–Tutorial -4

Question 1: Prove each of the following :statements, or give a counterexample

a. Breadth-first search is a special case of uniform-cost .search

When all step costs are equal, g(n) = depth(n), so uniform-.cost search reproduces breadth-first search

.b. Depth-first search is a special case of best-first tree search

Breadth-first search is best-first search with *f(n) = depth(n);* depth-first search is best-first search with *f(n) = −depth(n);* uniform-cost search is best-first search with *).f(n) = g(n*

.c. Uniform-cost search is a special case of A∗ search

.Uniform-cost search is A∗ search with h(n) = 0

Question 2:

1 . Describe a heuristic function that will make A* search behave exactly like uniform-cost search .for a given cost function

- We know:
  - A* expands nodes with minimal *f(n) = g(n) + h(n).*
  - UCS expands nodes with minimal path cost *g(n).*
- Then:
  - For *h(n) = 0* or constant A* Expands nodes with minimal path cost *g(n)* i.e. A* behave like UCS.

Question 2:
2 . Describe a heuristic function that will make . greedy search behave like breadth-first search

- We know:
  - Greedy search expand nodes with minimal h(n).
  - Then, If *h(n) = d(n)* where *d(n)* is the depth of node n:
    - Nodes will be expanded according to increasing values of depth i.e. the shallowest before the deepest i.e. like BFS.

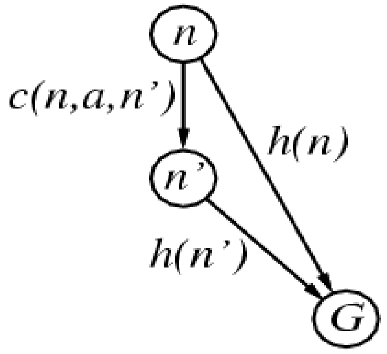
Question 2:
3. Prove that Breadth first search and uniform . cost search are special cases of best first search

- We know:
  - Best first search expands nodes with minimal f(n).
  - Using Q2.2 Greedy search behave like BFS when h(n) = d(n) f(n) = d(n) BFS is a special case of Greedy search.
  - Greedy search is a best first search.
  - Then:
  - BFS is a special case of Best First search.
-
- We know:
  - Best first search expands nodes with minimal f(n).
  - Using Q2.1 A* behave like UCS when h(n) = 0 f(n) = g(n) UCS is a special case of A*.
  - A* is a best first search.
  - Then:
  - UCS is a special case of Best First search.

Question3: Give a proof for the following consequence of heuristic consistency:

If h(n) is consistent, then the values of f(n) along .any path are nondecreasing

h(n) ≤ c(n,a,n') + h(n')



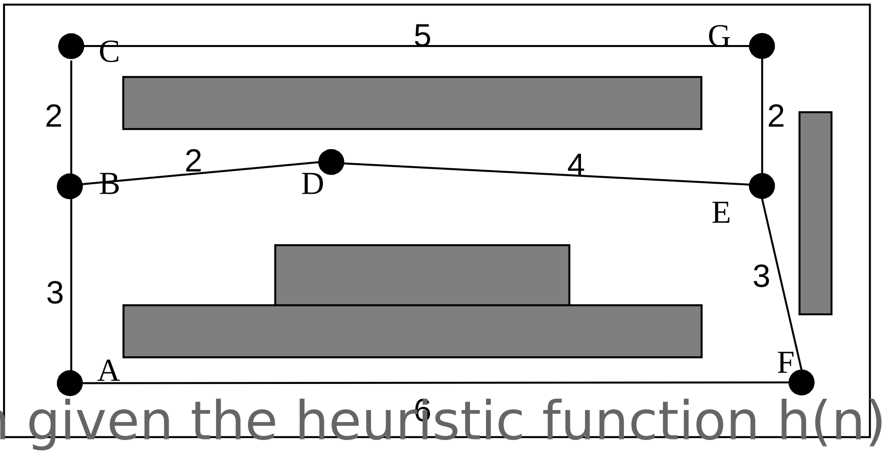
- *We know in A*:*
  - *F(n) = g(n) + h(n)*
  - *If n' successor of n g(n') = g(n) + c(n,a,n')*
  - *if h(n) consistent h(n) < c(n,a,n') + h(n').*

it can be: h(n) ≤ c(n,a,n') + h(n')

- *Then:*
  - *F(n') = g(n') + h(n')*
  - *F(n') = g(n) + c(n,a,n') + h(n')*
  - *F(n') = g(n) + c(n,a,n') + h(n') >= g(n) + h(n)*
  - *F(n') >= F(n) i.e. nodes expanding is non-decreasing order of [f(n)].*

Problem1:

Given an environment in which a robot has to move from location A to location G as shown in the figure below. We want to design a search method to allow the robot finding its route to G. The robot can only move in straight lines .and it cannot pass through the obstacles



Apply A* search given the heuristic function h(n) for which values for each node are defined in the table below. Show the Explored list, the Frontier, the search tree, the solution path and the solution cost. Expand nodes in alphabetical order when you have more than one candidate for .expansion

