Darko Cejkov 160835590

Madhav Kashyap 173247600

Arhan Ikram 170399300

Sizhao Lin 150531100

# CP468 Artificial Intelligence: Assignment #1

Part 1

# Question 1: T/F? Explain:

1. DFS expands at most as many nodes as A\* search with an admissible heuristic

an A\* search on a tree with an admissible heuristic means that A\* is optimal (reaches the goal node of depth *d* in the shortest cost), so it expands few nodes. DFS expands all deepest nodes of the tree, as it is an uninformed search algo., so assuming that depth *d* is the depth of the goal node, A\* expands less nodes because it is informed and can make better choices on expansion. Thus, DFS cannot at most expand as many nodes as A\* as it is not as optimal as A\*; this is **false**.

1. *h(n) = -5* is an admissible heuristic for the 8-puzzle

**False**, since a heuristic in the sense of the 8-puzzle cannot be a negative number, nor is it a “heuristic”, it’s simply a result of an unknown heuristic h(n); it also cannot be considered admissible since it doesn’t overestimate, but it doesn’t estimate anything.

1. Breadth-first Search is only complete even if zero step costs are allowed

**True**, since BFS is complete whenever the branching factor is finite. BFS’s completeness does not depend on step-costs, but if zero step-costs are allowed then it’s optimality depends on whether *all* costs are equal (all zero in this case)

1. In a search tree, not every explored node is connected to the initial state by a path of explored nodes.

**False**, since in order to maintain the structure of a search tree, all nodes must be connected to any parents and children. So, visiting a node implies that you arrived at it via it’s parent, and an assumption is that every node other than the root node has a parent. Thus, every explored node is connected to the initial state by a path of previously explored nodes.

# Question 2: Exercise 3.15:

    “Consider a state space where the start state is number 1 and each state k has two

successors: numbers 2k and 2k + 1”

1. Draw the portion of the state space for states 1 to 15

|  |  |
| --- | --- |
|  |  |

1. Suppose the goal state is 11. List the order in which nodes will be visited for breadth-first search, depth-limited search with limit 3, and iterative deepening search.

BFS: 1 → 2 → 3 → 4 → 5 → 6 → 7 → 8 → 9 → 10 → 11

DLS(3): 1 → 2 → 4 → 8 → 9 → 5 → 10 → 11

IDS:

1. limit = 0: 1
2. limit = 1: 1 → 2 → 3
3. limit = 2: 1 → 2 → 4 → 5 → 3 → 6 → 7
4. limit = 3: 1 → 2 → 4 → 8 → 9 → 5 → 10 → 11
5. ~~How well would bidirectional search work on this problem? What is the branching~~

~~factor in each direction of the bidirectional search?~~

~~Bidirectional searching~~ *~~can~~**~~work~~* ~~on this problem, since every node has at most 1 successor, making it very simple to proceed backwards from the goal using BFS. The branching factor~~ *~~b~~* ~~forwards (from root) is exactly 2 (since every parent has 2 children; 2k & 2k+1), and~~ *~~b~~* ~~backwards is exactly 1, since every node has at most 1 parent (except root); and each predecessor is easily computable as it is just floor(k/2).~~

1. ~~Does the answer to (c) suggest a reformulation of the problem that would allow you to~~

~~solve the problem of getting from state 1 to a given goal state with almost no search?~~

~~Yes, if we were to use BFS both ways within bidirectional searching, we cut down the amount of searching since we find our selection via the intersection of frontier nodes of both searches.~~

1. ~~Call the action going from k to 2k Left, and the action going to 2k + 1 Right. Can you~~

~~find an algorithm that outputs the solution to this problem without any search at all?~~

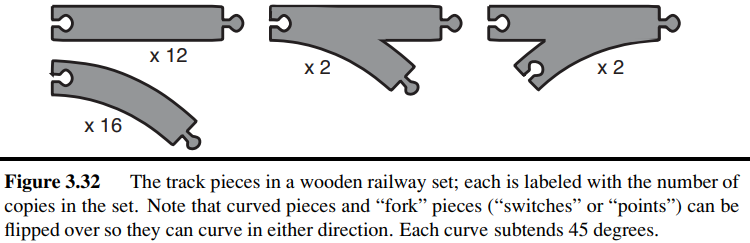
~~We know that 2k is even and 2k+1 is odd, so every odd number is on the right,~~

~~and since every node has 1 parent whose value is floor(k/2); we start at the goal state and “search” towards 1 very easily because the breadth of every node is now 1 (backwards) so we don’t search.~~

# Question 3: Exercise 3.16:

    “A basic wooden railway set contains the pieces shown in Figure 3.32. The task is to

connect these pieces into a railway that has no overlapping tracks and no loose ends where a train could run off onto the floor.



1. Suppose that the pieces fit together exactly with no slack. Give a precise formulation of

the task as a search problem.

