CPSC 322 Assignment 2

Make sure you follow the course policies on Canvas. Failure to do so may result in heavy penalties.

Make sure that in your answers you clearly indicate the exact question/subquestion you are answering.

Start each question on a new page (eg. don't put your answer to Q3 on the same page as your answer to Q2).

Question 1 (10 points): Comparing Search Algorithms

Consider the problem of finding the best path from start state *S* to goal state *Z* in the following directed graph:

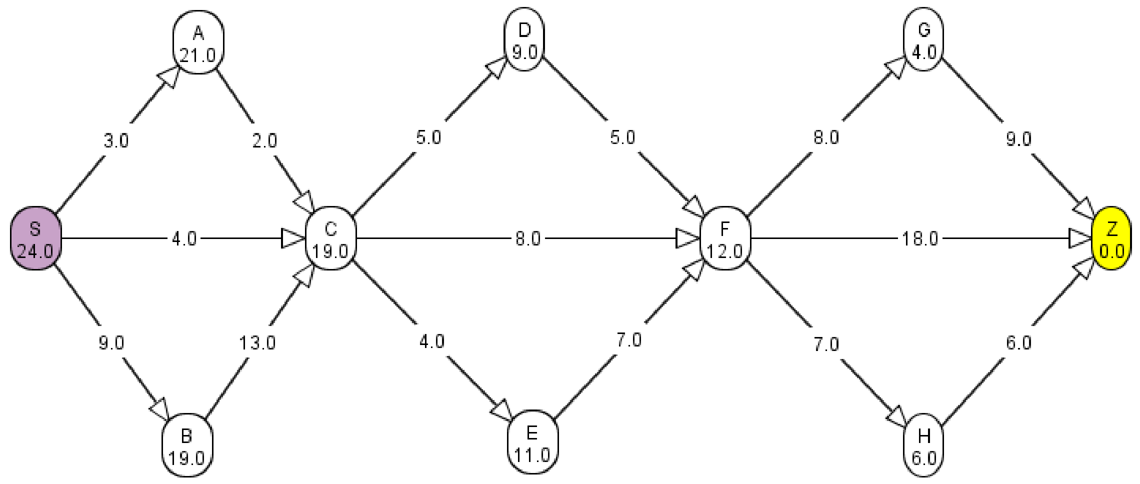


Figure 1. Graph for question 1. Nodes are labeled with h values, and arcs are labeled with costs.

For search strategies that do not consider costs when determining which path to remove from the frontier, **expand neighbours of a node in alphabetical order**; and when considering costs **break ties in alphabetical order**. For example, suppose you are running an algorithm that does not consider costs when removing paths from the frontier, and you expand **S**; you will add the paths *<S,A>, <S,B>* and *<S,C>* to the frontier in such a way that *<S,A>* is expanded before *<S,B>* or *<S,C>*. Also, the entire path is taken into account for determining alphabetical order; for example, *<S,A,C>* would be considered as coming before *<S,B,C>* alphabetically.

Additional notes:

- We will say that a node is **expanded** when the algorithm checks to see if it is a goal node.
- If a node is expanded multiple times, it must appear in your list of expanded nodes multiple times.

For the algorithms in questions 1.1 and 1.2 (as discussed in class), answer the following questions:

(a) What nodes are expanded by the algorithm? Order the nodes from first expanded to the last.
(b) What path is returned by the algorithm?
(c) What is the cost of this path?

1.1. **[3 points]** B&B
1.2. **[3 points]** A*

1.3. **[4 points]**

(a) [2 points] Which of the above strategies, if any, found the optimal solution for this graph? Would this result be true in general? Briefly justify your answer.
(b) [2 points] Did B&B expand fewer nodes than A*? Is this behaviour true in general? Briefly justify your answer.

