state, action):
909         nextState = self.nextState(state, action)
910         if abs(nextState) < self.maxVal:
911             return (nextState, nextState)
912         else:
913             return (state, state)
914 </pre>
915 Here's what happens if we give it a range of -20 to +20 to work in:
916 <pre>
917 >>> smSearch(NumberTestFiniteSM(10, 20), initialState = 1, depthFirst =
          True,
918             DP = False)
919 expanding:  1
920 expanding:  1--x->-1
921 expanding:  1--x->-1-x+1->0
922 expanding:  1--x->-1-x*2->-2
923 expanding:  1--x->-1-x*2->-2--x->2
924 expanding:  1--x->-1-x*2->-2--x->2-x+1->3
925 expanding:  1--x->-1-x*2->-2--x->2-x+1->3--x->3
926 expanding:  1--x->-1-x*2->-2--x->2-x+1->3--x->3-x**2->9
927 20 states visited
928 [(None, 1), ('-x', -1), ('x*2', -2), ('-x', 2), ('x+1', 3), ('-x', -3),
          ('x**2', 9), ('x+1', 10)]
929 </pre>
930 We generate a much longer path!</p>
931
932 <p>We can see from trying lots of different searches in this space that
933 (a) the DP makes the search much more efficient and (b) that the
934 difficulty of these search problems varies incredibly widely.</p>
935
936 <p><b>Computational complexity</b></p>

```

937
938 <p>To finish up this segment, let's consider the computational
complexity
939 of these algorithms. As we've already seen, there can be a huge
940 variation in the difficulty of a problem that depends on the exact
941 structure of the graph, and is very hard to quantify in advance.
942 It can sometimes be possible to analyze the average case running time
943 of an algorithm, if you know some kind of distribution over the
944 problems you're likely to encounter. We'll just stick with the
945 traditional *worst-case analysis*, which tries to characterize the
946 approximate running time of the worst possible input for the algorithm.
</p>
947
948 <p>First, we need to establish a bit of notation. Let
949
950 <tt>b</tt> be the *branching factor* of the graph; that is,
the
951 number of successors a node can have. If we want to be truly
952 worst-case in our analysis, this needs to be the maximum branching
953 factor of the graph.
954 <tt>d</tt> be the *maximum depth* of the graph; that is, the
955 length of the longest path in the graph. In an infinite space, this
956 could be infinite.
957 <tt>l</tt> be the *solution depth* of the problem; that is,
the
958 length of the shortest path from the start state to the shallowest
959 goal state.
960 <tt>n</tt> be the *state space size* of the graph; that is
the
961 total number of states in the domain.
962 </p>
963
964 <p>{{depth-first-search}}, in the worst case, will search the entire
search
965 tree. It has <tt>d</tt> levels, each of which has <tt>b</tt> times as
many paths as

966 the previous one. So, there are b^d paths on the d^{th} level.

967 The algorithm might have to visit all of the paths at all of the

968 levels, which is about b^{d+1} states.

969 But the amount of storage it needs for the agenda is only $b \cdot d$.

970

971 `breadth_first_search`, on the other hand, only needs to search as deep

972 as the depth of the best solution. So, it might have to visit as many

973 as b^{l+1} nodes. The amount of storage required for the agenda can be

974 as bad as b^l , too.

975

976 So, to be clear, consider the numeric search problem. The branching

977 factor $b = 5$, in the worst case. So, if we have to find a sequence

978 of 10 steps, breadth-first search could potentially require visiting

979 as many as $5^{11} = 48828125$ nodes!

980

981 This is all pretty grim. What happens when we consider the DP version

982 of `breadth_first_search`? We can promise that every state in the state

983 space is visited at most once. So, it will visit at most n states.

984 Sometimes n is much smaller than b^l

985 (for instance, in a road network). In other cases, it can be much larger (for instance,

986 when you are solving an easy (short solution path) problem embedded in

987 a very large space). Even so, the DP version of the search will visit fewer

988 states, except in the very rare case in which there are never multiple

989 paths to the same state (the graph is actually a tree). For example,

990 in the numeric search problem, the shortest path from 1 to 91 is 9

991 steps long, but using DP it only requires visiting 1973 states, rather

992 than $5^{10} = 9765625$.

993 `{{basic_search_video}}`

994

Uniform cost search

995

996

In many cases, the arcs in our graph will actually have different
997 costs. In a road network, we would really like to find the shortest
998 path in miles (or in time to traverse), and different road segments
999 have different lengths and times. To handle a problem like this, we
1000 need to extend our representation of search problems, and add a new
1001 algorithm to our repertoire.

1002

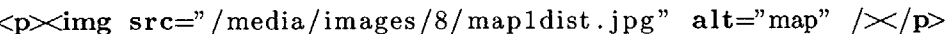
1003

We will extend our notion of a successor function, so that it takes a
1004 state and an action, as before, but now it returns a pair `(newS,
cost)`, which represents the resulting state, as well as
1005 the cost that is incurred in traversing that arc. To guarantee that
1006 all of our algorithms are well behaved, we will require that all costs
1007 be positive (not zero or negative).

1008

1009

Here is our original city map, now with distances associated with the
1010 roads between the cities.

1011 

1012

1013 We can describe it in a dictionary, this time associating a cost with
1014 each resulting state, as follows:

1015

