Question1

a) To simulate Pac-Man as a search problem, give following coordinate for the 5x5 grid

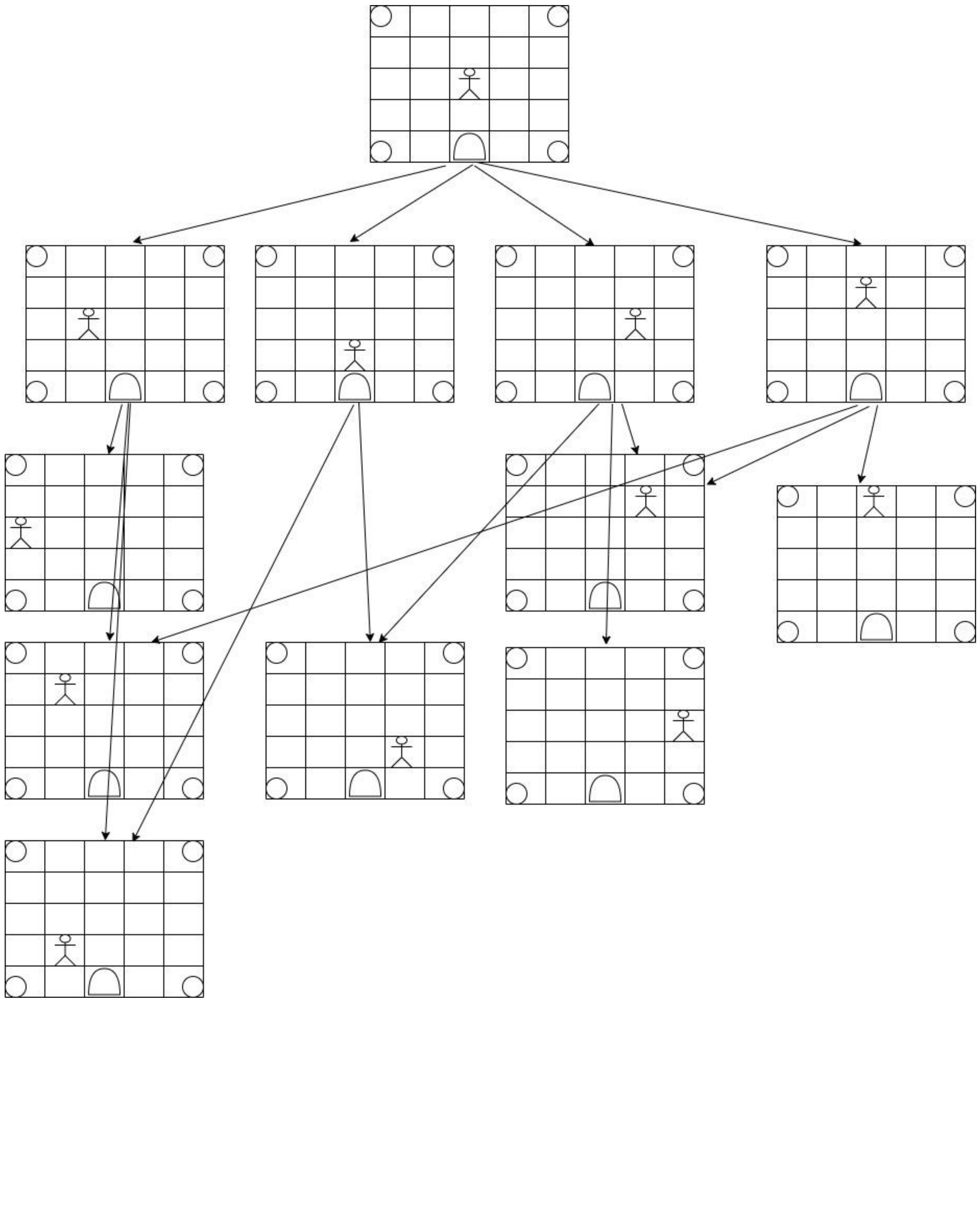
(0,0)	(0,1)	(0,2)	(0,3)	(0,4)
(1,0)	(1,1)	(1,2)	(1,3)	(1,4)
(2,0)	(2,1)	(2,2)	(2,3)	(2,4)
(3,0)	(3,1)	(3,2)	(3,3)	(3,4)
(4,0)	(4,1)	(4,2)	(4,3)	(4,4)

The search problem is Pac-Man navigate in the 5x5 grid and find a path to eat four food at (0,0), (0,4), (4,0), (4,4) without being in the same coordinate with the ghost
Use tuple to represent problem state, state can have three parts (Pac-Man position, Ghost position, food positions), e.g. ((x_p, y_p), (x_g, y_g), ((x_food1, y_food1), (x_food2, y_food2))

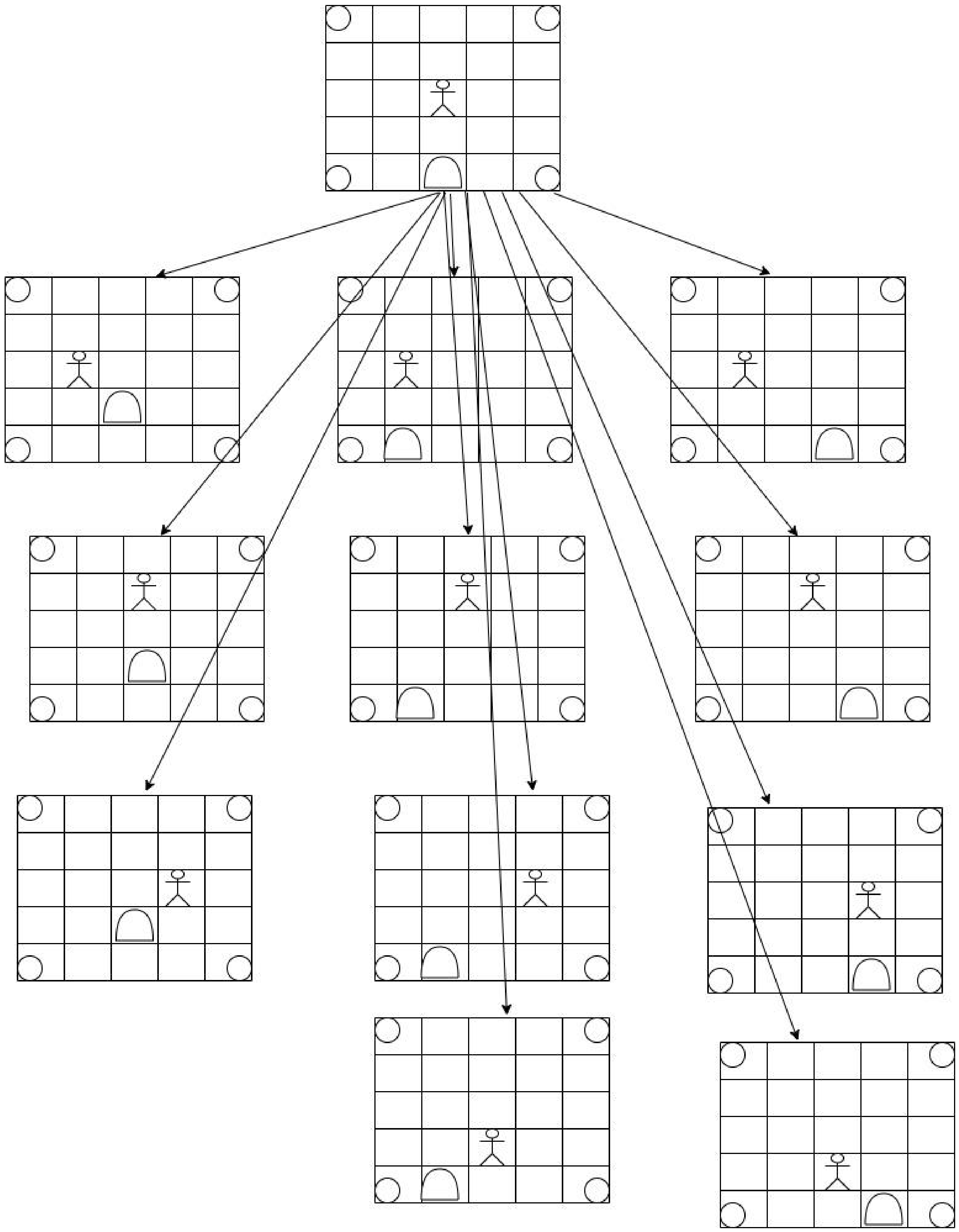
b) Pac-Man starts at position (2,2), Ghost starts at (4,2), there are four food at (0,0), (0,4), (4,0), (4,4)
Therefore, initial state can be represented as
((2,2), (4,2), ((0,0), (0,4), (4,0), (4,4)))

c) Goal state is when there is no food position, the Pac-Man position and Ghost position is not important in final state, as long as they are not the same (illegal state)
Final state can be represented as (X, X, ())

