Massachusetts Institute of Technology

16.410-13 Principles of Autonomy and Decision Making

Problem Set #2

Objective

You will implement in Java the following search algorithms:

1. Depth-first
2. Breadth-first.

You will then experiment with these algorithms using the Battlecode platform. You will install and familiarize yourself with Battlecode, which will be used in some other problem sets as well.

You will demonstrate good programming style, including the appropriate use of exceptions, comments, interfaces, and classes. This will allows the teaching staff to assess your programming ability, coming into the course.

In addition, the second objective of this problem set is to develop your skill at performing an asymptotic analysis of uninformed search.

Submission Instructions

Please submit your source code [src] electronically. However, do NOT submit a collection of files. You must submit a single ZIP file containing all of the source files. Make sure to comment your code sufficiently so that the grader can easily understand your code. No partial credit can be given if the code cannot be understood.

Submit a single PDF [pdf] file containing all additional material such as execution traces and analysis. You must also submit a hard copy of the PDF file.

At the end of each problem, a list specifies all items that must be included in your submission. Each item that has [pdf] written next to it must be part of the PDF document, and not a separate file. As you complete your assignment, please include in your PDF document, at the end of each answer, how long you spent on it.

Background

Implement the following problems according to the lecture notes (2 and 3). Use AIMA Chapter 3 for background reading but implement your algorithms according to the lecture

notes (not AIMA).

Asymptotic analysis is a powerful method for analyzing computational systems. We recommend that those of you who are interested in learning more about asymptotic analysis read any of Chapters 1 - 4 of "Introduction to Algorithms," by Thomas Cormen, Charles Leiserson and Ronald Rivest, MIT Press.

Problem 1 (40 points)

Implement a general search capability by supplying Java methods corresponding to the following search algorithms:

- breadth-first search
- depth-first search.

You must use the provided source codes, which consist of partial implementations. You must not change the specification. Look for "`//*** You fill in here! ***//`" and implement the missing sections as indicated. Read the comments in the source code carefully. The comments will provide additional helpful information.

Since you will be testing your code with in BattleCode, we will not require Junit tests on this assignment. ***However,*** the BattleCode stack is limited to 64 calls and therefore we suggest that you do not use recursion in implementing your search algorithms.

1. Implement a general uninformed search by completing the implementation of the following three methods in the UninformedSearch class:

```
public void initializeSearch(State start)
```

This method is used to initialize the search, by resetting or clearing the member variables of the UninformedSearch class. The search queue should be initialized to a single search node that only includes the start state with no parent. The visited list should be initialized with the start state as the only element. More specifically, this method should:

- reset the number of search nodes,
- clear the search queue,
- reset the maximum size of the queue,
- clear the set of visited states,
- insert into the search queue, a node that includes only the start state,
- add the start state to the set of visited state, and
- update the maximum queue size reached.

```
protected Set<Node> getExtendedPaths(Node node)
```

This method is used to find the possible nodes that can be searched. Find all the states that are not in Visited and are reachable from the current search node in one step. Then create all of the new search nodes which link to them.

- Remember to update the visited list here, regardless of the value of `m_useVisitedList`.
- If the visited list is used, make sure to extend the path to the children that have **NOT** been visited; otherwise, the path should be extended to all children, (which may lead to cycles.

```
public LinkedList<Action> incrementSearch(int S, int G)
```

This method performs a general uninformed graph search. This method should implement the pseudo code provided in the lecture. Make note of the comments that include "`[Modification]`". These are the only modifications that are made to the pseudo code provided in the lecture.

- Make sure to only use the `addToQueue()` method when adding a set of paths to the search queue.
- Make sure to extend a path only if it is allowed to be extended by the `isExtensionAllowed()` method.
- This method returns a sequence of actions (a "plan") that will take us from the start state to the goal state.

2. Implement breadth-first search, which extends the implementation of the general uninformed search class, by completing the implementation of the following method:

```
protected void addToQueue(Set<Node> reachableNodes)
```

This method is used to add the set of new nodes to the search queue. Remember that breadth-first search adds the new nodes to the tail of the search queue, i.e. the search queue is of type First-In First-Out (FIFO). Make sure to maintain the ordering of the nodes when added to the search queue. For example, if Q = (A,B,C,D) and the reachableNodes = (E,F,G), then the result of adding the extended paths to Q should be (A,B,C,D,E,F,G).

3. Implement depth-first search, which extends the implementation of the general uninformed search class by completing the implementation of the following method:

```
protected void addToQueue(Set<Node> reachableNodes)
```

Remember that depth-first search adds a set of paths to the front of the search queue, i.e. the search queue is of LIFO type or a stack. For example, if Q = (A,B,C,D) and the extended paths = (E,F,G), then the result of adding the extended paths to Q should be (E,F,G,A,B,C,D).