```
1016 mapldist = {'S' : [('A', 2), ('B', 1)],  
1017             'A' : [('S', 2), ('C', 3), ('D', 2)],  
1018             'B' : [('S', 1), ('D', 2), ('E', 3)],  
1019             'C' : [('A', 3), ('F', 1)],  
1020             'D' : [('A', 2), ('B', 2), ('F', 4), ('H', 6)],  
1021             'E' : [('B', 3), ('H', 2)],  
1022             'F' : [('C', 1), ('D', 4), ('G', 1)],  
1023             'H' : [('D', 6), ('E', 2), ('G', 4)],  
1024             'G' : [('F', 1), ('H', 4)]}
```

1025

1026

1027 <p>When we studied `breadth_first_search`, we argued that it found the
1028 shortest path, in the sense of having the fewest nodes, by seeing that
1029 it investigate all of the length 1 paths, then all of the length 2
1030 paths, etc. This orderly enumeration of the paths guaranteed that
1031 when we first encountered a goal state, it would be via a shortest
1032 path. The idea behind `uniform cost search` is basically the
1033 same: we are going to investigate paths through the graph, in the
1034 order of the sum of the costs on their arcs. If we do this, we
1035 guarantee that the first time we extract a `node` with a given state
1036 from the `agenda`, it will be via a shortest `path`, and so the
first time
1037 we extract a node with a `goal` state from the agenda, it will be an
1038 optimal solution to our problem.</p>

1039

1040 <h4>Priority Queue</h4>

1041 Just as we used a stack to manage the agenda for depth-first search
1042 and a queue to manage the agenda for bread-first search, we will need
1043 to introduce a new data structure, called a `priority queue` to
1044 manage the agenda for `uniform_cost_search`. A priority queue is a
data

1045 structure with the same basic operations as stacks and queues, with
1046 two differences:

1047

1048 Items are pushed into a priority queue with a numeric score, called
a `cost`.

1049 When it is time to pop an item, the item in the priority queue with
the least `cost` is returned and removed from the priority
queue.

1050

1051 There are many interesting ways to implement priority queues so that
1052 they are very computationally efficient. Here, we show a very simple
1053 implementation that simply walks down the entire contents of the
1054 priority queue to find the least-`cost` item for a pop operation. Its
1055 `data` attribute consists of a list of `(cost, item)`
pairs.

1056 It calls the `argmaxIndex` procedure from our utility package,

1057 which takes a list of items and a scoring function, and returns a pair
1058 consisting of the index of the list with the highest scoring item, and
1059 the score of that item. Note that, because `argmaxIndex` finds
1060 the item with the **highest score**, and we want to extract the
item

1061 with the **least cost**, our scoring function is the **negative**
em>
1062 of the cost.

1063 **<pre>**