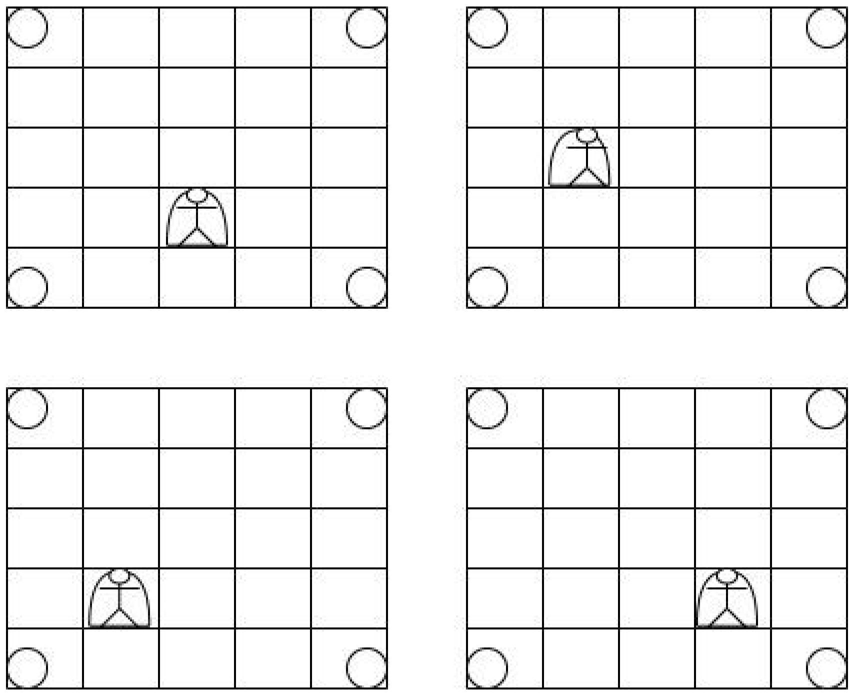
d) move up, move down, move left, move right e)



f)

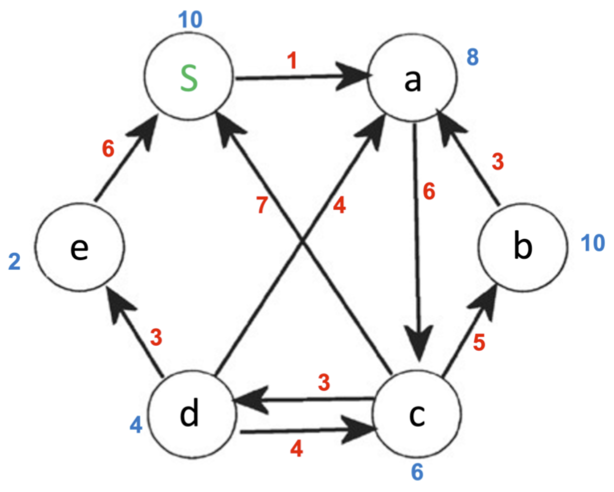


g) There are 25 illegal states, example



Question2

(a) S → "No Goal"



BFS

1. Open [S-null] closed []
2. Open [a-S] closed [S]
3. Open [c-a] closed [a, S]
4. Open [b-c, d-c] closed [c, a, S]
5. Open [d-c] closed [b, c, a, S]
6. Open [e-d] closed [d, b, c, a, S]
7. Open [] closed [e, d, b, c, a, S]

DFS

1. Open [S] closed [ ]
2. Open [a] closed [S]
3. Open [c] closed [a, S]
4. Open [b, d] closed [c, a, S]
5. Open [d] closed [b, c, a, S]
6. Open [e] closed [d, b, c, a, S]
7. Open [ ] closed [e, d, b ,c , a, S]

Iterative Deepening DFS

Limit = 0

1. Open [] closed [S]

Limit = 1

1. Open [S] closed [ ]
2. Open [a] closed [S]
3. Open [] closed [a, S]

Limit = 2

1. Open [S] closed [ ]
2. Open [a] closed [S]
3. Open [c] closed [a, S]
4. Open [ ] closed [c, a, S]

Limit = 3

1. Open [S] closed [ ]
2. Open [a] closed [S]
3. Open [c] closed [a, S]
4. Open [b, d] closed [c, a, S]
5. Open [d] closed [b, c, a, S]
6. Open [ ] closed [d, b, c, a, S]

Limit = 4

1. Open [S] closed [ ]
2. Open [a] closed [S]
3. Open [c] closed [a, S]
4. Open [b, d] closed [c, a, S]
5. Open [d] closed [b, c, a, S]
6. Open [e] closed [d, b, c, a, S]
7. Open [ ] closed [e, d, b ,c , a, S]

Uniform cost search, represent each node in queue with cost

1. Open [S] closed [ ]
2. Open [a-1] closed [S]
3. Open [c-7] closed [a, S]
4. Open [d-10, b-12] closed [c, a, S]
5. Open [b-12, e-13] closed [d, c, a, S]
6. Open [e-13] closed [b, d, c, a, S]
7. Open [ ] closed [e, b, d, c, a, S]

Hill Climbing

1. S ->
2. S -> a
3. S -> a -> c
4. S -> a -> c -> d
5. S -> a -> c -> d -> e
6. S -> a -> c -> d -> e -> STOP solution not found

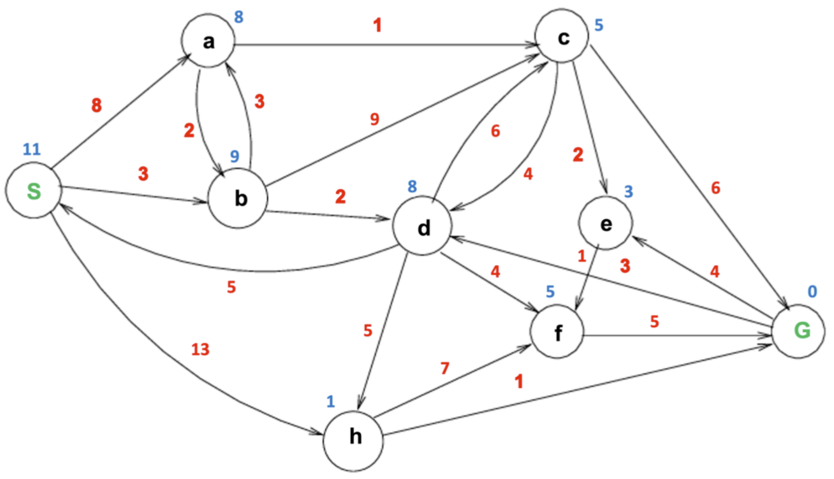
Best first search
Insert nodes in open list so that the nodes are sorted in ascending h(n)

1. Open [S] closed [ ]
2. Open [a-1] closed [S]
3. Open [c-6] closed [a, S]
4. Open [d-4, b-10] closed [c, a, S]
5. Open [e-2, b-10] closed [d, c, a, S]
6. Open [b-10] closed [e, d, c, a, S]
7. Open [ ] closed [b, e, d, c, a, S]
8. No goal

Algorithm A
f(n) = g(n) + h(n)

1. Open [S] closed [ ]
2. Open [a] closed [S] f(a) = 1+8
3. Open [c] closed [a, S] f(c) =7+6
4. Open [d, b] closed [c, a, S] f(d) = 10+4 f(b)=12+10
5. Open [e, b] closed [d, c, a, S] f(e) = 13+2 f(b) = 12+10
6. Open [b] close [e, d, c, a, S] f(b) = 12+10
7. Open [] close [b, e, d, c, a, S]
8. No goal

