[Assignment 2] Solving the 2048 Game: online Graph Search

Due: 12:00 Noon, Friday, 21 of October

Purpose

The purpose of this assignment is for you to:

- Increase your proficiency in C programming, your dexterity with dynamic memory allocation and your understanding of data structures, through programming a search algorithm over Graphs.
- Gain experience with applications of graphs and graph algorithms to solving games, one form of artificial intelligence.

Assignment description

In this programming assignment you'll be expected to build a *solver* for the 2048 game. The game has been described by the Wall Street Journal as "almost like Candy Crush for math geeks". You can play the game compiling the code given to you using the keyboard, or using this web implementation http://2048game.com/.

The 2048 game

2048 is played on a 4x4 grid, with numbered tiles that slide smoothly when a player moves them using the four arrow keys. Every turn, a new tile will randomly appear in an empty spot on the board with a value of either 2 or 4. Tiles slide as far as possible in the chosen direction until they are stopped by either another tile or the edge of the grid. If two tiles of the same number collide while moving, they will merge into a tile with the total value of the two tiles that collided. The resulting tile cannot merge with another tile again in the same move.

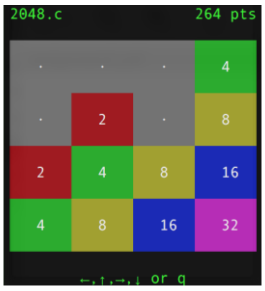

Figure 1: A possible configurations of the game 4x4 grid

A scoreboard on the upper-right keeps track of the user's score. The user's score starts at zero, and is incremented whenever two tiles combine, by the value of the new tile.

The game is won when a tile with a value of 2048 appears on the board, hence the name of the game. **After reaching the 2048 tile, players can continue to play (beyond the 2048 tile) to reach higher scores**. In this assignment, your solver should continue playing after reaching tile 2048. When the player has no legal moves (there are no empty spaces and no adjacent tiles with the same value), the game ends.

```
GraphSearch(Graph,start,maxDepth)
 1     node ← start
```

```
    explored ← empty Array
    frontier ← priority Queue Containing node Only
    while frontier 6= empty
    do
        node ← frontier.pop()
        explored.add(node)
        if node.depth < maxDepth
        then
            for each action
            do
                newNode ← applyAction(node)
                if newNode.board 6= node.board
                then
                    frontier.add(n)
                    propagateBackScoreToFirstAction(n)

                else
                    delete newNode

freeMemory(explored)
bestAction ← select best action breaking ties randomly
return bestAction
```

Figure 2: Online Graph variant of Dijkstra

The Algorithm

Each possible configuration of the 2048 4x4 grid is called a *state*. The 2048 Graph G = hV,Ei is implicitly defined. The vertex set V is defined as all the possible 4x4 configurations (states), and the edges E connecting two vertexes are defined by the legal movements (right, left, up, down).

Your task is to find the path leading to the higest score, i.e. leading to the most rewarding vertex (state). A path is a sequence of movements. You are going to use a variant of Dijkstra to explore the most rewarding path first, up to a **maximum depth** D.

Every time the game asks you for a movement (action), you should explore all possible paths up to depth D. Once you finished generating all the paths, you should return the **first action only** of the path leading to the highest score vertex. This action will be then executed by the game engine.

You might have multiple paths with the same maximum score. If more than one action (left,right,up or down) begins paths with the same maximum score, you'll have to break ties randomly.

Make sure you manage the memory well. Everytime you finish running the algorithm, you have to free all the nodes from the memory, otherwise you are going to run out of memory fairly fast.

When you *applyAction* you have to create a new node, that points to the parent, updates the board with the action chosen, updates the priority of the node with the new score, and updates any other auxiliary data in the node.

You are going to need some auxiliary data structures to update the scores of the first 4 applicable actions. The function *propagateBackScoreToFirstAction* takes the score of the newly generated node, and propagates back the score to the first action of the path.

This propagation can be either *Maximeze* or *Average* :

- If you *Maximize*, you have to make sure that the first action is updated to reflect the maximum score of any of its children up to depth D.
- If you *Average*, you have to make sure that the first action is updated to reflect the average score taking into account all its children up to depth D.

Deliverables, evaluation and delivery rules

Deliverable 1 – *Solver* source code

You are expected to hand in the source code for your solver, written in C. Obviously, your source code is expected to compile and execute flawlessly using the following makefile command: make generating an executable called 2048. Remember to compile using the optimization flag gcc -O3 for doing your experiments, it will run twice faster than compiling with the debugging flag gcc -g. For the submission, please **submit your makefile with** gcc -g **option**, as our scripts need this flag for testing.

Your implementation should achive scores higher than 5000 points.

Base Code

You are given a base code. You can compile the code and play with the keyboard. The default solver chooses an action randomly. You are going to have to program your solver in the file ai.c. Look at the file 2048.c to know which function is called to select an action to execute.

You are given the structure of a node, and also a priority queue implementation. Look into the utils.* files to know about the functions you can call to apply actions.

You are free to change any file.

Input

You can play the game with the keyboard by executing ./2048

In order to execute your solver use the following command:
```
./2048 ai <max/avg> <depth>
```
Where <max/avg> is either max or avg, to select the 2 options for propagationg scores, and <depth> is an integer number indicating the depth of your search.

for example:
```
./2048 ai avg 6
```
Will run average updates up to depth 6.