```
1064 class PQ:
1065     def __init__(self):
1066         self.data = []
1067     def push(self, item, cost):
1068         self.data.append((cost, item))
1069     def pop(self):
1070         (index, cost) = util.argmaxIndex(self.data, lambda (c, x): -c)
1071         return self.data.pop(index)[1] # just return the data item
1072     def isEmpty(self):
1073         return self.data is []
```

1074 **</pre></p>**

1075

1076 **<h4>UC Search</h4>**

1077 Now, we're ready to study the `uniform_cost_search` algorithm itself.
1078 We will start with a simple version that doesn't do any pruning or
1079 dynamic programming, and then add those features back in later.
1080 First, we have to extend our definition of a `SearchNode`, to
1081 incorporate costs. So, when we create a new search `node`, we pass in
1082 an additional parameter `actionCost`, which represents the `cost`
}}

1083 just for the action that moves from the parent node to the state.

1084 Then, we create an attribute `self.cost`, which encodes the cost
1085 of this entire path, from the starting state to the last state in the
1086 path. We compute it by adding the path cost of the parent to the cost
1087 of this last action, as shown by the red text below.

1088 **<pre>**

```
1089 class SearchNode:
```

```

1090     def __init__(self, action, state, parent, /BTEX\redtext{actionCost}/
        ETEX ):
1091         self.state = state
1092         self.action = action
1093         self.parent = parent
1094         if self.parent:
1095             self.cost = self.parent.cost + actionCost
1096         else:
1097             self.cost = actionCost
1098 </pre></p>
1099
1100 <p>Now, here is the search algorithm. It looks a lot like our standard
1101 search algorithm, but there are two important differences:
1102 <ul>
1103 <li> The agenda is a priority queue.
1104 <li> Instead of testing for a goal state when we put an element <em>
1105 into</em> the {{agenda}}, as we did in {{breadth_first_search}}, we test
        for a
1106 goal state when we take an element <em>out of</em> the agenda. This is
1107 crucial, to ensure that we actually find the shortest path to a goal
1108 state.
1109 </ul>
1110
1111 <p><pre>
1112 def ucSearch(initialState, goalTest, actions, successor):
1113     startNode = SearchNode(None, initialState, None, 0)
1114     if goalTest(initialState):
1115         return startNode.path()
1116     agenda = PQ()
1117     agenda.push(startNode, 0)
1118     while not agenda.isEmpty():
1119         n = agenda.pop()
1120         if goalTest(n.state):
1121             return n.path()
1122         for a in actions:
1123             (newS, cost) = successor(n.state, a)

```



```

1124         if not n.inPath(newS):
1125             newN = SearchNode(a, newS, n, cost)
1126             agenda.push(newN, newN.cost)
1127     return None
1128 </pre></p>
1129
1130 <h4>Example</h4>
1131 Consider the following simple graph:
1132
1133 <p></p>
1134 Let's simulate the uniform-cost search algorithm, and see what
1135 happens when we try to start from <tt>S</tt> and go to <tt>D</tt>:
1136 <ul>
1137 <li> The agenda is initialized to contain the starting node. The
1138 agenda is shown as a list of cost, node pairs.
1139 <pre>
1140 agenda: PQ([(0, S)])
1141 </pre>
1142 </li>
1143 <li> The least-cost node, <tt>S</tt>, is extracted and expanded, adding
1144 two new nodes to the agenda. The notation <tt>S-0->A</tt> means that
1145 the path starts in state <tt>S</tt>, takes action <tt>0</tt>, and goes
        to
1146 state <tt>A</tt>.
1147 <pre>
1148     0 :   expanding: S
1149 agenda: PQ([(2, S-0->A), (1, S-1->B)])
1150 </pre>
1151 </li>
1152 <li> The least-cost node, <tt>S-1->B</tt>, is extracted, and expanded,
1153 adding one new node to the agenda. Note, that, at this point, we
1154 have discovered a path to the goal: <tt>S-1->B-1->D</tt> is a path to
1155 the goal, with cost 11. But we cannot be sure that it is the
1156 shortest path to the goal, so we simply put it into the agenda, and
1157 wait to see if it gets extracted before any other path to a goal
1158 state.

```