(b) $S \rightarrow G$



BFS

1. Open [S-null] closed [ ]
2. Open [a, b, h] closed [S]
3. Open [b, h, c] closed [a, S]
4. Open [h, c, d] closed [b, a, S]
5. Open [c, d, f, G] closed [h, b, a, S]
6. Open [d, f, G, e] closed [c, h, b, a, S]
7. Open [f, G, e] closed [d, c, h, b, a, S]
8. Open [G, e] closed [f, d, c, h, b, a, S]
9. Open [e] closed [G, f, d, c, h, b, a, S]

DFS

1. Open [S] closed [ ]
2. Open [a, b, h] closed [S]
3. Open [c, b, h] closed [a, S]
4. Open [d, e, G, b, h] closed [c, a, S]
5. Open [f, e, G, b, h] closed [d, c, a, S]
6. Open [e, G, b, h] closed [f, d, c, a, S]
7. Open [G, b, h] closed [e, f, d, c, a, S]
8. Open [b, h] closed [G, e, f, d, c, a, S]

Iterative Deepening DFS

Limit = 0

1. Open [S] closed [ ]
2. Open [ ] closed [S]

Limit = 1

1. Open [S] closed [ ]
2. Open [a, b, h] closed [S]
3. Open [b, h] closed [a, S]

4. Open [h] closed [b, a, S]
5. Open [ ] closed [h, b, a, S]

Limit = 2

1. Open [S] closed [ ]
2. Open [a, b, h] closed [S]
3. Open [c, b, h] closed [a, S]
4. Open [b, h] closed [c, a, S]
5. Open [d, h] closed [b, c, a, S]
6. Open [h] closed [d, b, c, a, S]
7. Open [f, G] closed [h d, b, c, a, S]
8. Open [G] closed [f, h d, b, c, a, S]
9. Open [ ] closed [G, f, h d, b, c, a, S]

Uniform cost search

1. Open [S] closed [ ]
2. Open [b-3, a-8, h-13] closed [S] 8.1
3. Open [d-5, a-6, c-12, h-13] closed [b, S]
4. Open [a-6, f-9, h-10, c-11] closed [d, b, S]
5. Open [c-9, f-9, h-10] closed [a, d, b, S]
6. Open [f-9, h-10, e-11, G-15] closed [c, a, d, b, S]
7. Open [h-10, e-11, G-14] closed [f, c, a, d, b, S]
8. Open [e-11, G-11] closed [h, f, c, a, d, b, S]
9. Open [e-11, G-11] closed [h, f, c, a, d, b, S]
10. Open [G-11] closed [e, h, f, c, a, d, b, S]
11. Open [ ] closed [G, e, h, f, c, a, d, b, S]

Hill Climbing

1. S
2. S -> a
3. S -> a -> c
4. S -> a -> c -> G (goal found)

Best first search

Insert nodes in open list so that the nodes are sorted in ascending h(n)

1. Open [S] closed [ ]
2. Open [h-1, a-8, b-9] closed [S]
3. Open [G-0, f-5, a-8, b-9] closed [h, S]
4. Open [f-5, a-8, b-9] closed [G, h, S]
5. Goal G found, solution path S->h->G

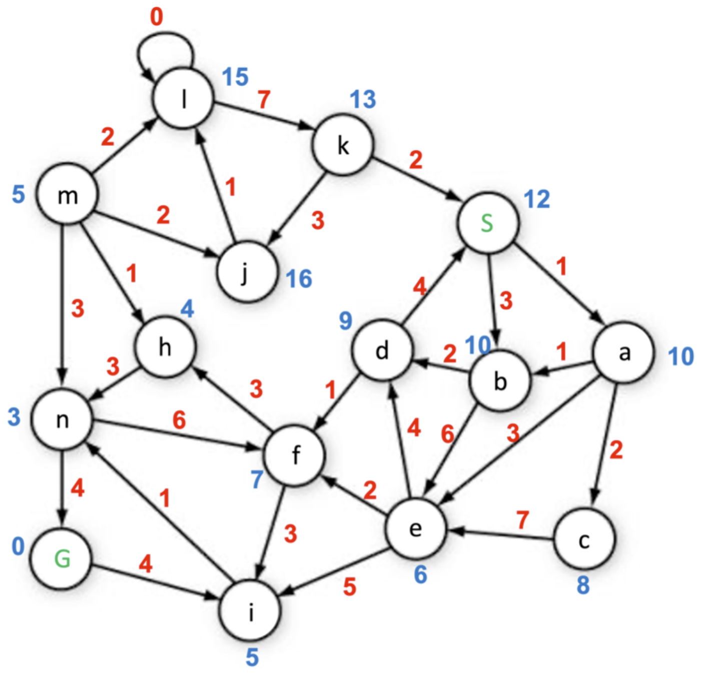
Algorithm A

f(n) = g(n) + h(n)

1. Open [S] closed [ ]
2. Open [b, h, a] closed [S] f(a)=8+8 f(b)=3+9 f(h)=13+1

3. Open [d, a, h, c] closed [b, S] f(a)=6+8 f(h)=13+1 h(c)=12+5 h(d)=5+8
4. Open [h, a, f, c] closed [d, b, S] f(a)=6+8 f(h)=10+1 h(c)=11+5 h(f)=9+5
5. Open [G, a, f, c] closed [h, d, b, S] f(a)=6+8 h(c)=11+5 h(f)=9+5 h(G)=11+0
6. Open [a, f, c] closed [G, h, d, b, S]
7. Goal G found, solution path S->b->d->h->G

(c) S → G



BSF

1. open [S] closed [ ]
2. open [a, b] closed [S]
3. open [b, c, e] closed [a, S]
4. open [c, e, d] closed [b, a, S]
5. open [e, d,] closed [c, b, a, S]
6. open [d, f, i] closed [e, c, b, a, S]
7. open [f, i] closed [d, e, c, b, a, S]
8. open [i, h] closed [f, d, e, c, b, a, S]
9. open [h, n] closed [i, f, d, e, c, b, a, S]
10. open [n] closed [h, i, f, d, e, c, b, a, S]
11. open [G] closed [n, h, i, f, d, e, c, b, a, S]
12. open [ ] closed [G, n, h, i, f, d, e, c, b, a, S]

DFS

1. Open [S] closed [ ]
2. Open [a, b] closed [S]

3. Open [c, e, b] closed [a, S]
4. Open [e, b] closed [c, a, S]
5. Open [d, f, i, b] closed [e, c, a, S]
6. Open [f, i, b] closed [d, e, c, a, S]
7. Open [h, i, b] closed [f, d, e, c, a, S]
8. Open [n, i, b] closed [h, f, d, e, c, a, S]
9. Open [G, i, b] closed [n, h, f, d, e, c, a, S]
10. Open [i, b] closed [G, n, h, f, d, e, c, a, S]

Iterative Deepening DFS
Limit = 0
1. Open [S] closed [ ]
2. Open [ ] closed [S]

Limit = 1
1. Open [S] closed [ ]
2. Open [a, b] closed [S]
3. Open [b] closed [a, S]
4. Open [ ] closed [b, a, S]

Limit = 2
1. Open [S] closed [ ]
2. Open [a, b] closed [S]
3. Open [c, e, b] closed [a, S]
4. Open [e, b] closed [c, a, S]
5. Open [b] closed [e, c, a, S]
6. Open [d] closed [b, e, c, a, S]
7. Open [ ] closed [d, b, e, c, a, S]

Limit = 3
1. Open [S] closed [ ]
2. Open [a, b] closed [S]
3. Open [c, e, b] closed [a, S]
4. Open [e, b] closed [c, a, S]
5. Open [d, f, i, b] closed [e, c, a, S]
6. Open [f, i, b] close [d, e, c, a, S]
7. Open [i, b] close [f, d, e, c, a, S]
8. Open [b] close [i, f, d, e, c, a, S]
9. Open [ ] close [b, i, f, d, e, c, a, S]

