Chue Zhang

Professor Lucci

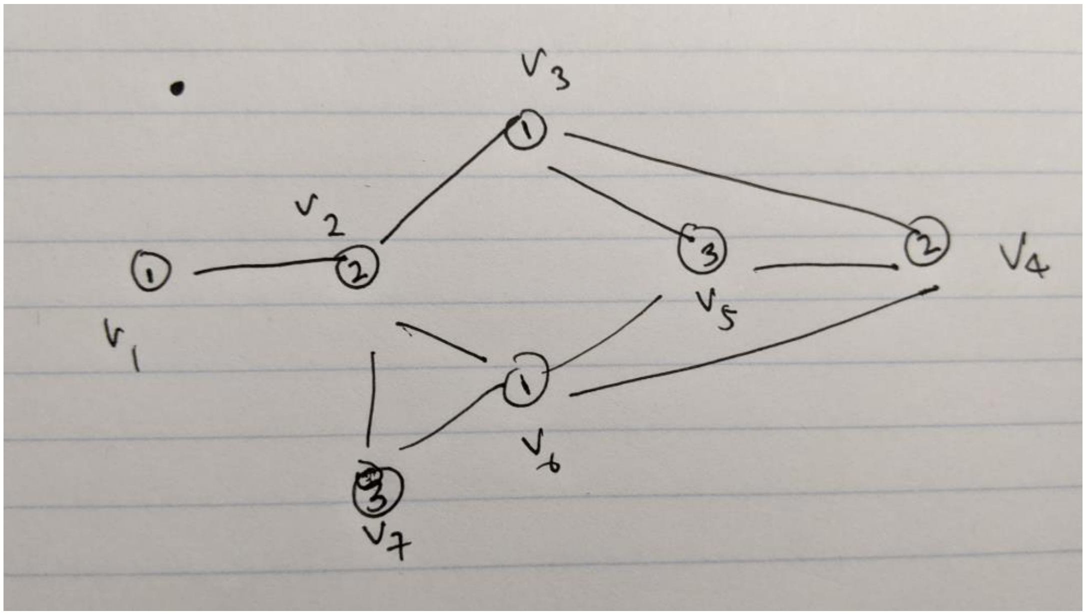
CSC 448

Exam_448

1.

```
For node in all_nodes: #NODE COLORS REPRESENTED AS 1 2 OR 3
    If (NOT Get color of current_node): #if no color on curr node
        Set color to 1
    Get neighbors
    For neighbor in neighbors:
        If neighbors color = color of current_node
            fill current_node with different color
        else continue
```

The algorithm I employ for this problem is the greedy algorithm such that no two adjacent vertices have the same color. If any adjacent vertices are of the same color, then swap the current nodes color to something else. This will run recursively until every node is colored.



As shown in the image above, We are traversing through the graph from points v1 to v7 (numerical order) .

1) In v1, there is no color so fill with 1
2) In v2, no color so fill with 1, You check neighbor and see an Adj 1
3) v2 color changed to 2
4) v3 no color, fill in 1
5) v4 no color, fill in 1, You check neighbors and see Adj 1
6) v4 color changed to 2
7) v5 no color, fill in 1, You check neighbors and see an Adj 1
8) v5 color change to 2, you check neighbors and see an Adj 2
9) v5 color change to 3

10) v6 no color, fill in 1
11) v7 no color, fill in 1, Check neighbor and see a 1
12) v7 change color to 2, Check neighbor and see a 2
13) v7 change color to 3
14) ALL NODES FILLED

2.

Best-first-search applies the use of heuristics whereas hill-climbing is basically the optimization of a solution. BFS looks for the best node to go to where as hill-climbing would try to look for the best mathematical solution. The reason why best-first-search would be better is that it will be going for the most optimal path where as hill-climbing might not optimize for the best path, just a better path. Best-first-search yields better initial results whereas hill-climbing only excels in local searches. The best way to visualize hill-climbing is that there are two hills, a and b where b is greater than a. what hill climbing will do is it will find the maximum of a and call it the highest when it hasn't even evaluated b. Hill-climbing requires context where as best-first search does not require context and will find the most optimal path through best values

3.