```

1159 <pre>
1160     1 :   expanding: S-1->B
1161 agenda: PQ([(2, S-0->A), (11, S-1->B-1->D)])
1162 </pre>
1163 </li>
1164 <li> The least-cost node, <tt>S-0->A</tt> is extracted, and expanded,
1165 adding one new node to the agenda. At this point, we have two
1166 different paths to the goal in the agenda.
1167 <pre>
1168     2 :   expanding: S-0->A
1169 agenda: PQ([(11, S-1->B-1->D), (4, S-0->A-1->D)])
1170 </pre>
1171 </li>
1172 <li> Finally, the least-cost node, <tt>S-0->A-1->D</tt> is extracted.
1173 It is a path to the goal, so it is returned as the solution.
1174 <pre>
1175 5 states visited; Solution cost: 4
1176 [(None, 'S'), (0, 'A'), (1, 'D')]
1177 </pre>
1178 </li>
1179 </ul></p>
1180
1181 <h4>Dynamic programming</h4>
1182 Now, we just need to add dynamic programming back in, but we have to
1183 do it slightly differently. We promise that, once we have <em>expanded<
1184 /em> a node, that is, taken it out of the agenda, then we have
1185 found the shortest path to that state, and we need not consider any
1186 further paths that go through that state. So, instead of remembering
1187 which nodes we have visited (put onto the {{agenda}}) we will remember
1188 nodes we have <em>expanded</em> (gotten out of the agenda), and never
1189 {{visit}} or {{expand}} a node that has already been expanded. In the
1190 code
1191 below, the first test ensures that we don't expand a node that goes to
1192 a state that we have already found the shortest path to, and the
1193 second test ensures that we don't put any additional paths to such a
1194 state into the {{agenda}}.

```

```

1193 <pre>
1194 def ucSearch(initialState, goalTest, actions, successor):
1195     startNode = SearchNode(None, initialState, None, 0)
1196     if goalTest(initialState):
1197         return startNode.path()
1198     agenda = PQ()
1199     agenda.push(startNode, 0)
1200     expanded = { }
1201     while not agenda.isEmpty():
1202         n = agenda.pop()
1203         if not expanded.has\_key(n.state):
1204             expanded[n.state] = True
1205             if goalTest(n.state):
1206                 return n.path()
1207             for a in actions:
1208                 (newS, cost) = successor(n.state, a)
1209                 if not expanded.has\_key(newS):
1210                     newN = SearchNode(a, newS, n, cost)
1211                     agenda.push(newN, newN.cost)
1212     return None
1213 </pre></p>
1214
1215 <p>Here is the result of running this version of {{uniform_cost_search}}
    on
1216 our bigger city graph with distances:
1217 <pre>
1218 mapDistTest(map1dist, 'S', 'G')
1219 agenda: PQ([(0, S)])
1220     0 :   expanding: S
1221 agenda: PQ([(2, S-0->A), (1, S-1->B)])
1222     1 :   expanding: S-1->B
1223 agenda: PQ([(2, S-0->A), (3, S-1->B-1->D), (4, S-1->B-2->E)])
1224     2 :   expanding: S-0->A
1225 agenda: PQ([(3, S-1->B-1->D), (4, S-1->B-2->E), (5, S-0->A-1->C), (4, S
    -0->A-2->D)])
1226     3 :   expanding: S-1->B-1->D

```