Limit = 4
1. Open [S] closed [ ]
2. Open [a, b] closed [S]
3. Open [c, e, b] closed [a, S]
4. Open [e, b] closed [c, a, S]
5. Open [d, f, i, b] closed [e, c, a, S]
6. Open [f, i, b] closed [d, e, c, a, S]
7. Open [h, i, b] closed [f, d, e, c, a, S]

8. Open [i, b] closed [h, f, d, e, c, a, S]
9. Open [n, b] closed [i, h, f, d, e, c, a, S]
10. Open [b] closed [n, i, h, f, d, e, c, a, S]
11. Open [ ] closed [b, n, i, h, f, d, e, c, a, S]

Limit = 5

1. Open [S] closed [ ]
2. Open [a, b] closed [S]
3. Open [c, e, b] closed [a, S]
4. Open [e, b] closed [c, a, S]
5. Open [d, f, i, b] closed [e, c, a, S]
6. Open [f, i, b] closed [d, e, c, a, S]
7. Open [h, i, b] closed [f, d, e, c, a, S]
8. Open [n, i, b] closed [h, f, d, e, c, a, S]
9. Open [i, b] closed [n, h, f, d, e, c, a, S]
10. Open [b] closed [i, n, h, f, d, e, c, a, S]
11. Open [ ] closed [b, i, n, h, f, d, e, c, a, S]

Limit = 6

1. Open [S] closed [ ]
2. Open [a, b] closed [S]
3. Open [c, e, b] closed [a, S]
4. Open [e, b] closed [c, a, S]
5. Open [d, f, i, b] closed [e, c, a, S]
6. Open [f, i, b] closed [d, e, c, a, S]
7. Open [h, i, b] closed [f, d, e, c, a, S]
8. Open [n, i, b] closed [h, f, d, e, c, a, S]
9. Open [G, i, b] closed [n, h, f, d, e, c, a, S]
10. Open [i, b] closed [G, n, h, f, d, e, c, a, S]

Uniform cost search

1. Open [S] closed [ ]
2. Open [a-1, b-3] closed [S]
3. Open [b-2, c-3, e-4] closed [a, S]
4. Open [c-3, d-4, e-4] closed [b, a, S]
5. Open [d-4, e-4] closed [c, b, a, S]
6. Open [e-4, f-5] closed [d, c, b, a, S]
7. Open [f-5, i-9] closed [e, d, c, b, a, S]
8. Open [h-8, i-8] closed [f, e, d, c, b, a, S]
9. Open [i-8, n-11] closed [h, f, e, d, c, b, a, S]
10. Open [n-9] closed [i, h, f, e, d, c, b, a, S]
11. Open [G-13] closed [n, i, h, f, e, d, c, b, a, S]
12. Open [ ] closed [G, n, i, h, f, e, d, c, b, a, S]

Hill Climbing

1. S
2. S -> a
3. S -> a -> e
4. S -> a -> e -> i
5. S -> a -> e -> i -> n
6. S -> a -> e -> i -> n -> G (goal found)

Best first search

1. Open [S] closed [ ]
2. Open [a-10, b-10] closed [S]
3. Open [e-6, c-8, b-10] closed [a, S]
4. Open [i-5, f-7, c-8, d-9, b-10] closed [e, a, S]
5. Open [n-3, f-7, c-8, d-9, b-10] closed [i, e, a, S]
6. Open [G-0, f-7, c-8, d-9, b-10] closed [n, i, e, a, S]
7. Open [f-7, c-8, d-9, b-10] closed [G, n, i, e, a, S]
8. Goal G found, solution path, S->a->e->i->n->G

Algorithm A

f(n) = g(n) + h(n)

1. Open [S] closed [ ]
2. Open [a, b] closed [S] f(a)=1+10 f(b)=3+10
3. Open [e, c, b] closed [a, S] f(b)=2+10 f(e)=4+6 f(c)=3+8
4. Open [c, b, f, i, d] closed [e, a, S] f(b)=2+10 f(c)=3+8 f(d)=8+9 f(f)=6+7 f(i)=9+5
5. Open [b, f, i, d] closed [c, e, a, S] f(b)=2+10 f(d)=8+9 f(f)=6+7 f(i)=9+5
6. Open [d, f, i] closed [b, c, e, a, S] f(d)=4+9 f(f)=6+7 f(i)=9+5
7. Open [f, i] closed [d, b, c, e, a, S] f(f)=5+7 f(i)=9+5
8. Open [h, i] closed [f, d, b, c, e, a, S] f(i)=8+5 f(h)=8+4
9. Open [i, n] closed [h, f, d, b, c, e, a, S] f(i)=8+5 f(n)=11+3
10. Open [n] closed [i, h, f, d, b, c, e, a, S] f(n)=9+3
11. Open [G] closed [n, i, h, f, d, b, c, e, a, S] f(G)=9+4
12. Open [ ] closed [G, n, i, h, f, d, b, c, e, a, S]
13. Goal G found

Question 3

(a) Since no heuristic values are available, informed search algorithms cannot be applied. BFS can find the sorted path in an unweighted graph but the distance graph is weighted. Uniform cost search can find the shortest path.

(b) Uniform cost search, use a priority queue sorted using the cost from Chicago (Start) to node n

    1. Open [Chicago] closed []
    2. Open [Indianapolis-182, Detroit-283, Cleveland-345], closed [Chicago]
    3. Open [Detroit-283, Cleveland-345, Columbus-358], closed [Indianapolis, Chicago]

4. Open [Cleveland-345, Columbus-358, Buffalo-Detroit-539], closed [Detroit, Indianapolis, Chicago]
5. Open [Columbus-Indianapolis-358, Pittsburgh-Cleveland-479, Buffalo–Cleveland-534], closed [Cleveland, Detroit, Indianapolis, Chicago]
6. Open [Pittsburgh-Cleveland-479, Buffalo–Cleveland-534], closed [Columbus, Cleveland, Detroit, Indianapolis, Chicago]
7. Open [Buffalo–Cleveland-534, Baltimore-Pittsburgh-726, Philadelphia-Pittsburgh-784], closed [Pittsburgh, Columbus, Cleveland, Detroit, Indianapolis, Chicago]
8. Open [Syracuse-Buffalo-684, Baltimore-Pittsburgh-726, Philadelphia-Pittsburgh-784], closed [Buffalo, Pittsburgh, Columbus, Cleveland, Detroit, Indianapolis, Chicago]
9. Open [Baltimore-Pittsburgh-726, Philadelphia-Pittsburgh-784, NY-Syracuse-938, Boston-Syracuse-996], closed [Syracuse, Buffalo, Pittsburgh, Columbus, Cleveland, Detroit, Indianapolis, Chicago]
10. Open [Philadelphia-Pittsburgh-784, NY-Syracuse-938, Boston-Syracuse-996], closed [Baltimore, Syracuse, Buffalo, Pittsburgh, Columbus, Cleveland, Detroit, Indianapolis, Chicago]
11. Open [NY-Philadelphia-881, Boston-Syracuse-996], closed [Philadelphia, Baltimore, Syracuse, Buffalo, Pittsburgh, Columbus, Cleveland, Detroit, Indianapolis, Chicago]
12. Open [NY, Boston-Syracuse-996], closed [NY, Philadelphia, Baltimore, Syracuse, Buffalo, Pittsburgh, Columbus, Cleveland, Detroit, Indianapolis, Chicago]

c) Lowest cost route: Chicago-> Cleveland-> Pittsburgh-> Philadelphia -> New York, total cost of the path is 881

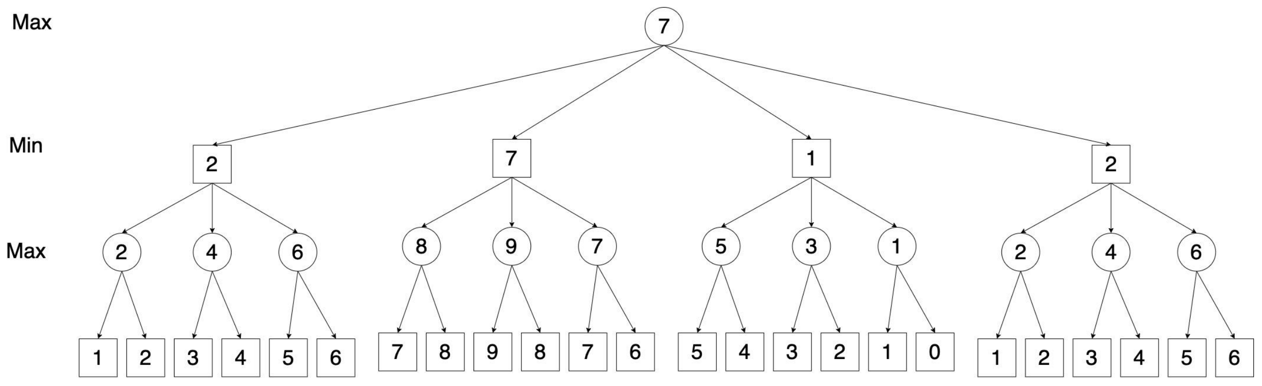
Question 4

a) The Manhattan distance heuristic for the 8-puzzle problem is an admissible heuristic if it satisfies the following property: the estimated cost from a given state to the goal state should never overestimate the actual cost.

