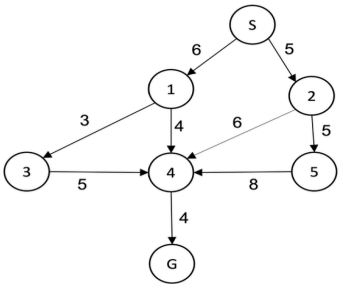
Name: Freya Zou

HW2: Search

1. (**30 points**) For the graph in Figure 1, implement A* to find the shortest path between nodes *S* and *G* using the heuristic provided in Figure 1. Write the following: (i) the shortest path; (ii) solution cost of the shortest path; (iii) the number of nodes expanded; and (iv) time taken to solve. Include a screenshot of your code displaying the answers to (i) - (iv). You are provided with a skeleton code in Python. You will need to fill in the missing lines in the code. You can also choose to implement A* in a different programming language of your choice.



Node	h value
S	7
1	6
2	5
3	4
4	2
5	3
G	0

Figure 1: Graph for informed search

(i) The shortest path

S-1-4-G

(ii) Solution cost of the shortest path

6+4+4+0 = 14

(iii) The number of nodes expanded

6

(Note: According to the provided course material and lecture slides G node is not expanded because we stop upon reaching it and do not generate any successors from it. Thus my answer is 6 not 7)

(iv) The time taken to solve. Include a screenshot of your code displaying

```
Shortest path ['S', '1', '4', 'G']
Solution cost: 14
Number of nodes expanded: ['S', '2', '1', '4', '3', '5']
Time taken to solve: 0.00001 seconds
```

2. **(10 points)** For the graph in Figure 1, implement A* with the heuristic in Figure 2 that has a value of 1 at all nodes except the goal node, where it is 0. Write the following: (i) the shortest path; (ii) solution cost of the shortest path; (iii) the number of nodes expanded; and (iv) time taken to solve. Include a screenshot of your code displaying the answers to (i) - (iv).

Node	S	1	2	3	4	5	G
h value	1	1	1	1	1	1	0

Figure 2: Heuristic for Q2

(i) The shortest path

S-1-4-G

(ii) Solution cost of the shortest path

14

(iii) The number of nodes expanded

The number of nodes expended might be 5 or 6

(Node***Because when we get to f(4) = f(5) = 11. Depending on the order of expansion, if A* explores node 4, it will find the goal node G and termite without needing to expand node 5, the result is [s,2,1,3,4]. However if node 5 is chosen to be expanded before node 4, it will
result in 6 expanded nodes [s,2,1,3,5,4]. In the problem, didn't specify tie-breaking strategies when nodes have equal f values.)

(iv) Include a screenshot of your code displaying the answer to (i)-(iv)

```
Shortest path ['S', '1', '4', 'G']
Solution cost: 14
Number of nodes expanded: ['S', '2', '1', '3', '5', '4']
Time taken to solve: 0.00001 seconds
```

3. (15 points) For the graph in Figure 1,

(i) provide an admissible heuristic that is NOT consistent.

h(s) = 7, h(1) = 6, h(2) = 12, h(3) = 4, h(4) = 3, h(5) = 3, h(G)= 0

(ii) Explain or prove how it satisfies admissibility and how it violates consistency.

To verify if the given heuristic is admissible, we need to check if it never overestimates the actual cost from each node to the goal. The actual cost from S to G is 14 (S-1-4-G). All the heuristic values from part(i) are less than and equal to the real cost 14, thus the heuristic I provided is admissible.

To check if the heuristic above is consistent, we must verify if, for every node n and its neighbor n′, the following condition holds: $h(n) \leq C(n, a, n') + h(n')$

let's check all the heuristics values to see if satisfy the condition above:

For node S:

$h(S) = 7 \leq C(n, a, n') + h(n') = 6 + 6 = 12$ (satisifed)

$h(S) = 7 \leq C(n, a, n') + h(n') = 5 + 2 = 7$ (satisfied)

For node 2:

$h(2) = 12 \leq C(n, a, n') + h(n') = 5 + 12 = 17$ (not satisfied)

Therefore we know the heuristic I provided is not consistent.

(iii) Solve the graph (you can solve it manually; no need to implement) using your heuristic and provide the solution path and cost. Does it find an optimal solution?