Question 2 (15 points): Implementing Search

Write a program to implement DFS and BFS on the graph from Question 1, subject to the following requirements:

- You may not have separate code for performing DFS and BFS; you must be able to do either using a single function (**that you must implement yourself from scratch**) that you will call **`search`**. This search function cannot be allowed to know whether it is performing DFS or BFS; it must use the same logic in both cases.
    - This does not mean that your entire program must be a single function; rather, this means that the search logic will be contained within a single function (though it may call other functions in order to interact with various data structures).
- While you may not use any existing libraries to perform the search itself, you may use existing libraries for representing necessary data structures. If those data structures are used in your **`search`** function, you may need to provide wrapper/helper functions or use function pointers so that the search function cannot tell what kind of search it is performing.
- You may hardcode the graph from Question 1 and represent it however you like (feel free to use existing libraries to help with this).
- When expanding a node, you may add neighbour paths to the frontier in any order you wish. Note that this is different from Question 1.
- Your program must report (i.e. provide concise but readable output):
    - When it expands a node (and which node that is)
    - What path it returns (or some message if no solution path is found).
- You may use whatever programming language you like, but you must make sure that your code is readable; if you choose a relatively esoteric programming language, this means that you may need to provide additional comments in case the TAs are not familiar with that language.

You must submit your code and your program output for both BFS and DFS, such that it is clear which nodes were expanded and which solution paths were returned.

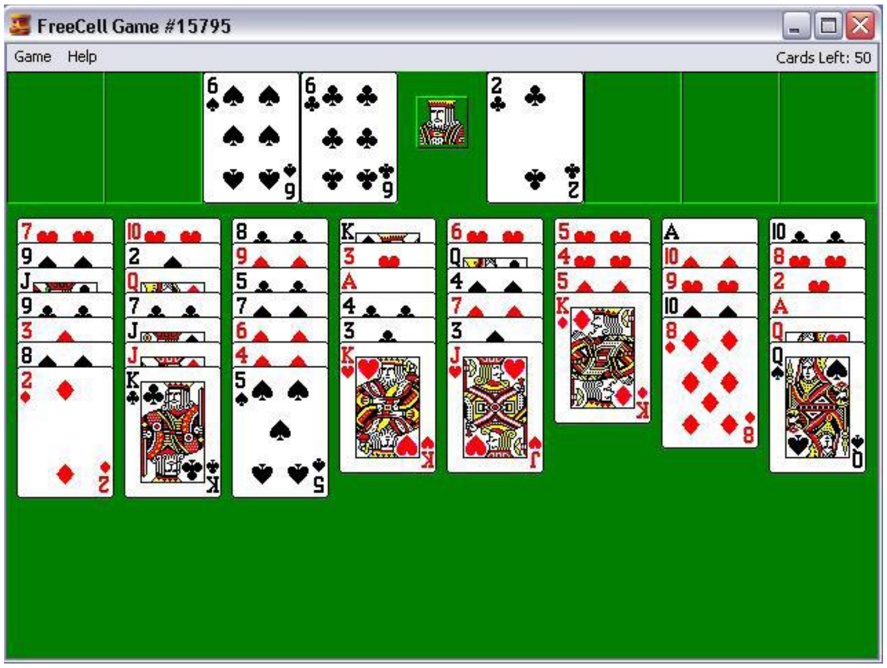
BONUS [5 points each] Update your program (and submit the code and program output as before) such that it can also perform:

- IDS
    - A recommended approach is to write an **`IDS`** function that repeatedly calls **`search`**.
- LCFS, BestFS, and A*
    - You should be able to do this without modifying your **`search`** function itself by using different wrapper/helper functions for interacting with certain data structures.
- B&B
    - In this case, it's fine to write a separate **`BandB`** function that is a modified version of **`search`**.
- searches on any graph that uses the same format as you have used.

Question 3 (15 Points): FreeCell

Consider the problem of trying to solve a game of FreeCell in the minimum number of moves.

FreeCell comes standard on nearly every version of Microsoft Windows (including Windows 10, under the Microsoft Solitaire Collection), and full rules are also available at the Wikipedia entry: https://en.wikipedia.org/wiki/FreeCell



The object of FreeCell is to move all the cards to the home cells [foundations], using the free cells as placeholders. To win, make four stacks of cards on the home cells [foundations], one for each suit, stacked in order of rank, from lowest (ace) to highest (king).