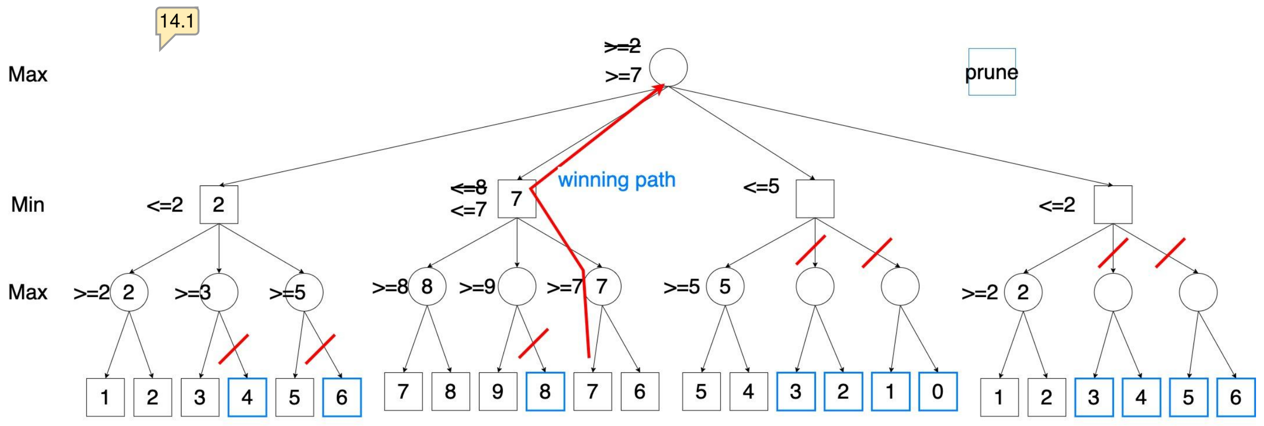
The Manhattan distance of a tile is the sum of the absolute differences between its current row and column position and its goal row and column position. Since each move of a tile can only change its position by one unit, the minimum number of moves required to reach the goal position is equal to the Manhattan distance. Since the Manhattan distance of each tile is a lower bound on the minimum number of moves required to reach the goal position, the sum of the Manhattan distances of all tiles is a lower bound on the minimum number of moves required to reach the goal state. Hence, the Manhattan distance heuristic never overestimates the actual cost, making it an admissible heuristic.

b) In Pac-Man problem, use Manhattan distance from one grid to another. An admissible heuristic can be the sum of Manhattan distance between Pac-Man's current position and all the uncollected food. This heuristic is admissible because it provides a lower bound on the minimum number of moves required to reach the uncollected food.

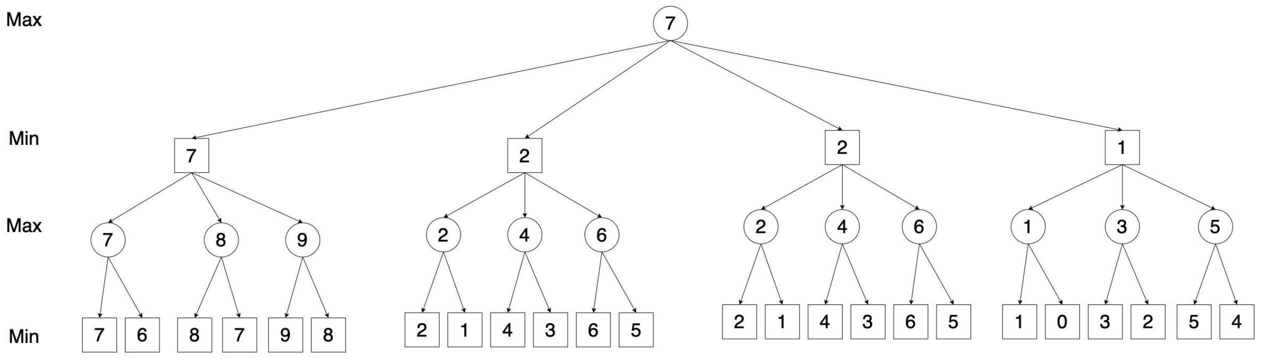
Adversarial Search
Question 1
a)
Compute the minimax game values for all the given nodes.



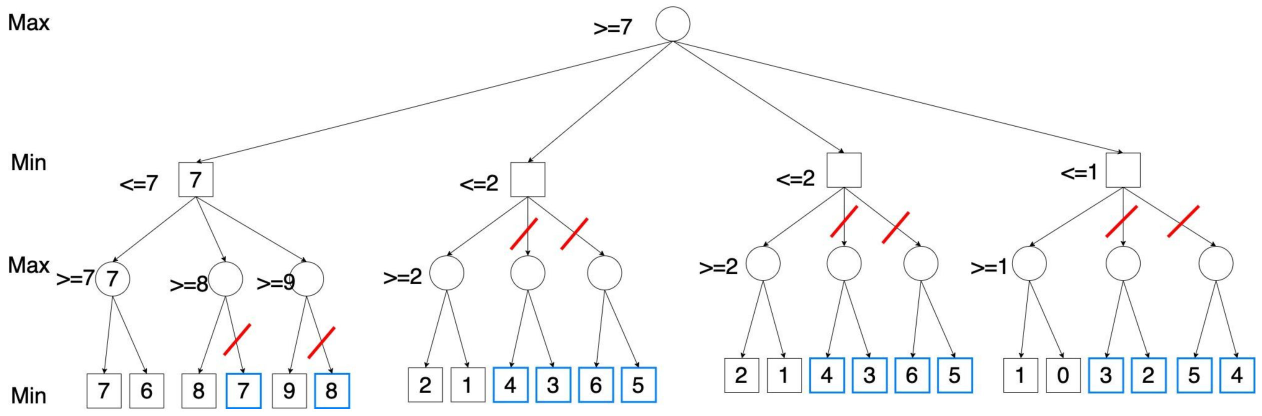
Explore the tree using Alpha-Beta pruning. Indicate all parts of the tree that are pruned, and indicate the winning path or paths. For each node, show the alpha/beta values throughout the process



11 nodes pruned out of 24 nodes
Re-draw the trees (with minimax values) by re-ordering the children of each internal node (maintaining an equivalent tree) such that the pruning is maximized.