These 3 are the possible solutions for going from start S to goal G.

How the branch and bound method works is that it branches off of the smallest option available starting from the left node until it reaches the goal. For this case what will happen is

S → A = 1

S → C = 2

EXPAND A BECAUSE SMALLER

S → A → C = 4

S → A → B = 5

EXPAND C BECAUSE SMALLER

S → A → C → D = 7

S → A → C → B = 6

S → A → C → E = HALT

EXPAND B BECAUSE SMALLER

S → A → C → B → D = 9

EXPAND D SINCE GOAL REACHED NEXT AND NO OTHER NODE

S → A → C → B → D → G = 10

Now that the fist solution is made, the algorithm will backtrack and look for more optimal solutions where the next solution will be

S→A→C→D→G = 8

Then onto the next solution

S→A→B→D→G = 8

Then finally to the best solution

S→C→D→G= 6

Which is will accept.

4a.

We look at the tree and need to notice what are the min and max levels

Nodes E through J are Max level nodes so we take the largest value

Nodes B through D are Min level nodes so we take the smallest value

Node A is a Max level node.

So working under this pretense we get

E: 3

F: 5

G: 7

H: 8

I: 10

J: 12

Where I – J are max level and we take the largest of its value and next we evaluate the min level.

B: 3

C: 7

D: 10

Where B – D are min level and we take the smallest value and finally we evaluate the max level of A

A: 10

So in the end, it traverses like so

A → D → I → T

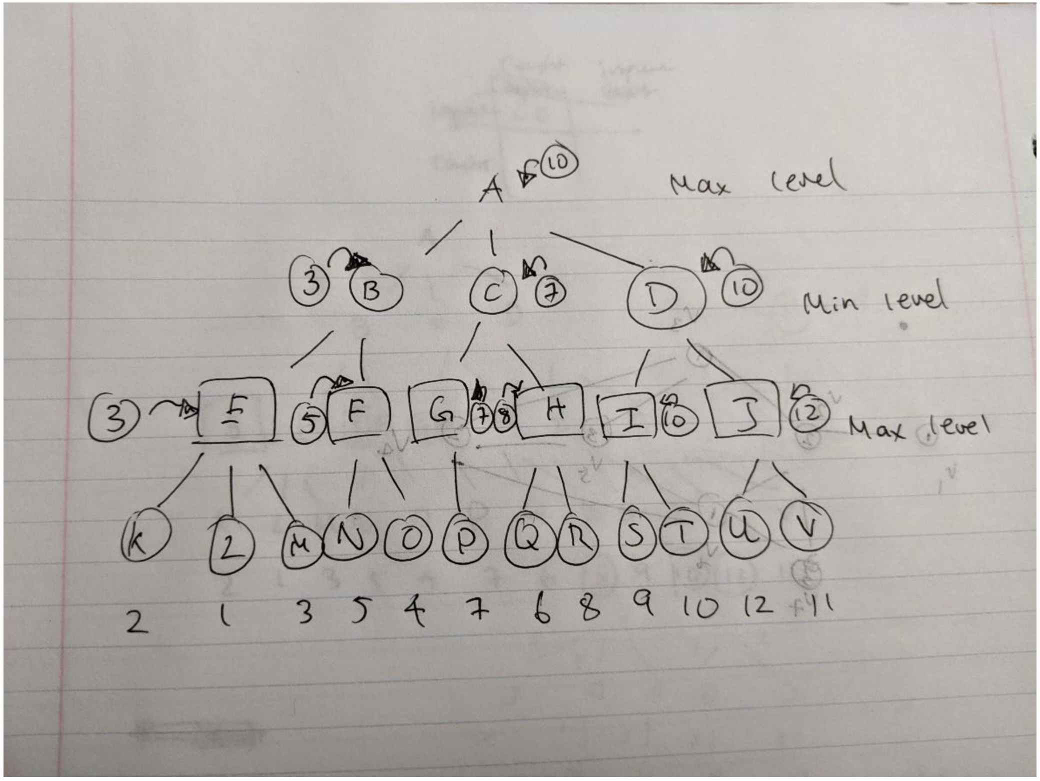
4b.

