CIS 571 - Assignment 1

Ali Hassani

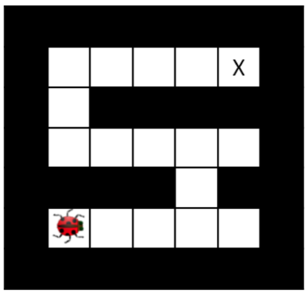
October 3, 2021

Question 1

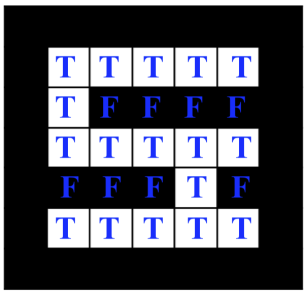
1.1 Lonely bug

1.1.1 Environment

The state could be represented by a $M \times N$ boolean matrix (5×5 for this particular case), specifying whether each cell is accessible or not (walls, blocks, etc). Therefore it could be represented either with an $M \times N$ boolean matrix.



View



T	T	T	T	T
T	F	F	F	F
T	T	T	T	T
F	F	F	T	F
T	T	T	T	T

State grid

Figure 1: Lonely bug map representation

1.1.2 States

The insect's location and the target location could also be stored as two tuples separately (specifying the indices). They could be in any of the MN cells in this particular game, which means there are MN possible combinations for the bug state. Given that the number of walls within the map are exactly P, the number of bug states would come down to $MN - P$.

1.1.3 Actions

As for actions, the bug can take actions **stay**, **left**, **right**, **up** or **down**, only if they are legal (don't end up in a wall). It was not specified whether the bug can face a specific direction, therefore storing that in the state space can be excluded.

1.1.4 Successor

We can easily check the player (insect) cell and neighboring cells to figure out which ones are legal to be moved into, and with an action we can simply update the player tuple.

1.1.5 Goal state

Whenever the player coordinates match the goal coordinates, the player's won. Goal state does not move, and there is only a single goal, therefore storing it only requires the coordinates, but it can exist only in a single place, therefore there are no additional states because of it. Therefore the exact minimum number of possible world states comes down to:

$$len(S) = M \cdot N - P$$

and the goal state is simply when the bug is at the X mark.

1.2 Jumping bug

Similar to the previous question, there are $M \cdot N - P$ possible states. Again, if P is unknown, we can set it to 0 and assume the bug and the goal can be anywhere for the sake of simplicity.

Now the actions are: Face **right**, **left**, **up**, or **down**. The reason why we don't need a **stay** action is that if it takes the action to face the direction it is already facing, that would be equivalent to staying as-is.

So in each $M \cdot N - P$ cell in which the bug can exist, it can be facing either of the 4 directions, therefore the state space is expanded to

$$len(S) = 4 \cdot (M \cdot N - P)$$

or simply $4MN$ if we don't eliminate the walls.

We can further eliminate states if we assume that moving in any direction results in moving as far as possible, therefore the bug could never exist at intermediate cells.

1.3 Swarm movement

Each insect can exist in one of $M \cdot N - P$ (or $M \cdot N$) free cells. However, no two insects can exist in the same cell, therefore there are $K - 1$ states (except its own) that can be eliminated: $M \cdot N - (P + K - 1)$ (or $M \cdot N - K + 1$):

$$len(S) = K \cdot (M \cdot N - (P + K - 1))$$

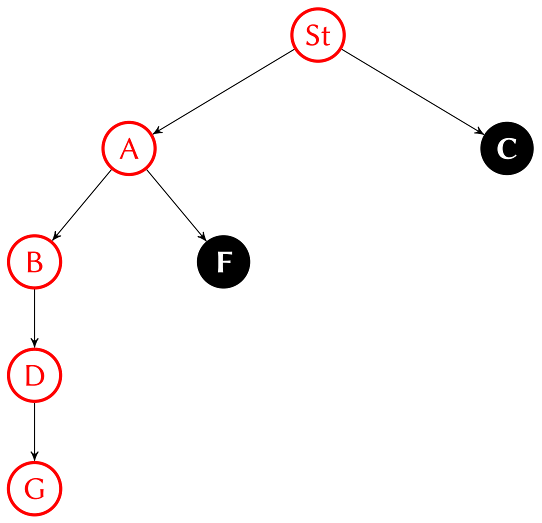
The actions are the same, but would be further constrained so that no bug can jump into a cell in which another exists. Similarly, the goal state changes to when each bug is at its own target state.