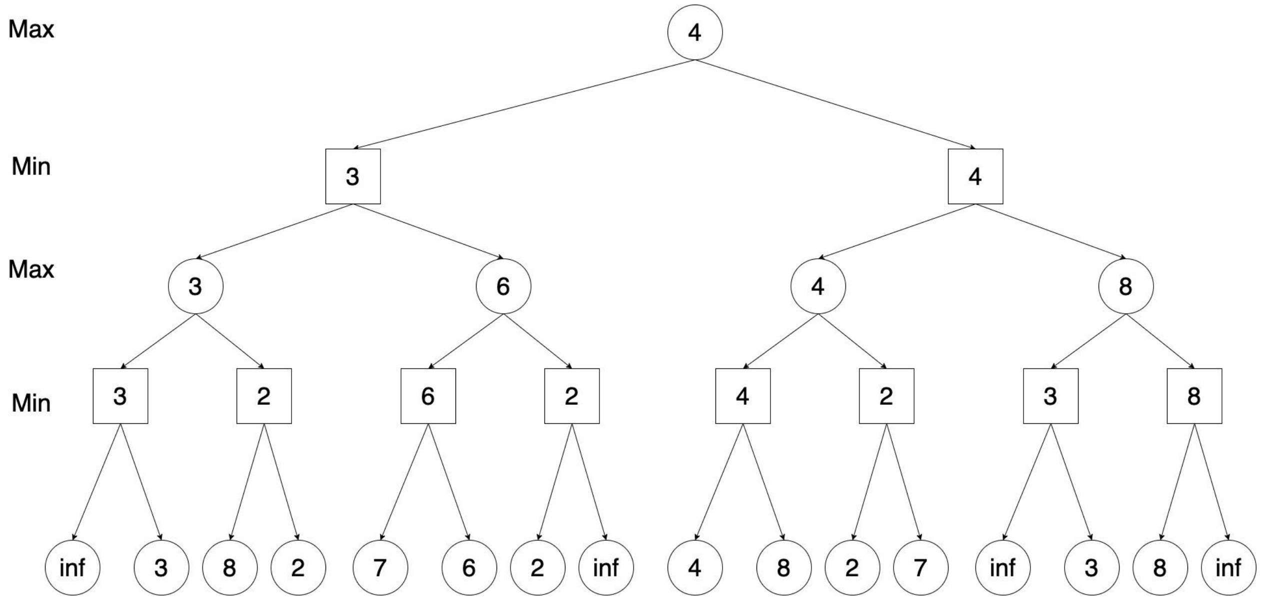
Perform alpha-beta pruning again on the re-ordered trees and indicate the difference in pruning (total number of nodes pruned compared to the previous version of the tree).



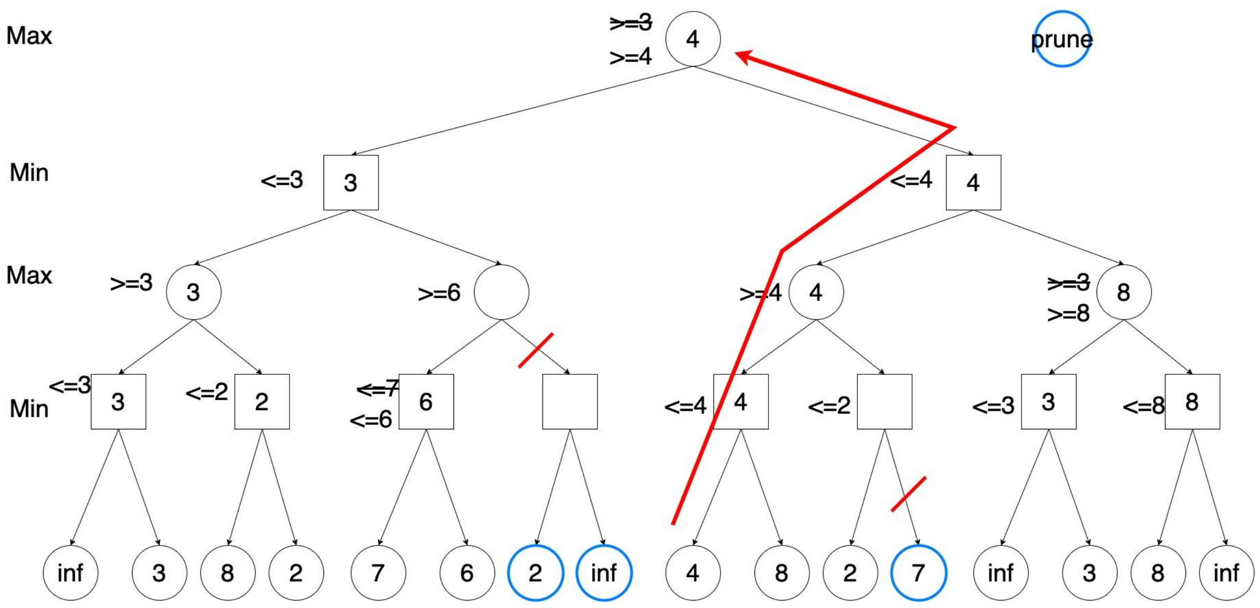
14 nodes pruned after reorder compare to 11 nodes pruned before reorder

b)

Compute the minimax game values for all the given nodes.

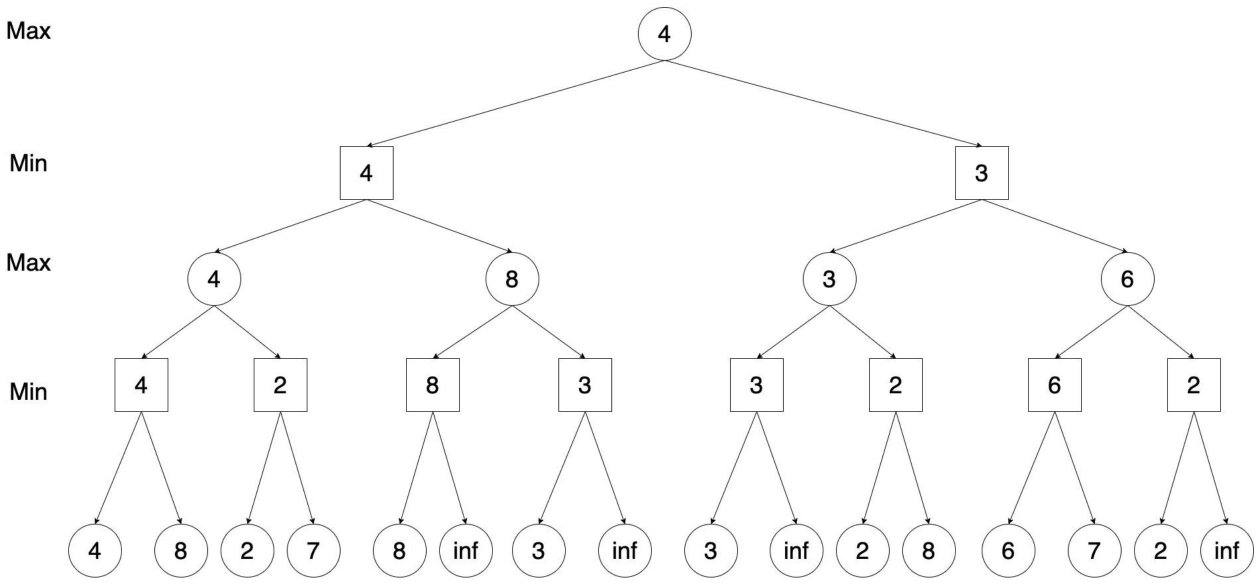


Explore the tree using Alpha-Beta pruning. Indicate all parts of the tree that are pruned, and indicate the winning path or paths. For each node, show the alpha/beta values throughout the process