- *When moving cards to columns, cards must be moved in order from highest (king) to lowest (ace), alternating suit colors.*
- *When moving cards to home cells [foundations], cards must be moved in order from lowest (ace) to highest (king), same suit.*
- *A card from the bottom of a column can move to a free cell, the bottom of another column, or a home cell [foundation].*
- *A card from a free cell can move to the bottom of a column, or to a home cell [foundation].*

Microsoft Windows FreeCell help file

3.1. **[10 points]** Suppose you were trying to write a program to solve FreeCell using search.
- (a) **[4 points]** How would you represent a node/state in your code?
- (b) **[2 points]** In your representation, what is a goal node?
- **(c)** **[4 points]** How would you represent the arcs? ***Note:*** *you do* ***not*** *need to consider differences between valid and invalid moves; you may assume that some other part of your code would handle that.*

3.2. **[5 points]** Give an admissible heuristic for this problem; explain why your heuristic is admissible. More points will be given for tighter lower bounds; for example, “h(n)=0 for all n” is a trivial (and useless) heuristic, and thus it will award you no points.

Question 4 (15 points) Modified Heuristics

For this question you are to think about the effect of heuristic accuracy and admissibility on *A** search. That is, you are to experiment using a sample graph in the AIspace search applet, and think about how the closeness of *h(n)* to the actual cost from node *n* to a goal, and whether h(n) is greater or less than the actual cost, affect the efficiency and accuracy of *A**.

Use the sample graph ***"Vancouver Neighbourhood Graph"*** in AIspace with *h(n)* admissible. To achieve this, while in *Create* mode, you should first set the cost values automatically by clicking on the option *Set edge costs automatically* and then set the *h(n)* values to an underestimate by clicking on the option *Set node heuristic automatically* in the *Search Options* menu of the applet. See how *A** works with this setting and **report the number of nodes expanded as well as the optimal path found [1 point]**. **This is your baseline.**

Then you will need to experiment with changing the *h(n)* values as described below. You can change the *h*(n) value of individual nodes by clicking on '*Set Properties'* in Create mode and then right click on the desired node. You can also change the *h*(n) value globally by selecting '*set costs and heuristics*' under "*search options*" (but remember not to modify the arc costs).

Devise a series of "experiments" in order to explore the performance of *A** as the heuristic values change. Can you come up with heuristic values within the graph that improve or reduce the efficiency of A* (or even make it choose sub-optimal paths)?

Some questions you may want to consider include (but are not limited to) the following:

- What happens if you increase (or decrease) heuristics everywhere in the graph?
  - By the same amount? By different amounts?
- What happens if you increase (or decrease) heuristics on the solution path only?
  - Does the amount by which the values are changed matter?
- What happens if you increase (or decrease) heuristics everywhere ***except*** on the solution path?
  - Does the amount by which the values are changed matter?
- What happens if you set the heuristic values to be the exact minimum path costs from each node to a goal?
- What happens if you set one of the other nodes to be a second goal node? For example, is it possible to use heuristic values to make A* find a path to SRY instead of SP?

Some things to note:

- Goal node heuristics are to be kept at zero at all times.
- Arc costs should not be changed from their initialized values.
- During your experiments, you are not required to keep the heuristic values admissible.
- The term "efficiency" in this question does **not** refer to the worst-case asymptotic time complexity of A*, which we have established to be O(b^m).

[10 points] Report your findings. Give **brief (but informative)** descriptions of what you did and the results (particularly in terms of the numbers of nodes expanded and the solution paths found). You may include relevant screenshots to illustrate your report.
IMPORTANT: *We're not looking for a massive lab report or academic paper; you should be aiming for a small set of scenarios that sufficiently illustrate how the performance of A* can change with changes to heuristic accuracy and admissibility.*

[4 points] Summarize your findings (in a single sentence, if possible): **what makes a heuristic "better"?**