You must include in your homework submission:

1. *[src] The completed source codes of UninformedSearch.java.*
2. *[src] The completed source codes of BreadthFirstSearch.java.*
3. *[src] The completed source codes of DepthFirstSearch.java.*
4. *[pdf] The amount of time you spent on this problem.*

Problem 2 (20 points)

In this problem you will experiment with the algorithms you wrote in Problem 1 on the 2008 Battlecode platform. Battlecode was developed as part of the 6.370 programming competition.

1) Install Battlecode

- go to http://battlecode.mit.edu/2008/contestants/downloads to find the installer and instructions.

- After installing, place the "team16410" folder into the "teams" folder of your installation and place the ps2map.xml into the "maps" folder.
- We suggest running BattleCode using the Eclipse IDE. Follow the instructions at: http://battlecode.mit.edu/2008/docs/software.html#toc11
- Once the BattleCode dialog box pops up , select "team16410" for "Team A" and "team16410" again for "Team B"
- Clear all other maps from the map list with the minus button and then select the map "ps2map.xml" from the drop down and then press the plus button. "ps2map.xml" should be the only map appearing in the box.
- Click "Okay" to start the match.

The red player will execute the plan produced by your algorithm.

2) Collect data about the performance of your algorithms.
Modify RobotPlayer.java to collect the following information for each of the two *complete* algorithms,

1. Number of Extended Paths
2. Number of Visited States
3. Maximum Queue Size
4. Length of returned plan

Note: Look for comments `//Problem 2` and fill in code to collect the appropriate statistics.

- Collect these statistics for several different goal locations (The goal is changed by moving the location of the '`G`' in the ASCII-art portion of the constructor, Line 29. Take care to add a space where you remove 'G', and to remove a space where you add 'G'.)

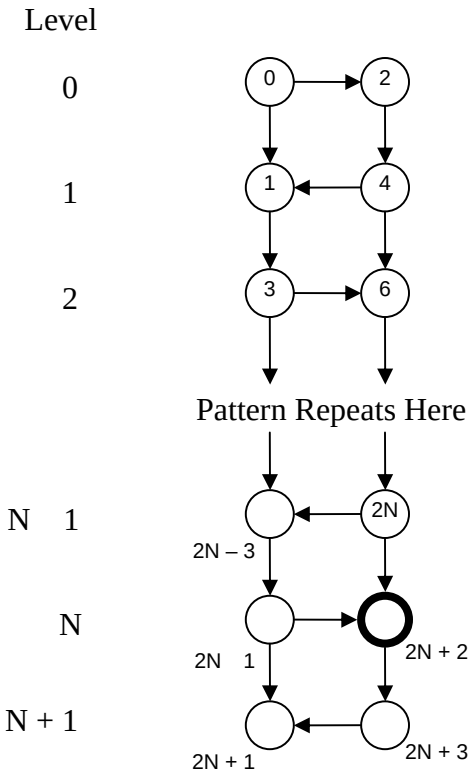
- Briefly comment on the statistics you collected for each search method.

- *Extra Credit:* Create a new map on which the above results are qualitatively different. Discuss the reasons for the difference. Note that to change the barriers on the map, you must edit both the ASCII art in RobotPlayer.java and the xml map file. The reason for this inelegance is that loading the map (or anything) from an external source is considered cheating in Battlecode. Therefore it had to be hard-coded.

You must include in your submission:

1. *[pdf] Execution trace of Problem2_2.java.*
2. *[pdf] Tabular form of the results.*
3. *[pdf] Discussion of the results.*
4. *[pdf] The amount of time you spent on this problem.*

Problem 3 (40 points)
Consider the following directed graph.



Assume that we are performing a search starting from Vertex 0, and our goal is Vertex 2N + 2. Search terminates when the goal has been found, as presented in class. Also, whenever faced with two equal choices, the vertex with the lower index is examined first.

Hint: Please note the direction of the arrow entering G. This direction implies that N will always be an even number, when determining your analytical expressions.

a. Derive a **precise analytical expression** for the number of paths that depth first search **examines**. Assume that **depth first search uses a visited list**. Note that there is a subtle distinction between the vertices examined and the paths that are merely placed on the queue, but whose fringe vertex is never examined. Next derive an analytical expression for the **largest number** of paths that will be under consideration (on the queue) at any given time. Note that these are the time and space requirements, respectively. Assume **N > 0**.

b. Repeat these two derivations for **breadth first search**, assume that a **visited list** has been used.

c. Repeat the two derivations for a breadth first search that **does not use a visited list.**

You must include in your submission:

1. *[pdf] Analytical expression for number of vertices examined and the maximum queue size for depth-first search with visited list.*
2. *[pdf] Analytical expression for number of vertices examined and the maximum queue size for breath-first search with visited list.*
3. *[pdf] Analytical expression for number of vertices examined and the maximum queue size for breath-first search without visited list.*
4. *[pdf] The amount of time you spent on this problem.*

16.410 / 16.413 Principles of Autonomy and Decision Making

Fall 2010