Question 2

2.1 DFS

Once `Start` is visited, its children `A` and `C` are added to the fringe. The only remaining nodes are the two, and since their depth is the same, the tie is broken alphabetically and `A` is visited. Its children `B` and `F` are added to the fringe, ties is broken in favor of `B`. `B` only leads to `D`, which is now the deepest node in the fringe. Once `D` is visited, its only child `Goal` is added to the fringe, and being the deepest node is visited. Therefore `Goal` is reached in the following order:

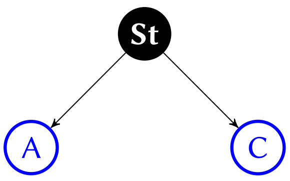
`Start` ⟶ `A` ⟶ `B` ⟶ `D` ⟶ `Goal`



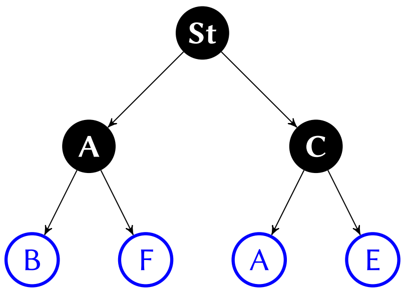
2.2 BFS

Once `Start` is visited, its children `A` and `C` are added to the fringe. The only remaining nodes are the two, and since their depth is the same, the tie is broken alphabetically and `A` is visited. Its children `B` and `F` are added to the fringe. Now `C` is visited and its children `A` and `E` are added to the fringe. Now the fringe contains `A`, `B`, `E` and `F`. After visiting them, the fringe contains `B` ⟶ `D`, `F` ⟶ `D`, `A` ⟶ `B`, `A` ⟶ `F`, `E` ⟶ `G`, and `E` ⟶ `F`. Eventually we visit `E` ⟶ `G` which is at the goal state, therefore the path is returned:

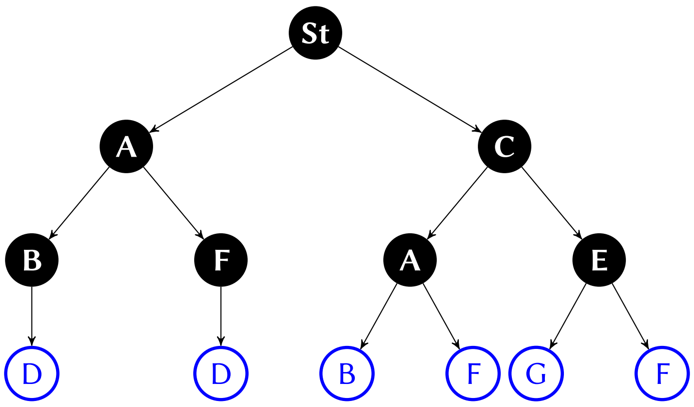
`Start` ⟶ `C` ⟶ `E` ⟶ `Goal`



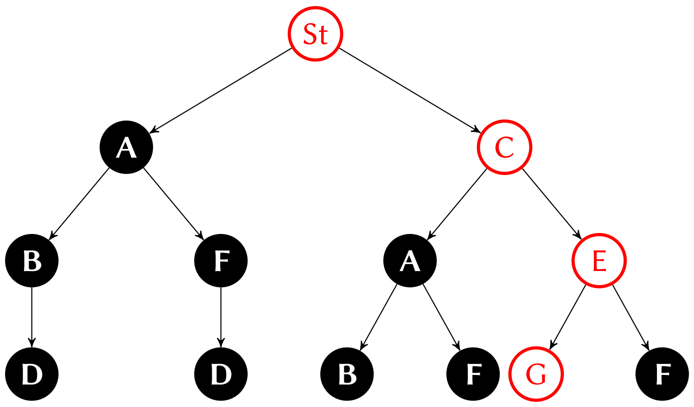
Step 1



Step 2



Step 3



Step 4

2.3 UCS

As presented in figures 2 and 3,once `Start` is expanded, `A` and `C` are pushed to the fringe with costs 3 and 2 respectively. Therefore, `C` is expanded next, which leads to both `A` and `E` at cost 2, and therefore with the cost from `Start` to `C`, it costs 4 to get to each. Now among `A`, `C` ⟶ `A` and `C` ⟶ `E`, which cost 3, 4, and 4 respectively, `A` is chosen. This continues until the following node is visited at cost 10:

`Start` ⟶ `C` ⟶ `E` ⟶ `Goal`

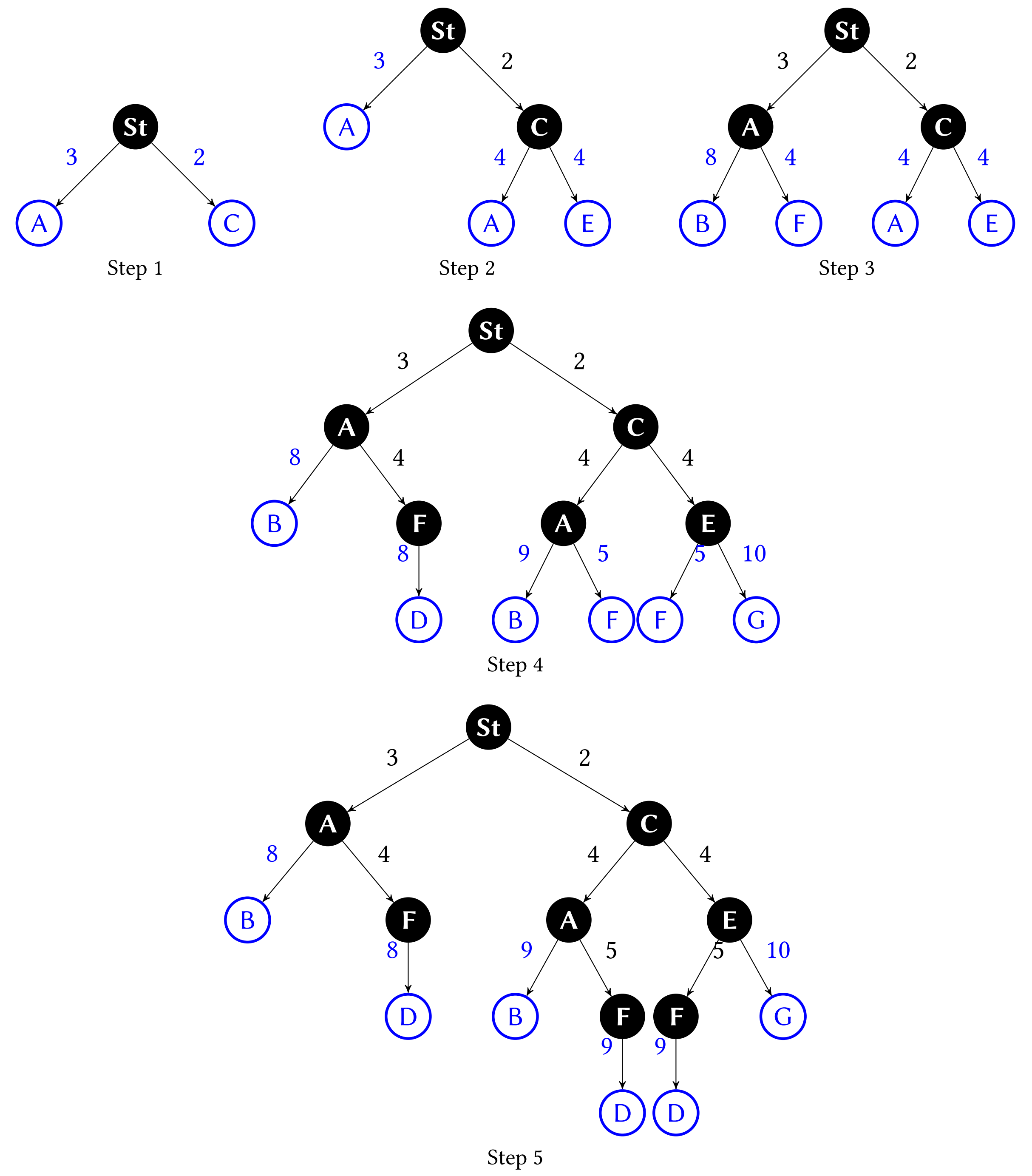