Each track piece has an insert and a tab. Two track pieces fit in together by putting their tabs into another track’s insert, of the same or different type. No piece can overlap. Curved/forked pieces can be flipped to change the direction.

The initial state starts with any piece, and the goal is to combine all pieces such that they form a loop with no disconnections.

Goal test: all pieces are used in a single connected track, no open pegs or holes, no overlapping tracks.

Step costs: 1 per piece.

1. Identify a suitable uninformed search algorithm for this task and explain your choice.
   * 1. With all the constraints on the problem, there are many resulting states within the state-space. There are 32 pieces, so in order to connect all 32 pieces, there needs to be a depth of 32 (which is quite large) since each node is adding 1 piece, and there are many branching nodes because there are 4 types of pieces available to add on, and 3 of those types can be flipped.
     2. That is why I think **Depth-first search** would be the most efficient uninformed algo. for this task because it fully explores the depth of the subtree, removing it from memory to make space for the other subtrees.
2. Explain why removing any one of the “fork” pieces makes the problem unsolvable.
   * 1. If we were to remove a single instance of either of the 2 types of fork pieces, then there is no solution because if we were to create a split track, the only way to create a single, larger loop is to recombine the 2 tracks using a fork.
     2. If we were to remove fork pieces entirely, there is no way to put together 12 straight pieces and 16 curved pieces into a single, non-overlapping loop since there are more curves than straight pieces; since each curve is a 45°, 2 are required to make a 90° change (like the corner of a square), and a square loop has 4 corners, meaning 4\*2 = 8 curved pieces. Thus, you must have and equal amount of forked out and forked in pieces to accommodate for this.
3. Give an upper bound on the total size of the state space defined by your formulation.

(Hint: think about the maximum branching factor for the construction process and the

maximum depth, ignoring the problem of overlapping pieces and loose ends. Begin by

pretending that every piece is unique.)

    We can think about the amount of open pegs (both tabs and slots) as a goal,

since if all pieces were connected, there are no open pegs. At the beginning there is 2 (1 tab, 1 slot) assuming we start with a straight piece. The maximum amount of open pegs in any state is 3, if we add a forked piece, since it has an extra open peg. If every piece were unique, then the amount of choices for each peg is:

    12 [straight] + (2 \* 16) [curved] + (2 \* 2) [2 tab fork] + (2\*2\*2) [2 slot fork] = **56**

(we multiply each curved/forked pieces by 2 to account for the fact each of those pieces can be flipped.)

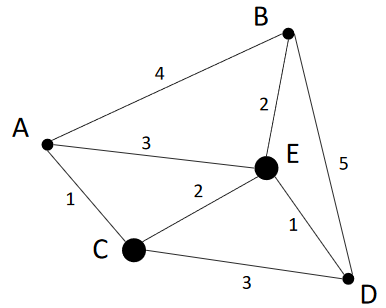
Since there is a max of 3 pegs to add a piece to, 56\*3 = 168 choices for each of the 3 pegs. Thus, the upper bound is: .

# Question 4:

    “Imagine that the following graph represents a road map of cities. Each city has a road

        connecting it to another city. The numbers on the roads indicate how long it takes to

travel between each other.”

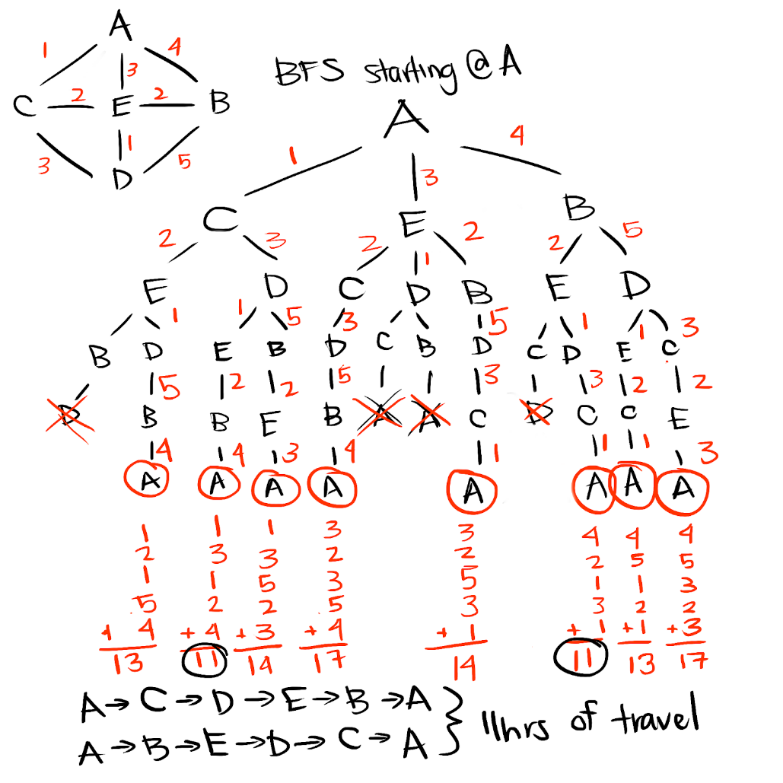
<https://gyazo.com/90bc75d6e6af2e9029cbc90b32ebc002>

Suppose that you are arranging for a travel from your home city ‘A’, to visit each city on the given map, only once, and return to your home city at the end of the trip.