```

1227 agenda: PQ([(4, S-1->B-2->E), (5, S-0->A-1->C), (4, S-0->A-2->D), (7, S
      -1->B-1->D-2->F), (9, S-1->B-1->D-3->H)])
1228   4 :   expanding: S-1->B-2->E
1229 agenda: PQ([(5, S-0->A-1->C), (4, S-0->A-2->D), (7, S-1->B-1->D-2->F),
      (9, S-1->B-1->D-3->H), (6, S-1->B-2->E-1->H)])
1230 agenda: PQ([(5, S-0->A-1->C), (7, S-1->B-1->D-2->F), (9, S-1->B-1->D
      -3->H), (6, S-1->B-2->E-1->H)])
1231   5 :   expanding: S-0->A-1->C
1232 agenda: PQ([(7, S-1->B-1->D-2->F), (9, S-1->B-1->D-3->H), (6, S-1->B
      -2->E-1->H), (6, S-0->A-1->C-1->F)])
1233   6 :   expanding: S-1->B-2->E-1->H
1234 agenda: PQ([(7, S-1->B-1->D-2->F), (9, S-1->B-1->D-3->H), (6, S-0->A
      -1->C-1->F), (10, S-1->B-2->E-1->H-2->G)])
1235   6 :   expanding: S-0->A-1->C-1->F
1236 agenda: PQ([(7, S-1->B-1->D-2->F), (9, S-1->B-1->D-3->H), (10, S-1->B
      -2->E-1->H-2->G), (7, S-0->A-1->C-1->F-2->G)])
1237 agenda: PQ([(9, S-1->B-1->D-3->H), (10, S-1->B-2->E-1->H-2->G), (7, S
      -0->A-1->C-1->F-2->G)])
1238 13 states visited; Solution cost: 7
1239 [(None, 'S'), (0, 'A'), (1, 'C'), (1, 'F'), (2, 'G')]
1240 </pre></p>
1241
1242 <p><h3>Connection to state machines</h3>
1243 When we use a state_machine to specify a domain for a cost-based
1244 search, we only need to make a small change: the
1245 getNextValues method of a state machine can still serve as the
      successor function in a search (the inputs to the
      machine are the
1246 actions). We usually think of getNextValues as returning the
1247 next state and the output: now, we will modify that interpretation
1248 slightly, and think of it as returning the next state and the
1249 incremental cost of taking the action that transitions to that next
1250 state. This has the same form as the ucSearch.search
1251 procedure expects a successor_function to have, so we don't need to
1252 change anything about the smSearch procedure we have already
      defined.</p>

```

1253

1254 <p><h2>Search with heuristics</h2></p>

1255

1256 <p>Ultimately, we'd like to be able to solve huge state-space search
1257 problems, such as those solved by a GPS that can plan long routes
1258 through a complex road network. We'll have to add something to
1259 {{uniform_cost_search}} to solve such problems efficiently.

1260 Let's consider the city below, where the actual distances between the
1261 intersections are shown on the arcs:

1262 <p></p>

1263

1264

1265 <p>If we use {{uniform_cost_search}} to find a path from <tt>G</tt> to <
tt>X</tt>,

1266 we expand states in the following order (the number at the beginning
1267 of each line is the length of the path from <tt>G</tt> to the state at
1268 the end of the path:

1269 <pre>

1270 >>> bigTest('G', 'X')

1271 0 : expanding: G

1272 14.2 : expanding: G-2->H

1273 20.7 : expanding: G-1->F

1274 25.5 : expanding: G-0->I

1275 30.1 : expanding: G-2->H-2->T

1276 32.3 : expanding: G-2->H-0->D

1277 36.7 : expanding: G-0->I-3->J

1278 40.5 : expanding: G-0->I-0->C

1279 44.3 : expanding: G-2->H-2->T-1->R

1280 45.9 : expanding: G-2->H-1->E

1281 50.4 : expanding: G-2->H-0->D-0->S

1282 50.5 : expanding: G-0->I-2->L

1283 52.6 : expanding: G-0->I-3->J-2->M

1284 54.7 : expanding: G-2->H-0->D-2->B

1285 54.7 : expanding: G-0->I-0->C-0->A

1286 54.8 : expanding: G-0->I-3->J-1->K

1287 58.5 : expanding: G-2->H-2->T-1->R-1->V