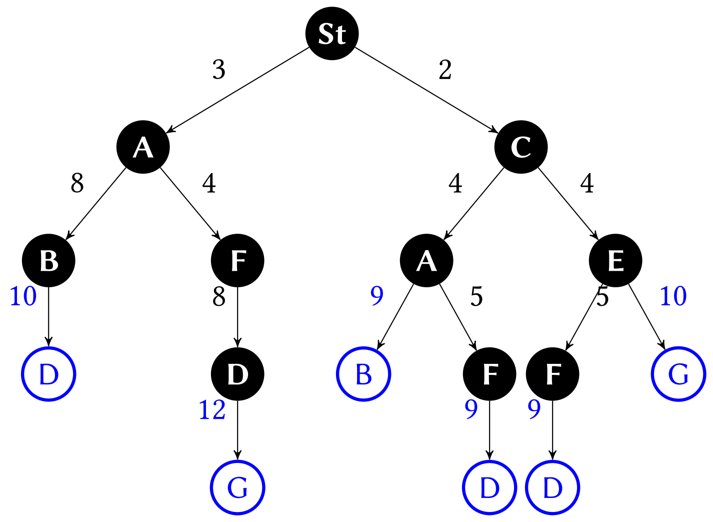
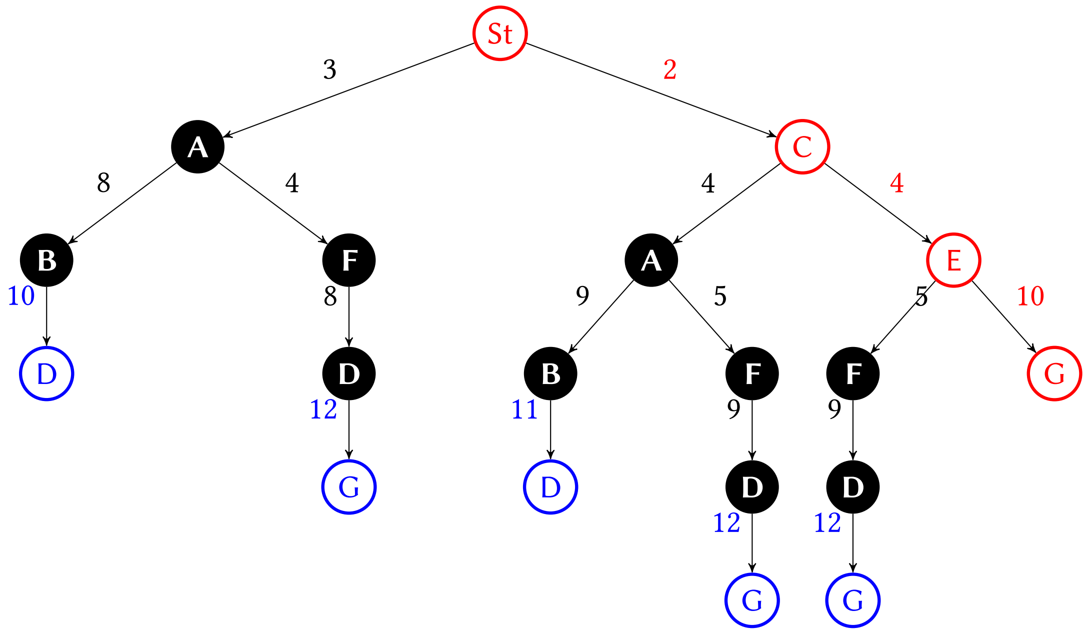