Applies the same methodology of Min-Max level where there are mini levels and max levels that you have to consider for.

So working on our current graph

1. We traveled all the way down left until we reach leaf Node K
2. We want to make sure no value is larger than leaf Node K = 2
3. B >= 3 because Max node in E is 3
4. Prune F because it is greater than 3
5. Move to node C where leaf node P = 7 is leftmost value
6. C >= 7 because no node greater than 7
7. H is pruned as its greater than C
8. We move to leftmost leaf node of D which is leaf node S = 9
9. D >= 10 because 10 is greater than 9
10. J is pruned as its value is greater than 10
11. So now we are left with B = 3, C = 7 and D = 10
12. So now look for Max node which is D = 10 there for   A = 10
13. The path taken is A → D → I → T

What was pruned : *F, H, J* Since there is no point in traversing there



5.

Basically, you cannot distribute the source-code without any improvements

Defecting is when you keep the improvements to yourself or distribute only source code without improvement.

Payoff Matrix

a. You modify the source code, you distribute = LEGAL
b. You modify the source code, you don't distribute = ILLEGAL
c. You don't modify, you distribute = ILLEGAL
d. You don't modify, you don't distribute = LEGAL

What happens here is that you can keep the improvements to yourself and do whatever you want with the code and not get any litigation or the opposite may happen and you get caught and go to jail. This same logic applies to distributing the source code without any improvements where distributing the source code is just releasing it as your own or uploading it somewhere.

This is very similar to the prisoners dilemma where it uses the **Nash equilibrium** to show the options available and the same can technically be applied for this payoff matrix where you either you get caught or you don't.

For the paretos optimization, it's the balance of the pay off matrix where you try to yield the best possible results while losing the least amount but applying it to our payoff matrix, if you get caught, you're screwed anyways so there is no optimization for this problem.

	Modified	Not modified
Distribute	No Litigation	Litigation
Don't Distribute	Litigation	No Litigation

6a.

DFS – What will happen is that it will backtrack from the possible options and backtrack from there. For example, the final couple options could be something like, Wolf goes with human last and Wolf goes with sheep last and then check for the validity of each of these options and go with the choice that works. It's a state transition with possible moves. An example of how it the back tracking will work is it will find the valid state where everything is at the other side, and test to see which method is valid going from left to right.

Open = [FWSC |], Closed = []

Open = [(SC | FW), (WC | FS) , (WS | FC)], **Closed** = [(FWSC)]

Open = [(W | FCS),(C | FWS)] **Closed** = [(WC | FS) , (FWSC)]

Open = [(FW | CS), (FCW | S), (FSW | C)] **Closed** = [(W | FCS), (WC | FS), (FWSC)]

Open = [(CW | FS), (FW | CS), (FS | CW)] **Closed** = [(FCW | S), (W | FCS) , (WC | FS), (FWSC)] **NO VALID STEPS, POP**

Open = [(FW | CS), (FCW | S), (FSW | C)] **Closed** = [(W | FCS), (WC | FS), (FWSC)]

Open = [(SW | FC), (S | FWC), (W | SFC)] Closed = [(FSW | C), (W | FCS), (WC | FS), (FWSC)]

Open = [(SF | WC), (SFC | W), (SFW | C)] Closed = [(S | FWC), (FSW | C), (W | FCS), (WC | FS), (FWSC)]

Open = [(SFWC)]Closed = [(SF | WC) (S | FWC), (FSW | C), (W | FCS), (WC | FS), (FWSC)]

Closed = [(| SFWC) , (SF | WC) (S | FWC), (FSW | C), (W | FCS), (WC | FS), (FWSC)] **DONE WITH SEARCH, GOAL HAS BEEN REACHED**

6b.

BFS – How BFS would work on this problem is that it will test all possible valid steps first level by level and continue with the steps that are valid

On the first level, it will be the decision of choosing to move the sheep, the wolf and the cabbage.