```

1288 66.0 : expanding: G-0->I-3->J-1->K-1->N
1289 68.5 : expanding: G-0->I-3->J-2->M-1->P
1290 68.6 : expanding: G-0->I-2->L-1->O
1291 69.7 : expanding: G-2->H-2->T-1->R-1->V-2->Y
1292 82.7 : expanding: G-0->I-3->J-2->M-1->P-1->Q
1293 84.7 : expanding: G-2->H-2->T-1->R-1->V-2->Y-2->Z
1294 86.7 : expanding: G-0->I-2->L-1->O-1->W
1295 95.6 : expanding: G-0->I-2->L-1->O-2->U
1296 95.9 : expanding: G-2->H-2->T-1->R-1->V-2->Y-2->Z-2->AA
1297 39 nodes visited; 27 states expanded; solution cost: 105.9
1298 [(None, 'G'), (2, 'H'), (2, 'T'), (1, 'R'), (1, 'V'), (2, 'Y'), (2, 'Z')
      , (2, 'AA'), (1, 'X')]
1299 </pre>
1300 This search process works its way out, radially, from <tt>G</tt>,
1301 expanding nodes in contours of increasing path length. That means
1302 that, by the time the search expands node <tt>X</tt>, it has expanded
1303 every single node.
1304 This seems kind of silly: if you were looking for a good route from
1305 <tt>G</tt> to <tt>X</tt>, it's unlikely that states like <tt>S</tt> and
      <tt>B</tt>
1306 would ever come into consideration.</p>
1307
1308 <p><h4>Heuristics</h4></p>
1309
1310 <p>What is it about state <tt>B</tt> that makes it seem so irrelevant?
1311 Clearly, it's far away from where we want to go. We can incorporate
1312 this idea into our {{search}} algorithm using something called a <em>
1313 heuristic function</em>. A heuristic function takes a state as an
1314 argument and returns a numeric estimate of the total cost that it will
1315 take to reach the goal from there. We can modify our search algorithm
1316 to be biased toward states that are closer to the goal, in the sense
1317 that the heuristic function has a smaller value on them.</p>
1318
1319 <p>In a path-planning domain, such as our example, a reasonable
      heuristic
1320 is the actual Euclidean distance between the current state and the

```

1321 goal state; this makes sense because the states in this domain are
1322 actual locations on a map.</p>
1323
1324 <p><h4>A*</h4></p>
1325
1326 <p>If we modify the {{uniform_cost_search}} algorithm to take advantage
of a
1327 heuristic function, we get an algorithm called <TT>A^*</TT> (pronounced
'a
1328 star'). It is given below, with the differences highlighted in red.
1329 The only difference is that, when we insert a {{node}} into the
1330 priority queue, we do so with a {{cost}} that is <tt>newN.cost +
heuristic(newS)</tt>. That is, it is the sum of the actual cost of
the
1331 path from the start state to the current state, and the estimated cost
1332 to go from the current state to the goal.
1333 <pre>
1334 def ucSearch(initialState, goalTest, actions, successor, /BTEX\redtext{
heuristic}/ETEX):
1335 startNode = SearchNode(None, initialState, None, 0)
1336 if goalTest(initialState):
1337 return startNode.path()
1338 agenda = PQ()
1339 agenda.push(startNode, 0)
1340 expanded = { }
1341 while not agenda.isEmpty():
1342 n = agenda.pop()
1343 if not expanded.has_key(n.state):
1344 expanded[n.state] = True
1345 if goalTest(n.state):
1346 return n.path()
1347 for a in actions:
1348 (newS, cost) = successor(n.state, a)
1349 if not expanded.has_key(newS):
1350 newN = SearchNode(a, newS, n, cost)
1351 agenda.push(newN, newN.cost + heuristic(newS))