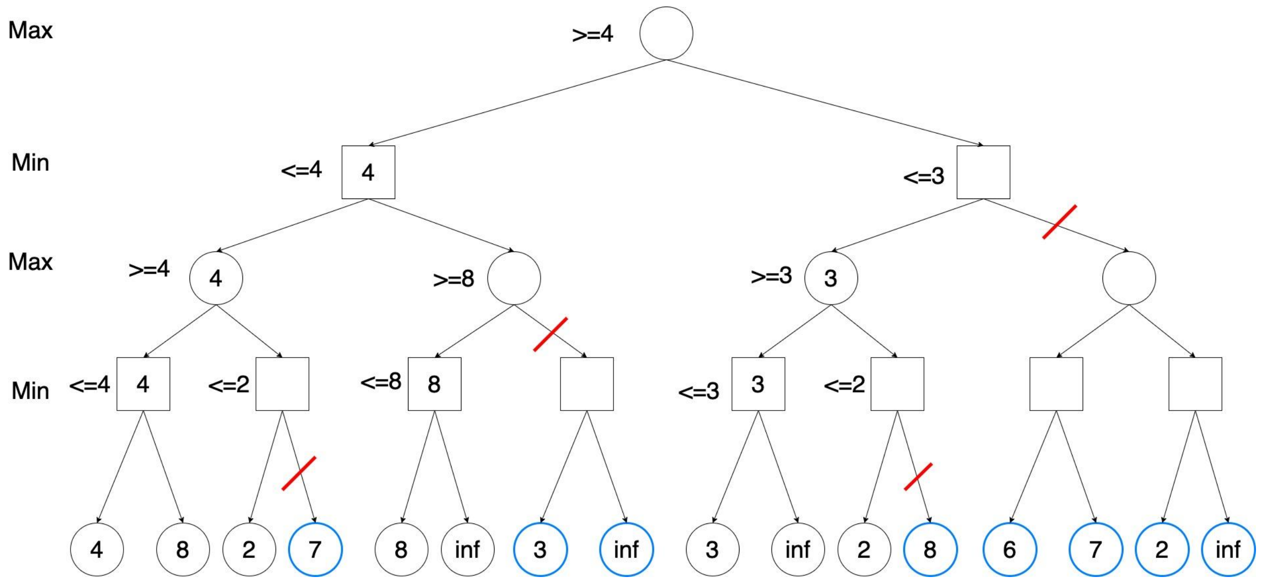


3 nodes pruned out of 16 nodes

Re-draw the trees (with minimax values) by re-ordering the children of each internal node (maintaining an equivalent tree) such that the pruning is maximized.

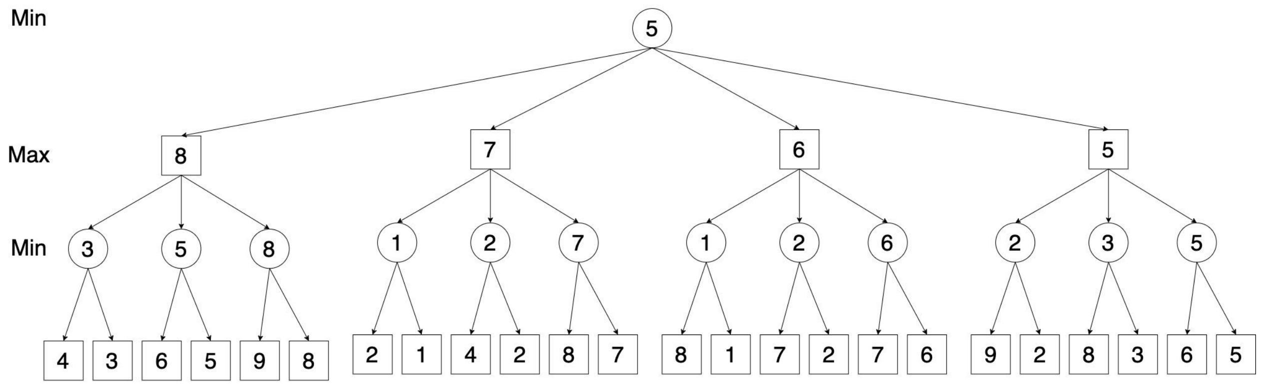


Perform alpha-beta pruning again on the re-ordered trees and indicate the difference in pruning (total number of nodes pruned compared to the previous version of the tree).

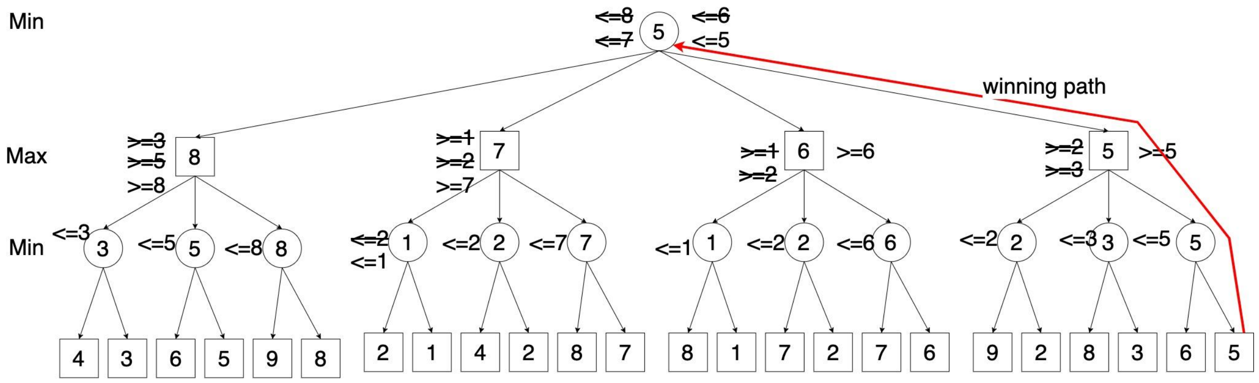


8 nodes pruned after reorder compare to 3 nodes pruned before reorder

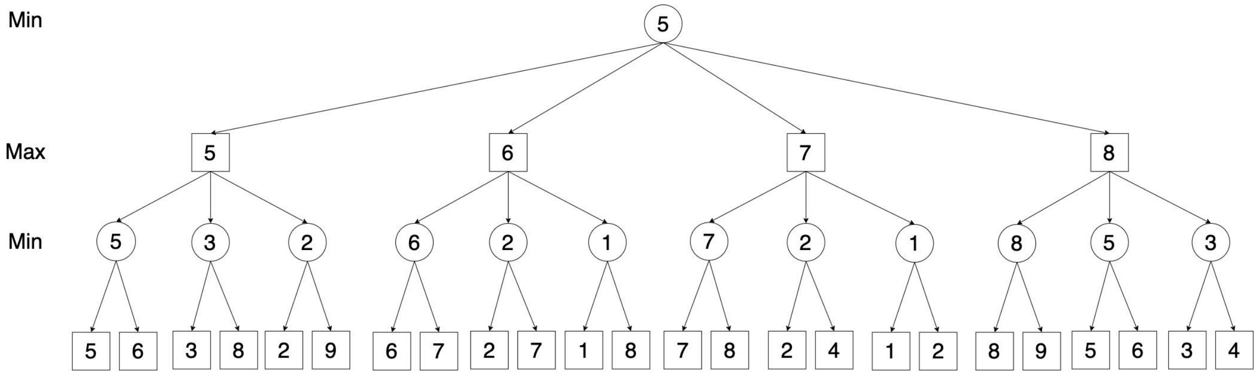
c)
Compute the minimax game values for all the given nodes.



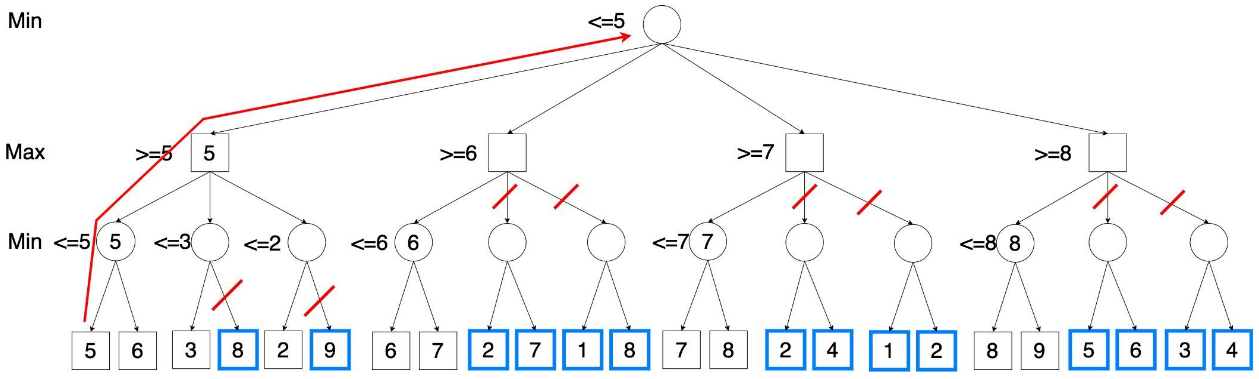
Explore the tree using Alpha-Beta pruning. Indicate all parts of the tree that are pruned, and indicate the winning path or paths. For each node, show the alpha/beta values throughout the process



Re-draw the trees (with minimax values) by re-ordering the children of each internal node (maintaining an equivalent tree) such that the pruning is maximized.