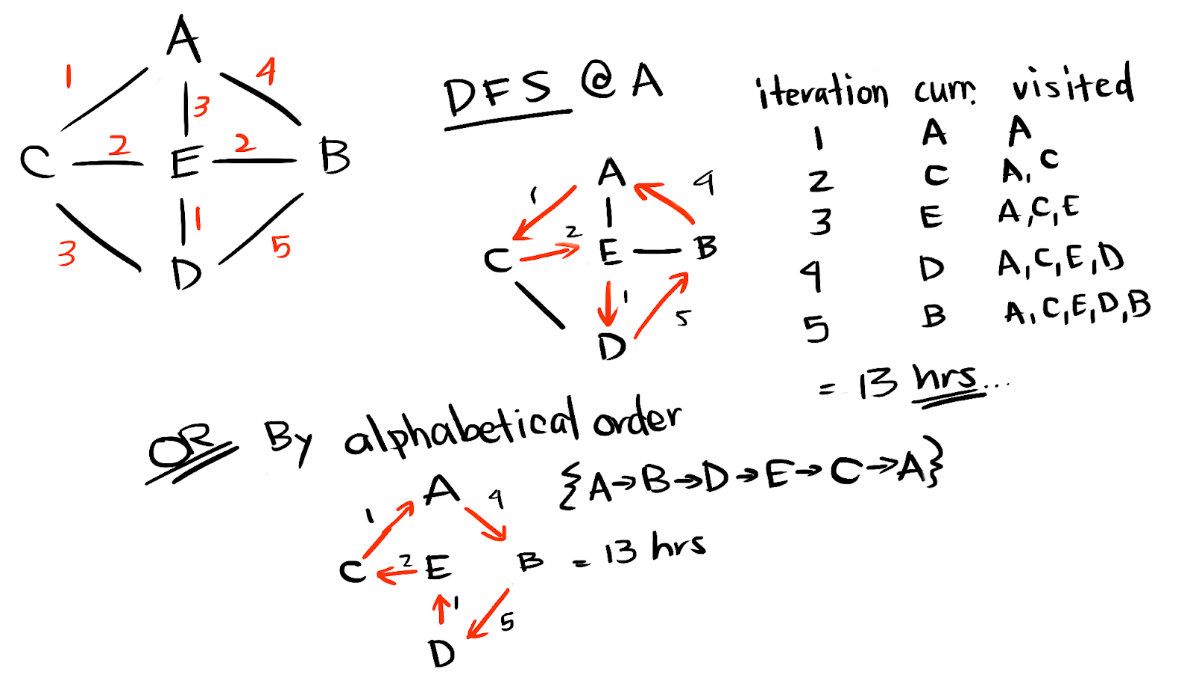
* ex. ABDCEA is one route that would take 17 hours, the aim is to choose a route the reduces time spent traveling

Try the following search strategies and list the order in which the cities are visited and its    cost