If you append the option "slow" at the end, it will slow the ai so you can see it playing
```
./2048 ai avg 6 slow
```

Output

Your solver will print into an output.txt file the following information:

1. Max Depth
2. Number of generated nodes.
3. Number of expanded nodes.
4. Number of expanded nodes per second.
5. Total Search Time, in seconds.
6. Maximum value in the board.
7. Score

For example, the output of your solver ./2048 ai avg 6 could be:

```
MaxDepth = 8
Generated = 499,911
Expanded = 253,079
Time = 7.05 seconds
Expanded/Second = 35,906,612 maxtile
= 2048 Score=14,000
```

These numbers are made up. We don't expect you to expand 35 million nodes per second. A node is expanded if it was popped out from the priority queue, and a node is generated if it was created using the applyAction function.

Deliverable 2 – Experimentation

Besides handing in the solver source code, you're required to provide a table with the mean score and deviation, mean max tile and deviation, and total execution time for each type of propagation (max/avg) you implement and each max depth from 0,..,6.

In order to test your solver, you have to average over multiple runs because 2048 has a random component: tiles can appear in different locations after each move. A sample of 10 runs is enough.

For each propagation type, plot a figure where the x axis is the depth, and y is the mean score.

Explain your results using your figures and tables. Which max depth works best? Is it better to propagate max or avg?

Answer concisely. **Please include your Username, Student ID and Full Name** in your Document.

Evaluation

Assignment marks will be divided into four different components.

1. Solver (11)
2. Code Style (1)
3. Experimentation (3)

Please note that you should be ready to answer any question we might have on the details of your assignment solution by e–mail, or even attending a brief interview with me, in order to clarify any doubts we might have.

Code Style

You can improve the base code according to the guidelines given in the first assignments. Feel free to add comments wherever you find convenient.

Extra 0.5 Mark

An extra 0.5 mark can be earned if you implement any (1 at least) of the tricks described in the following links:

- http://2048game.com/tips-and-tricks/
- http://2048game.com/videos/

Alternatively, you can implement any of the improvements below, taken from http://stackoverflow.com/questions/22342854/what-is-the-optimal-algorithm-for-the-game-2048:

Several heuristics are used to direct the optimization algorithm towards favorable positions. The precise choice of heuristic has a huge effect on the performance of the algorithm. The various heuristics are weighted and combined into a positional score, which determines how "good" a given board position is. The optimization search will then aim to maximize the average score of all possible board positions. The actual score, as shown by the game, is not used to calculate the board score, since it is too heavily weighted in favor of merging tiles (when delayed merging could produce a large benefit).

Initially, I used two very simple heuristics, granting "bonuses" for open squares and for having large values on the edge. These heuristics performed pretty well, frequently achieving 16384 but never getting to 32768.

Petr Mor´avek (@xificurk) took my AI and added two new heuristics. The first heuristic was a penalty for having non-monotonic rows and columns which increased as the ranks increased, ensuring that non-monotonic rows of small numbers would not strongly affect the score, but non-monotonic rows of large numbers hurt the score substantially. The second heuristic counted the number of potential merges (adjacent equal values) in addition to open spaces. These two heuristics served to push the algorithm towards monotonic boards (which are easier to merge), and towards board positions with lots of merges (encouraging it to align merges where possible for greater effect).

If you do any of the optimizations for the extra mark, please report and discuss it in your experimentation.

Delivery rules

You will need to make *two* submissions for this assignment:

- Your C code files (including your Makefile) will be submitted through the LMS page for this subject: *Assignments → Assignment 2 → Assignment 2: Code*.
- Your experiments report file will be submitted through the LMS page for this subject: *Assignments → Assignment 2 → Assignment 2: Experimentation*. This file can be of any format, e.g. .pdf, text or other.

Program files submitted (Code)

Submit the program files for your assignment and your Makefile.

Your programs *must* compile and run correctly on the CIS machines. You may have developed your program in another environment, but it still *must* run on the department machines at submission time. For this reason, and because there are often small, but significant, differences between compilers, it is suggested that if you are working in a different environment, you upload and test your code on the department machines at reasonably frequent intervals.

A common reason for programs not to compile is that a file has been inadvertently omitted from the submission. Please check your submission, and resubmit all files if necessary.

Plagiarism

This is an individual assignment. The work must be your own.

While you may discuss your program development, coding problems and experimentation with your classmates, you must not share files, as this is considered plagiarism.

If you refer to published work in the discussion of your experiments, be sure to include a citation to the publication or the web link.

"Borrowing" of someone else's code without acknowledgement is plagiarism, e.x. taking code from a book without acknowledgement. Plagiarism is considered a serious offense at the University of Melbourne. You should read the University code on Academic honesty and details on plagiarism. Make sure you are not plagiarizing, intentionally or unintentionally.

You are also advised that there will be a C programming component (on paper, not on a computer) on the final examination. Students who do not program their own assignments will be at a disadvantage for this part of the examination.

Administrative issues

When is late? What do I do if I am late? The due date and time are printed on the front of this document. The lateness policy is on the handout provided at the first lecture and also available on the subject LMS page. If you decide to make a late submission, you should send an email directly to the lecturer as soon as possible and he will provide instructions for making a late submission.

What are the marks and the marking criteria Recall that this project is worth 15% of your final score. There is also a hurdle requirement: you must earn at least 15 marks out of a subtotal of 30 for the projects to pass this subject.

Finally Despite all these stern words, **we are here to help!** There is information about getting help in this subject on the LMS pages. Frequently asked questions about the project will be answered in the LMS discussion group.

Have Fun! N.L.