The algorithm will move on each one of those steps in a tree diagram and run the function recursively, going through each nodes of the sheep, wolf and cabbage path until they cannot go any further or until the goal is reached

Open = [(FCWS |)] Closed = []

Open = [(WS | FC), (CS | FW), (CW | FS)] Closed = [(FCWS |]

Open = [(CS | FW), (CW | FS)] Closed = [(WS | FC), (FCWS |)]

Open = [(CW | FS)] Closed = [(CS | FW), (WS | FC), (FCWS |)]

Open = [(FCW | S), (FSCW |] Closed = [(CW | FS), (CS | FW), (WS | FC), (FCWS |)]

Open = [(C | FWS), (W | FCS)] Closed = [(FCW | S), (CW | FS), (CS | FW), (WS | FC), (FCWS |)]

Open = [(W | FCS), (FCW | S), (FSC | W)] Closed = [(C | FWS) , (FCW | S), (CW | FS), (CS | FW), (WS | FC), (FCWS |)]

Open = [(FCW | S), (FSC | W), (FSW | C), (FCW | S)] Closed = [(W | FCS), (C | FWS) , (FCW | S), (CW | FS), (CS | FW), (WS | FC), (FCWS |)]

Open = [(FSC | W), (FSW | C), (FCW | S) ,(W | FCS), (C | FWS)] Closed = [(FCW | S), (W | FCS), (C | FWS) , (FCW | S), (CW | FS), (CS | FW), (WS | FC), (FCWS |)]

Open = [(FSW | C), (FCW | S) ,(W | FCS), (C | FWS), (S | FCW), (C | FSW)] Closed = [(FSC | W) ,(FCW | S), (W | FCS), (C | FWS) , (FCW | S), (CW | FS), (CS | FW), (WS | FC), (FCWS |)]

……. Work until [(| FSWC) is reached

ITS GETTING MESSY SO THIS ALGORITHM WILL CONTINUE LEVEL BY LEVEL , IF VALID, UNTIL THE GOAL STATE IS REACHED.

7.

			YELLOW, END
		🟩	▨
	🟩	▨	
🟩	▨	BLACK	
GREEN, START		BLACK	

ALL GREENS = Manhattan distance heuristic

LIGHT GREENS = A* search application with Manhattan distance applied

Manhattan Distance = |X_Start – X_Destination| + |Y_Start – Y_Destination|

Suppose, Start is (1,1) and destination is (4,5)

|1-4| + |1-5| = 7

According to the Manhattan distance heuristic, it will take only 7 Steps to reach yellow as shown by the table above. This will also be the shortest path needed to reach the end state.

Also, how A* search works is that it will go to the node that is the closest to the end so let’s apply the Manhattan Heuristic onto A*

The heuristic also states that A* with Manhattan distance heuristic will find the path that basically follows a straight line, kind of a zig zag motion as shown in the table above as if the zig zag line is straightened, it will lead almost straight to the end state.

8a. The Turing test for intelligence is the measurement of an AI to see if it is intelligent or not. The definition of intelligence is that if it can hold a conversation with a human without being detected as an AI then it is deemed intelligent. The simplest way to interpret the Turing test is if the computer can deceive an interrogator, then we can say that it passes the Turing Test

8b. What Searle criticizes that machinery only know about rules that they have to follow to manipulate things but have no understanding of the meaning of words or its semantics. What it is believed he is looking for is a machine that is not perfect but a machine that can understand. It’s a common example of Functionality vs Implementation where the robot is merely functioning and not implementing where to implement is to understand.

8c. This question is difficult to answer because what can having faster computers do for the intelligence. Could it help the robot have a better understanding of words and work towards less functionality and more towards implementation? Or is it just going to know how to apply rules faster. To be human in Searle’s criticism is to understand but a robot with faster computation could just computer faster under the pretense that it is understanding. So, I don’t really think faster computers will work very well. Like our brains, we need to experience certain things to understand it, we need to make machines work like how our brain does at processing information. I think the quote “work smarter not harder” applies for this exact scenario.