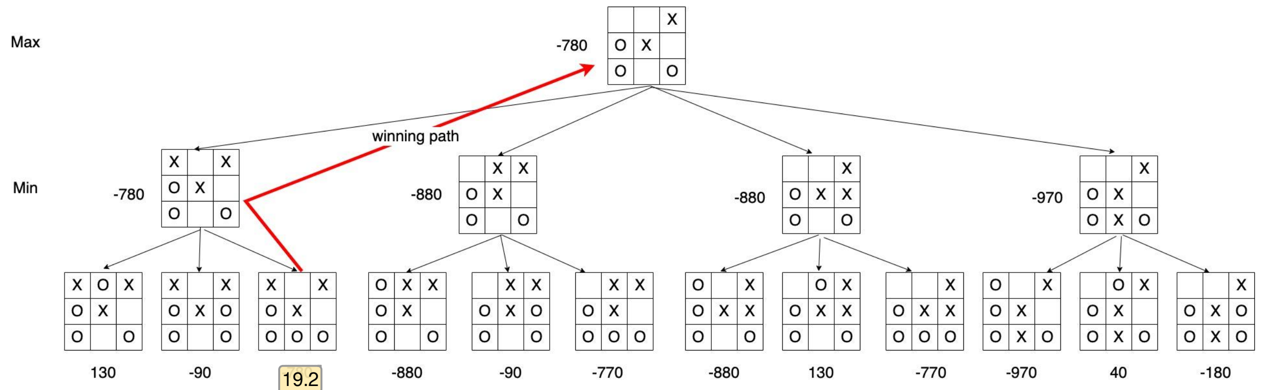
Perform alpha-beta pruning again on the re-ordered trees and indicate the difference in pruning (total number of nodes pruned compared to the previous version of the tree).



14 nodes pruned after reorder compare to no nodes pruned before reorder

19.1

Question 2



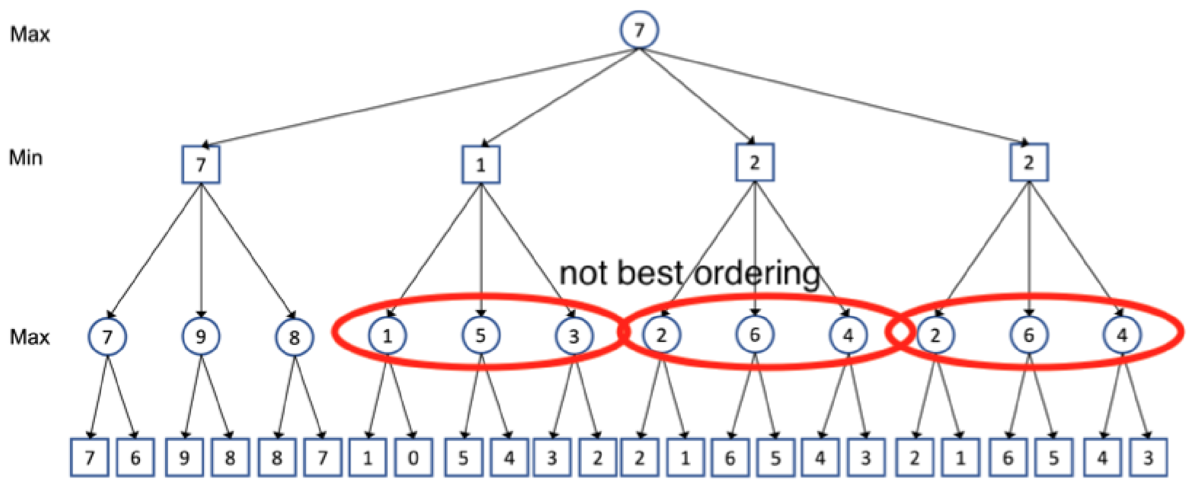
19.2

Question 3

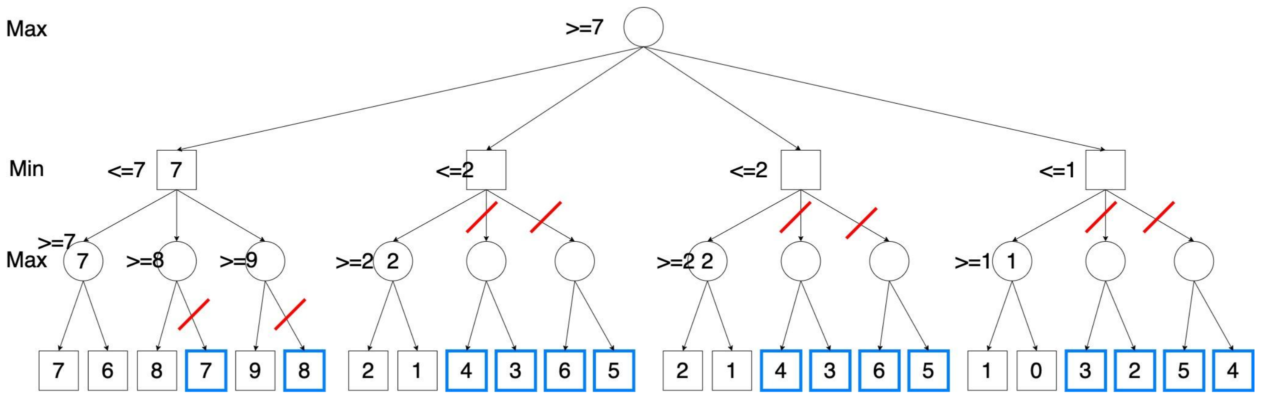
To maximize alpha-beta pruning, the tree needs to follow the best ordering, which is for the children of MIN, smallest node first; for children of MAX, largest node first.

Tree A is maximized, because Tree B is not best ordering

Tree B:



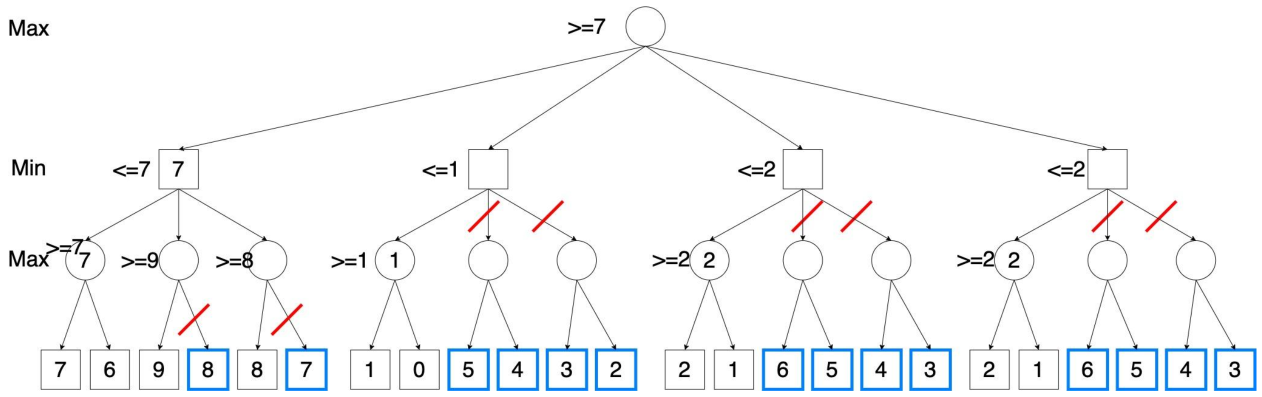
Run alpha-beta pruning on Tree A



14 nodes pruned out of 24 node in total

19.3

Run alpha-beta pruning on Tree B



14 nodes pruned out of 24 node in total

From alpha-beta pruning result, Tree A and Tree B pruned the same number nodes

Index of comments

4.1	Illegal states should include all the food configurations.
8.1	e's cost should be 9, and correct path would be s-b-d-a-c-e-f-h-n-G
14.1	no of pruned nodes should consider parent nodes as well which we will not explore.
19.1	Value is miscalculated here.
19.2	-770
19.3	if both has same pruning level, then we can't say which one is best. None wins here.