```

1352     return None
1353 </pre></p>
1354
1355 <p><h4>Example</h4>
1356 Now, we can try to search in the big map for a {{path}} from <tt>G</tt>
      to
1357 <tt>X</tt>, using, as our heuristic function, the distance between
1358 the state of interest and <tt>X</tt>. Here is a trace of what happens
1359 (with the numbers rounded to increase readability):
1360 <ul>
1361 <li> We get the start node out of the {{agenda}}, and add its children.
1362 Note that the costs are the actual path cost <em>plus</em> the
1363 {{heuristic}} estimate.
1364 <pre>
1365     0 :   expanding:  G
1366 agenda:  PQ([(107, G-0->I), (101, G-1->F), (79, G-2->H)])
1367 </pre>
1368 </li>
1369 <li> The least cost path is <tt>G-2->H</tt>, so we extract it, and add
1370 its successors.
1371 <pre>
1372     14.2 :   expanding:  G-2->H
1373 agenda:  PQ([(107, G-0->I), (101, G-1->F), (109, G-2->H-0->D), (116, G
      -2->H-1->E), (79, G-2->H-2->T)])
1374 </pre>
1375 </li>
1376 <li> Now, we can see the {{heuristic}} function really having an effect.
      The
1377 path <tt>G-2->H-2->T</tt> has length 30.1, and the path <tt>G-1-F</tt>
      has
1378 length 20.7. But when we add in the heuristic cost estimates, the path
1379 to <tt>T</tt> has a lower cost, because it seems to be going in the
      right
1380 direction. Thus, we select <tt>G-2->H-2->T</tt> to expand next:
1381 <pre>
1382     30.1 :   expanding:  G-2->H-2->T

```


1383 agenda: PQ([(107, G-0->I), (101, G-1->F), (109, G-2->H-0->D), (116, G
-2->H-1->E), (100, G-2->H-2->T-1->R)])

1384 </pre>

1385

1386 Now the path <tt>G-2->H-2->T-1->R</tt> looks best, so we expand it.

1387 <pre>

1388 44.3 : expanding: G-2->H-2->T-1->R

1389 agenda: PQ([(107, G-0->I), (101, G-1->F), (109, G-2->H-0->D), (116, G
-2->H-1->E), (103.5, G-2->H-2->T-1->R-1->V)])

1390 </pre>

1391

1392 Here, something interesting happens. The node with the least
1393 estimated cost is <tt>G-1->F</tt>. It's going in the wrong direction,
1394 but if we were to be able to fly straight from <tt>F</tt> to <tt>X</tt>,
1395 then that would be a good way to go. So, we expand it:

1396 <pre>

1397 20.7 : expanding: G-1->F

1398 agenda: PQ([(107, G-0->I), (109, G-2->H-0->D), (116, G-2->H-1->E),
(103.5, G-2->H-2->T-1->R-1->V), (123, G-1->F-0->D), (133, G-1->F-1->
C)])

1399 </pre>

1400

1401 Continuing now, basically straight to the goal, we have:

1402 <pre>

1403 58.5 : expanding: G-2->H-2->T-1->R-1->V

1404 agenda: PQ([(107, G-0->I), (109, G-2->H-0->D), (116, G-2->H-1->E),
(123, G-1->F-0->D), (133, G-1->F-1->C), (154, G-2->H-2->T-1->R-1->V
-1->E), (105, G-2->H-2->T-1->R-1->V-2->Y)])

1405 69.7 : expanding: G-2->H-2->T-1->R-1->V-2->Y

1406 agenda: PQ([(107, G-0->I), (109, G-2->H-0->D), (116, G-2->H-1->E),
(123, G-1->F-0->D), (133, G-1->F-1->C), (154, G-2->H-2->T-1->R-1->V
-1->E), (175, G-2->H-2->T-1->R-1->V-2->Y-0->E), (105, G-2->H-2->T
-1->R-1->V-2->Y-2->Z)])

1407 84.7 : expanding: G-2->H-2->T-1->R-1->V-2->Y-2->Z

1408 agenda: PQ([(107, G-0->I), (109, G-2->H-0->D), (116, G-2->H-1->E),
(123, G-1->F-0->D), (133, G-1->F-1->C), (154, G-2->H-2->T-1->R-1->V

-1->E), (175, G-2->H-2->T-1->R-1->V-2->Y-0->E), (151, G-2->H-2->T-1->R-1->V-2->Y-2->Z-1->W), (106, G-2->H-2->T-1->R-1->V-2->Y-2->Z-2->AA))

1409 95.9 : expanding: G-2->H-2->T-1->R-1->V-2->Y-2->Z-2->AA

1410 agenda: PQ([(107, G-0->I), (109, G-2->H-0->D), (116, G-2->H-1->E), (123, G-1->F-0->D), (133, G-1->F-1->C), (154, G-2->H-2->T-1->R-1->V-1->E), (175, G-2->H-2->T-1->R-1->V-2->Y-0->E), (151, G-2->H-2->T-1->R-1->V-2->Y-2->Z-1->W), (106, G-2->H-2->T-1->R-1->V-2->Y-2->Z-2->AA-1->X)])

1411 18 nodes visited; 10 states expanded; solution cost: 105.9

1412 [(None, 'G'), (2, 'H'), (2, 'T'), (1, 'R'), (1, 'V'), (2, 'Y'), (2, 'Z'), (2, 'AA'), (1, 'X')]

1413 </pre>

1414

1415 </p>

1416

1417 <p>Using `A_star` has roughly halved the number of `nodes` `visited` and `expanded`.

1418 In some problems it can result in an enormous savings, but, as we'll

1419 see in the next section, it depends on the heuristic we use.</p>

1420

1421 <h4>Good and bad heuristics</h4></p>

1422

1423 <p>In order to think about what makes a heuristic good or bad, let's

1424 imagine what the perfect heuristic would be. If we were to magically

1425 know the distance, via the shortest path in the graph, from each node

1426 to the goal, then we could use that as a heuristic. It would lead us

1427 directly from start to `goal`, without expanding any extra nodes. But

1428 of course, that's silly, because it would be at least as hard to

1429 compute the heuristic function as it would be to solve the original

1430 search problem.</p>

1431

1432 <p>So, we would like our heuristic function to give an estimate that is

1433 as close as possible to the true shortest-path-length from the state

1434 to the goal, but also to be relatively efficient to compute. </p>

1435
1436 <p>An important additional question is: if we use a heuristic function,
1437 are we still guaranteed to find the shortest path through our state
1438 space? The answer is: yes, if the heuristic function is admissible
/em>. A heuristic function is admissible if it is guaranteed
1439 to be an underestimate of the actual cost of the optimal path
to
1440 the goal. To see why this is important, consider a
1441 state <tt>s</tt> from which the goal can actually be reached in 10
1442 steps, but for which the heuristic function gives a value of 100.
1443 Any {{path}} to that state will be put into the {{agenda}} with a total
{{cost}}
1444 of 90 more than the true cost. That means that if a path is found
1445 that is as much as 89 units more expensive than the optimal path, it
1446 will be accepted and returned as a result of the search.</p>
1447
1448 <p>It is important to see that if our heuristic function always returns
1449 value 0, it is admissible. And, in fact, with that {{heuristic}}, the
1450 A^{*} algorithm reduces to uniform cost search.</p>
1451
1452 <p>In the example of navigating through a city, we used the Euclidean
1453 distance between cities, which, if distance is our cost, is clearly
1454 admissible; there's no shorter path between any two points. </p>
1455
1456 <p>
1457 Would the so-called 'Manhattan distance', which is the sum of the
1458 absolute differences of the <tt>x</tt> and <tt>y</tt> coordinates be an
admissible
1459 heuristic in the city navigation problem, in general? Would it be
1460 admissible in Manhattan?
1461 </p>
1462
1463 <p>
1464 If we were trying to minimize travel time on a road network (and so
1465 the estimated time to travel each road segment was the cost), what
1466 would be an appropriate heuristic function?