Figure 2: Uniform Cost Search



Step 6



Step 7

Figure 3: Uniform Cost Search

2.4 Greedy search

Greedy (best first) search is not optimal, and this is an example why. By expanding `Start`, it has to choose between `A` and `C`, and `A` has the smaller heuristic value. Afterwards, `A`'s two children `B` and `F` are appended to the fringe with heuristic values 4 and 6, both of which are less than `C`'s. Therefore, `B` is explored which leads to only `D` with a heuristic value less than the other two nodes in the fringe, and `D` only leads to `Goal`. The path is:

`Start` ⟶ `A` ⟶ `B` ⟶ `D` ⟶ `Goal`

which is what DFS also returned. It is not an optimal path neither w.r.t. step count or the cost function: 4 steps, cost 13.

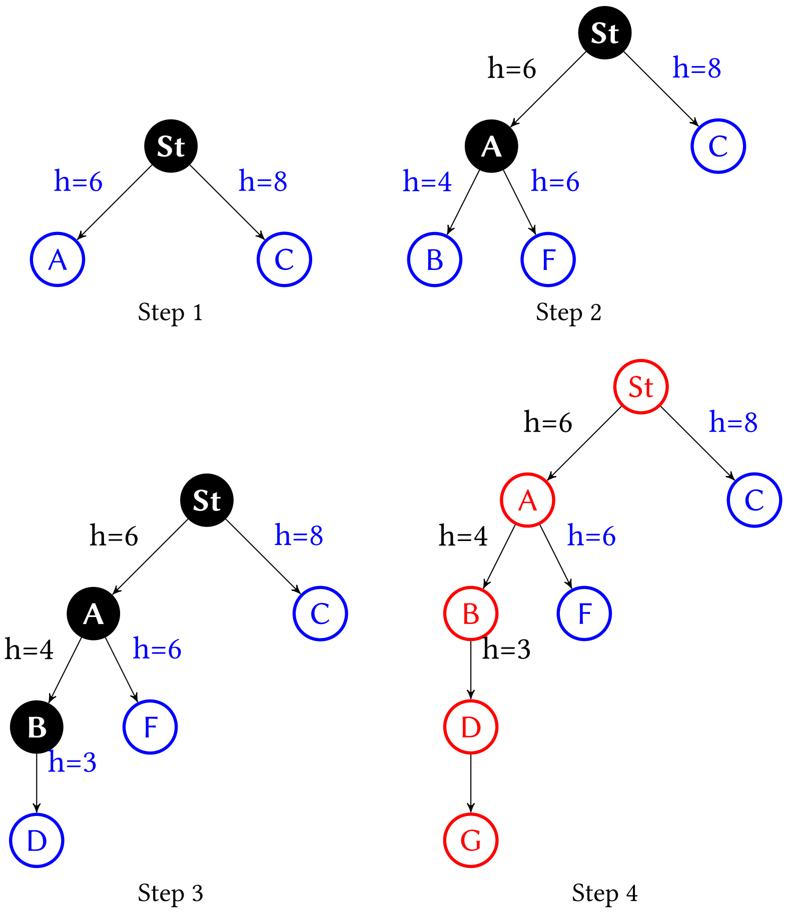


Figure 4: Greedy Search

2.5 A*

`Start` $\longrightarrow$ `C` $\longrightarrow$ `E` $\longrightarrow$ `Goal`

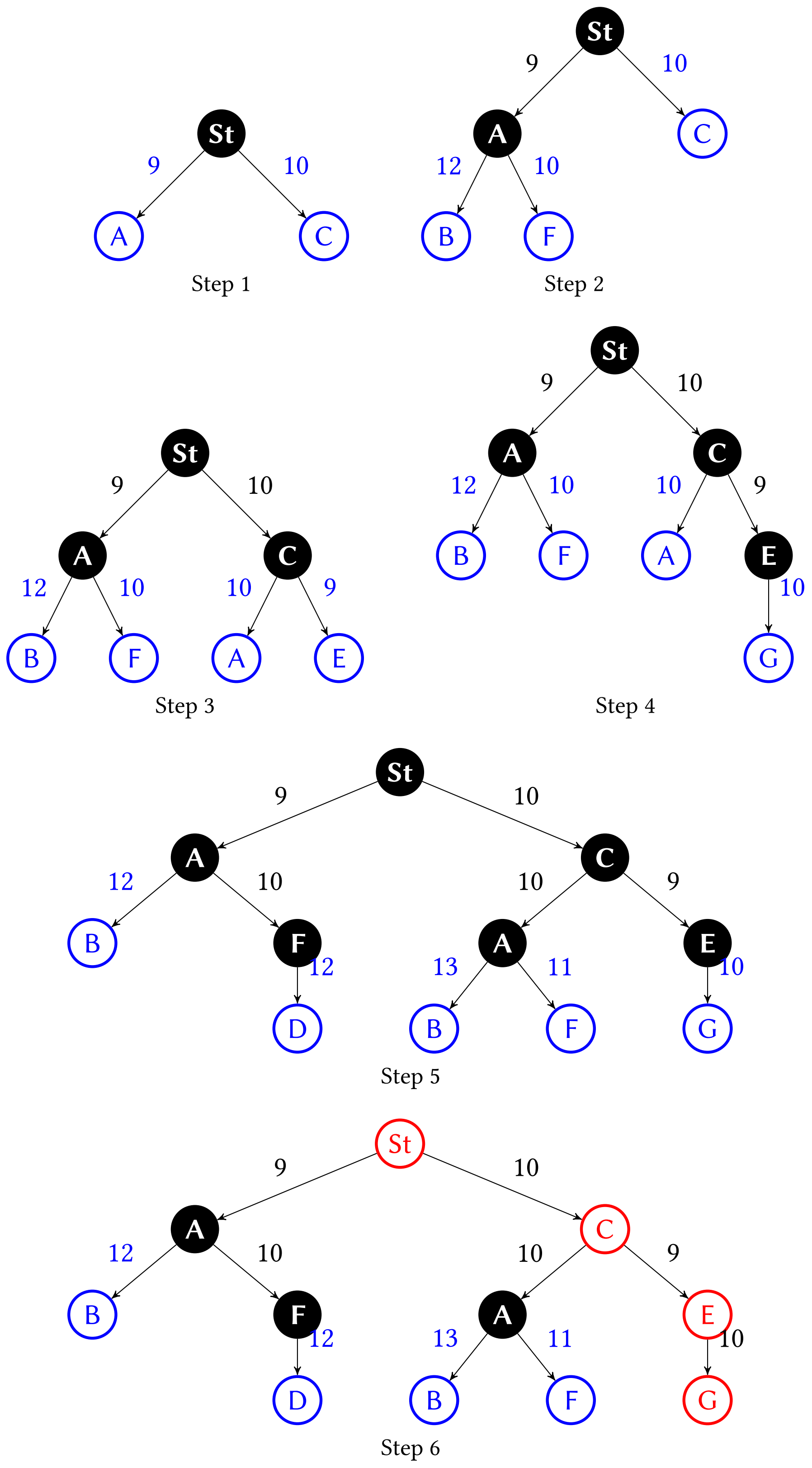


Figure 5: A* Search

2.6 Admissibility

It is admissible. For every node g, $h(g) \leq h^*(g)$:

Node	$h(x)$	Optimal cost
`Start`	9	10
`A`	6	8
`B`	4	5
`C`	8	8
`D`	3	3
`E`	5	6
`F`	6	7
`G`	0	0

2.7 Consistency

It is NOT consistent. It is even noticeable in the A* tree. `C` $\longrightarrow$ `E` has a total cost of $cost(C, E) = 2$, and $h(E) = 5$, while $h(C) = 8$. Therefore the consistency constraint DOES NOT HOLD TRUE here: $h(C) = 8 > 7 = 5 + 2 = h(E) + cost(C, E)$

This can be made consistent simply by fixing the heuristic function at node `E` by increasing it to 6.

Question 3

3.1 Lookahead search path

The algorithm will return the path `Start` $\longrightarrow$ `B` $\longrightarrow$ `Goal` which is NOT the optimal path w.r.t. the cost function. When the cost function is a constant function of $g(n) = 1$ (which would make the strategy essentially a BFS), this would be the optimal solution.