1. Sart Node S. Node S has two children node 1 and node 2:
   f(s) = g(S)+h(S) = 0 +7 = 7
2. Evaluate the children of node S:
   - Node 1 has two children: node 3 and node 4.
     f(1) = g(1)+h(1) = 6+6 = 12
   - Node 2 has two children: node 4 and node 5.
     f(2)=g(2)+h(2)=12+5=17

   Since f(2) = 17 is greater than f(1)= 12, we explore node 1's children.
3. Evaluate the children of node 1:
   f(3) = g(3)+h(3) = 9 +4 = 13
   f(4) = g(4)+h(4) = 10 + 3 = 13
   Node f(4) = 13 = f(3), so we can explore node 4's children or node 4's children. let's choose node 4's Children
5. Node 4 has only one child, which is G.
   f(G) =g(G) +h(G)= 14 +0 = 14. We have reached the Goal node

Therefore, It does find an optimal solution, the path we found is S-1-4-G with a total cost of 14 which is the optimal solution of the graph.

4. **(15 points)** Consider a graph of a country with each node denoting a city and the edge weights between the nodes denoting the distance between them. Let two friends A and B live in different cities in the map. On every turn, we can simultaneously move each friend to a neighboring city on the map. The amount of time needed to move from city i to city j is equal to the road distance $d(i, j)$ between the cities, but on each turn the friend that arrives first must wait until the other one arrives (and calls the friend on his/her cell phone) before the next turn can begin. We want the two friends to meet as quickly as possible.

   (i) Let $D(i, j)$ denote the straight line distance between cities i and j. Which of the following heuristic function is admissible? (a) $D(i, j)$ (b)$2 \cdot D(i, j)$ (c) $D(i, j)/2$. Briefly explain your answer.

   (ii) Does the solution cost vary depending on the heuristic for this problem? Briefly explain your answer.

   (iii) Are there completely connected maps for which no solution exists? Briefly explain your answer.

   (i) D(i, j) and D(i,j)/2 are admissible; D(i,j) is admissible because the straight-line distance is always less than or equal to the road distance between two points. So, the heuristic estimate will never exceed the actual cost. D(i,j)/2 is admissible because dividing a valid heuristic by a positive constant maintains admissibility. If D(i, j) is admissible, then D(i, j)/2 will also be less than or equal to the actual cost.

(ii) Yes, the solution cost can vary depending on the heuristic used for this problem, because the problem does not specify which heuristic should be used. If we use the admissible heuristic D (i,j) and D(i,j)/2, they guarantee that A* will find the optimal solution. An admissible heuristic influences the efficiency of the search (i.e., how quickly the algorithm finds the solution) but does not change the actual solution cost. However, if we use the inadmissible heuristic $2 \cdot D(i,j)$, it may overestimate the true cost, which could lead the algorithm to miss the optimal solution. In this case, the solution cost would vary depending on the heuristic used.

(iii) If a map is completely connected, it means that every node has a direct connection to every other node, with no isolated cities or obstacles preventing movement. Therefore, there will always be a solution.

5. (**30 points**) Consider the graph in Figure 3 with blue nodes denoting MAX nodes and red nodes denoting MIN nodes.

   (i) Solve the graph using minimax algorithm. Clearly show your calculations at each step. Identify what strategy A must choose and the corresponding payoff it will receive (value).

   (ii) Solve the graph using alpha-beta pruning. Clearly show your calculations at each step. What strategy should A choose and what is the corresponding payoff?

   (iii) Which nodes are pruned using alpha-beta pruning? Did it change A's strategy? Why or why not?

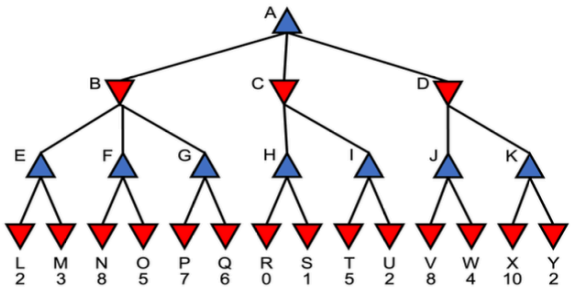


Figure 3:

(i) To solve the graph, we use the minimax algorithm, which works with a depth-first search approach. **Start** at node A (MAX), then

Explore **the B subtree first**:

Max E(L(2),M(3)) = 3 (E has two children L and M)

Max F(N(8),O(5)) = 8 (F has two children N and O)

Max G(P(7),Q(6)) = 7 (G has two children P and Q)

Then we need to backtrack the three nodes' (E, F, G) parent and get

Min B(E(3), F(8), G(7)) = 3 (B has three children E, F, and G)

Then we keep backtrack to B's parent A and pass the value 3 to max node A,

Explore the C subtree:

Max H(R(0),S(1)) = 1

Max I(T(5),U(2)) = 5

Then we backtrack to Node H and J's parent Min node C and we get

Min C(H(1),I(5)) = 1

We backtrack to node **A**. Since the value from node **C** is **1**, which is less than **3** (the current value at **A**), we don't update node **A**.