```
1467 </p>
1468
1469 <p>
1470 {{ Costs_Heuristics_and_A_star_video }}
1471 {% endblock %}
```

Bibliography

- [1] B. Bloom. The 2 sigma problem: The search for methods of group instruction as effective as one-to-one tutoring. *Educational Researcher*, 13(6):4–16, 1984.
- [2] Vannevar Bush. As We May Think. *Atlantic Monthly*, 176(1):641–649, March 1945.
- [3] edX. edx. <http://www.edxonline.org/>, June 2012. [Online; accessed 22-June-2012].
- [4] MIT-Microsoft iCampus alliance. icampus. <http://icampus.mit.edu/>, January 2007. [Online; accessed 22-June-2012].
- [5] inkling. inkling. <https://www.inkling.com/>, June 2012. [Online; accessed 22-June-2012].
- [6] Khanacademy. Khanacademy. <http://www.khanacademy.org/>, June 2012. [Online; accessed 22-June-2012].
- [7] OpenCourseWare. Opencourseware. <http://ocw.mit.edu>, June 2012. [Online; accessed 22-June-2012].
- [8] Sidney Pressey. A simple apparatus which gives tests and scores - and teaches. *School and Society*, 23:373–376, 1926.
- [9] Technology Review. Feedback. <http://www.technologyreview.com/letter/426443/feedback/>, January 2012. [Online; accessed 22-June-2012].
- [10] The RSA. Changing education paradigms. <http://comment.rsablogs.org.uk/2010/10/14/rsa-animate-changing-education-paradigms/>, June 2012. [Online; accessed 22-June-2012].