fringe-closed={S, A, B}

fringe={(A, 5), (B, 4)}

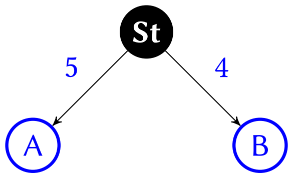


Figure 6: Step 1

fringe-closed={S, A, B, D, G}

fringe={(A, 5), (D, 8), (G, 11)}

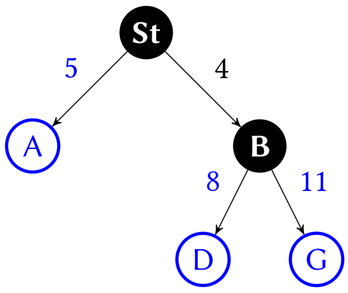


Figure 7: Step 2

fringe-closed={S, A, B, D, G, C}

fringe={(D, 8), (G, 11), (C, 6)}

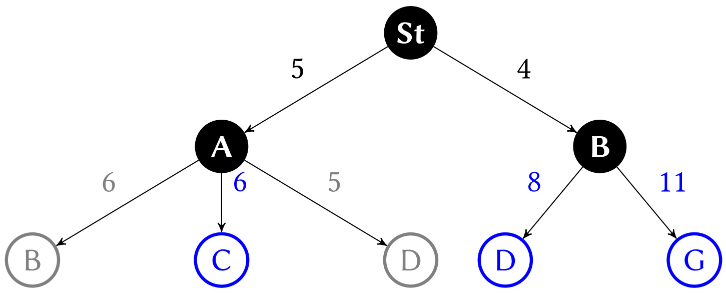


Figure 8: Step 3

fringe-closed={S, A, B, D, G, C}

fringe={(D, 8), (G, 11)}

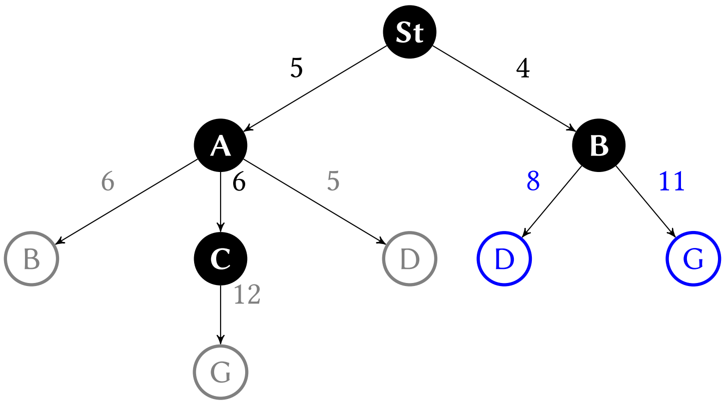


Figure 9: Step 4

fringe-closed={S, A, B, D, G, C}

fringe={(G, 11)}

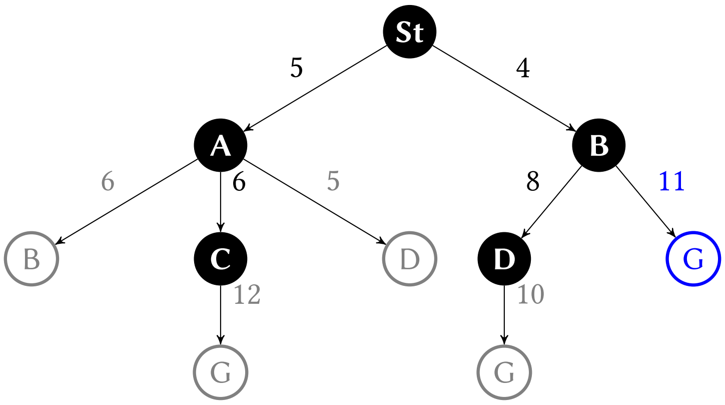


Figure 10: Step 5

fringe-closed={S, A, B, D, G, C}

fringe={}

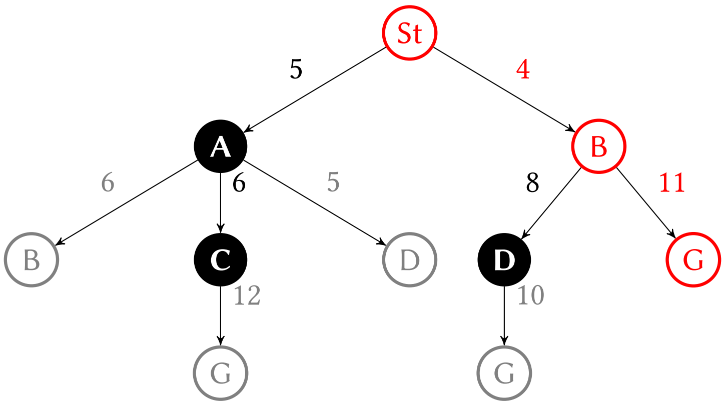


Figure 11: Solution

3.2 Select all that are true

a. The EXPAND function can be called at most once for each state.

   (i) This is true **IF my understanding of the question is correct.** **The notation of** $STATE[node]$ **and** $node$ **is confusing since they are used interchangeably. Assuming they are EQUAL, this statement holds true.**

   (ii) The initial state is appended both to the `fringe-closed` set and the fringe itself. From there, every node that is added to the fringe is also appended to `fringe-closed`, which means any node EXPLORED and not just EXPANDED is in `fringe-closed`. As a result, any one node or state is EXPANDED at most once.

   (iii) **It is worth noting that this also holds true for** `GRAPH-SEARCH`.

b. The algorithm is complete.
   The modification has no effect on completeness.

c. The algorithm will return an optimal solution.

   (i) This is NOT true.

   (ii) The example provided is the perfect example. The optimal solution is `Start` $\longrightarrow$ `A` $\longrightarrow$ `D` $\longrightarrow$ `Goal` with a cost of 7.

   (iii) Had the state `Start` $\longrightarrow$ `A` $\longrightarrow$ `D` with value 5 been expanded (which does happen in the normal implementation), this solution would have been found.