Explore the D subtree:

Max J(V(8),W(4)) = 8

Max K(X(10),Y(2)) = 10

Once we finished visiting the J subtree and K subtree, we need to update

Min D(J(8),K(10)) = 8

Backtrack to node A:

We update the value of node **A** to **8** because 8 (from node D) is greater than the previous value (3 from node B).

Therefore, the strategy for A is to choose D, and the corresponding payoff is 8.

(ii) 1. Node A (Max Node):
Alpha: -∞
Beta: +∞
2. Node B (Min Node):
Alpha: -∞
Beta: +∞
Children: E, F, G
Evaluate min(max(E), max(F), max(G))
3. Node E (Max Node):
Alpha: -∞
Beta: +∞
Children: L(2), M(3)
Evaluate max(2, 3)
v = max(-∞, 2) = 2; Alpha = 2
v = max(2, 3) = 3; Alpha = 3
Return value: 3
Back to Node B:
v = min(+∞, 3) = 3; Beta = 3
4. Node F (Max Node):
Alpha: -∞
Beta: 3
Children: N(8), O(5)
Evaluate max(8)
v = max(-∞, 8) = 8; Alpha = 8
Since v (8) ≥ Beta (3), prune the remaining children.
Pruned Node: O
Return value: 8
Back to Node B:
v = min(3, 8) = 3; Beta remains 3
5. Node G (Max Node):
Alpha: -∞
Beta: 3
Children: P(7), Q(6)
Evaluate max(7)
v = max(-∞, 7) = 7; Alpha = 7
Since v (7) ≥ Beta (3), prune the remaining children.
Pruned Node: Q
Return value: 7
Back to Node B:

v = min(3, 7) = 3; Beta remains 3
Return value: 3
Back to Node A:
v = max(-∞, 3) = 3; Alpha = 3

6. Node C (Min Node):
Alpha: 3
Beta: +∞
Children: H, I
Evaluate min(max(H), max(I))

7. Node H (Max Node):
Alpha: 3
Beta: +∞
Children: R(0), S(1)
Evaluate max(0, 1)
v = max(-∞, 0) = 0; Alpha remains 3
v = max(0, 1) = 1; Alpha remains 3
Return value: 1

8. Back to Node C:
v = min(+∞, 1) = 1; Beta = 1
Since v (1) ≤ Alpha (3), prune the remaining children.
Pruned Node: I
Return value: 1

9. Back to Node A:
v = max(3, 1) = 3; Alpha remains 3

10. Node D (Min Node):
Alpha: 3
Beta: +∞
Children: J, K
Evaluate min(max(J), max(K))
Node J (Max Node):
Alpha: 3
Beta: +∞
Children: V(8), W(4)
Evaluate max(8, 4)
v = max(-∞, 8) = 8; Alpha = 8
Return value: 8

11. Back to Node D:

v = min(+∞, 8) = 8; Beta = 8

12. Node K (Max Node):
    Alpha: 3
    Beta: 8
    Children: X(10), Y(2)
    Evaluate max(10)
    v = max(-∞, 10) = 10; Alpha = 10
    Since v (10) ≥ Beta (8), prune the remaining children.
    Pruned Node: Y
    Return value: 10
13. Back to Node D:
    v = min(8, 10) = 8; Beta remains 8
    Return value: 8
14. Back to Node A:
    v = max(3, 8) = 8; Alpha = 8
    Final Best Value: 8
    Best Strategy and Payoff:

Therefore We choose strategy D with a payoff of 8.

(iii) O, Q and the entire subtree of I (T and U), and Node Y are pruned using alpha-beta pruning. It didn't change A's strategy because alpha-beta pruning only optimizes the algorithm to make it faster; it doesn't change the final decision.