* 1. BFS

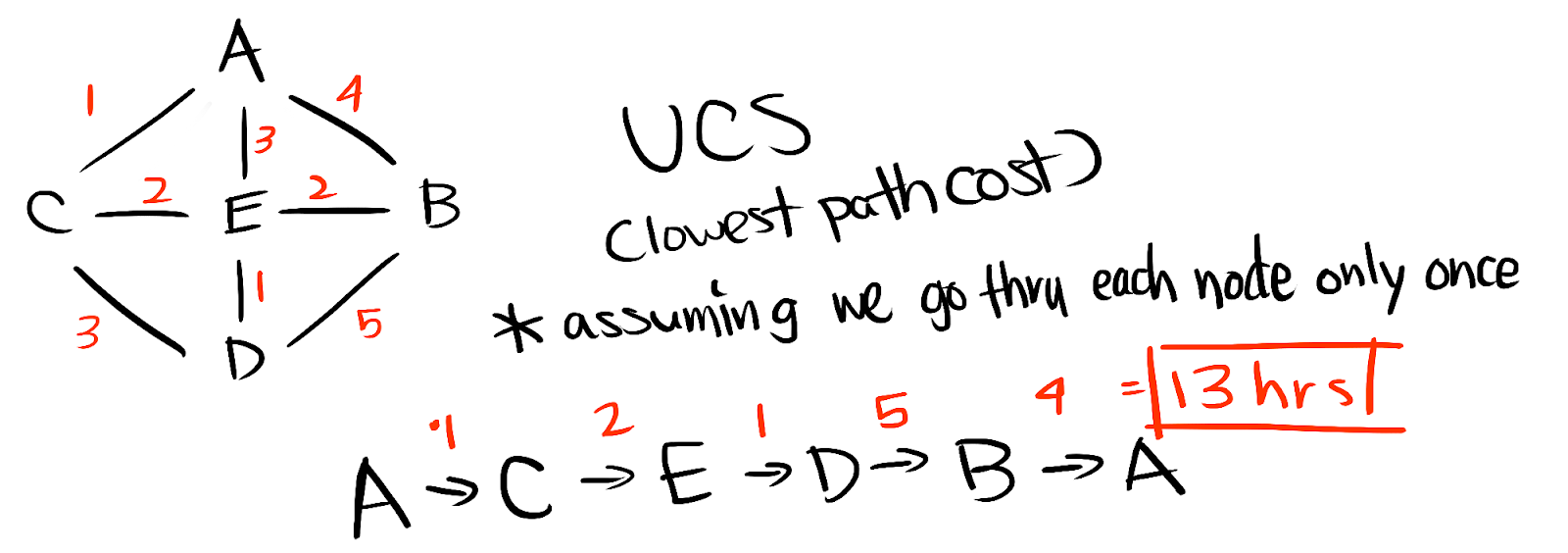


* 1. DFS
     + 1. The DFS traversals can be seen in the previous illustration as well in the search tree; this is a graph demonstration:



* 1. UCS

Uniform Cost search is based on the lowest cost paths



1. Compare time/space requirements for BFS/DFS/UCS

b = #non-root nodes / #non-leaf nodes = 42 / 31 = 1.35

d = depth of solution = 5

m = max depth = 5

C\* = optimal path of solution = 5

epsilon = the least cost of any step = 1

* + - * 1. BFS;

time: bd = 1.355 = **4.48**

space: bd = **4.48**

* + - * 1. DFS;

time: bm = 1.355 = 4.48

space: b\*m = 1.35\*5 = 6.75

* + - * 1. UCS;

time: b1+floor(C\*/epsilon) = 1.351+5 = 1.356 = 6.05

space: b1+floor(C\*/epsilon) = 1.351+5 = 1.356 = 6.05

Thus, BFS and DFS are tied for the best time complexity, with 4.48.

BFS emerges as the best for space complexity out of DFS and UCS, as

space(DFS) > space(UCS) > space(DFS).

Part 2: Maze solver

Repository can be found at: <https://github.com/darkocejkov/mazesearch>

Maze characters were converted to others to aid with clarity.

[■ = wall, ○ = unexplored node, ● = visited node, . = path of solution]

# Algorithm: Breadth-first Search [uninformed]

|  |  |  |
| --- | --- | --- |
| Size | Solutions | Statistics |
| Medium | A picture containing building, rain  Description automatically generated | Solution Cost = 42  Nodes Expanded = 224 |
| Large | A picture containing rain  Description automatically generated | Solution Cost = 63  Nodes Expanded = 777 |

# Algorithm: Depth-first Search [uninformed]

Results vary depending on the priority of neighbors in queue

Ex. Up > Down > Left > Right has a different result than Up > Left > Down > Right

This specific solution is based on Up > Down > Left > Right

|  |  |  |
| --- | --- | --- |
| Size | Solutions | Statistics |
| Medium | A close up of a logo  Description automatically generated | Solution Cost = 52  Nodes Expanded = 212 |
| Large | A picture containing rain, light, white  Description automatically generated | Solution Cost = 166  Nodes Expanded =214 |

# Algorithm: (Greedy) Best-first Search [informed]

Algorithm based on a heuristic function, specifically Manhattan Distance *h(n)* in this case, choosing the node with the lowest as priority.

|  |  |  |
| --- | --- | --- |
| Size | Solutions | Statistics |
| Medium | A picture containing rain  Description automatically generated | Solution Cost = 56  Nodes Expanded = 79 |
| Large | A close up of a person  Description automatically generated | Solution Cost = 76  Nodes Expanded = 134 |

Manhattan Distance = |x1 – y1| + |x2 – y2|

# Algorithm: A\* pathfinding [informed]

Like Best-first search, except also considering the cost to reach every node from the start *g(n)*  by adding it to the Manhattan distance; f(n) = h(n) + g(n)

|  |  |  |
| --- | --- | --- |
| Size | Solution | Statistics |
| Medium | A picture containing rain  Description automatically generated | Solution Cost = 56  Nodes Expanded = 79 |
| Large | A picture containing woman, white  Description automatically generated | Solution Cost = 76  Nodes Expanded = 134 |