Node n	**A**	**B**	**C**	**D**	**E**	**F**	**G**
)h(n	8	6	6	5	2	5	0

Frontier:

A8

B9 F11

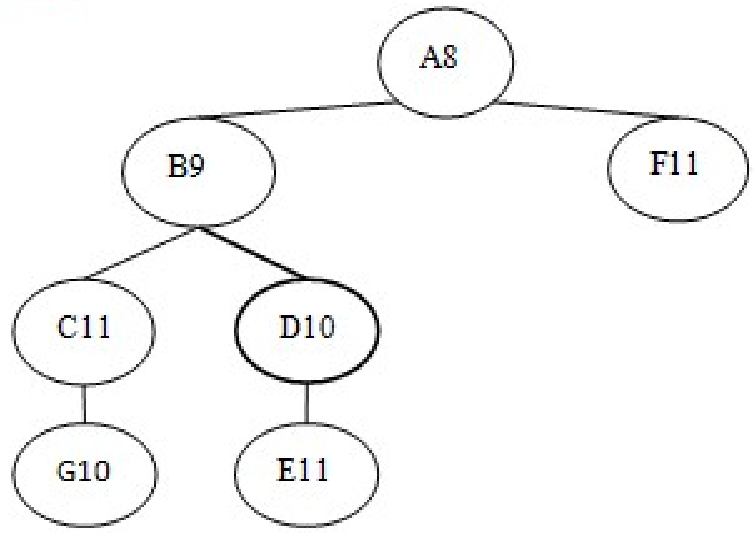
D10 C11 F11

C11 E11 F11

G10 E11 F11

E11 F11

Tree:



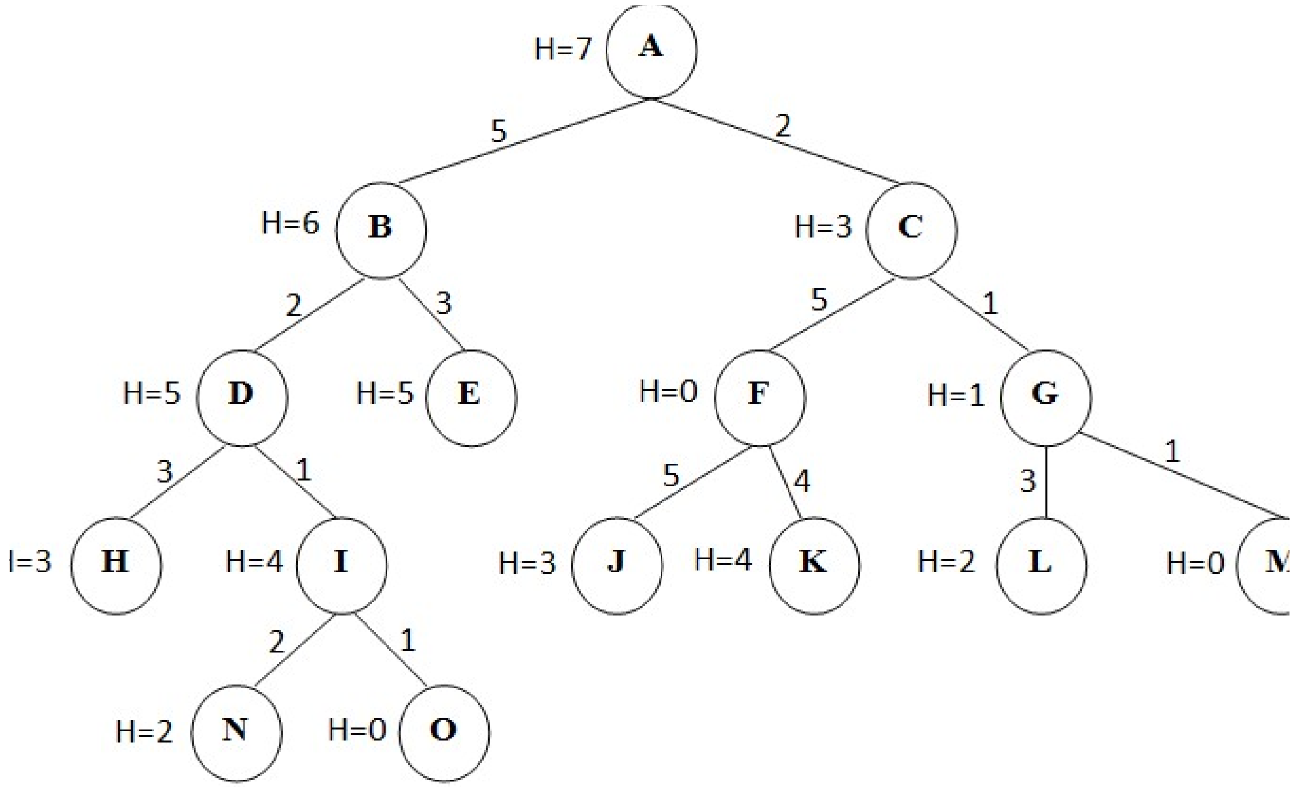
Explored list: A B D C

Cost: 10

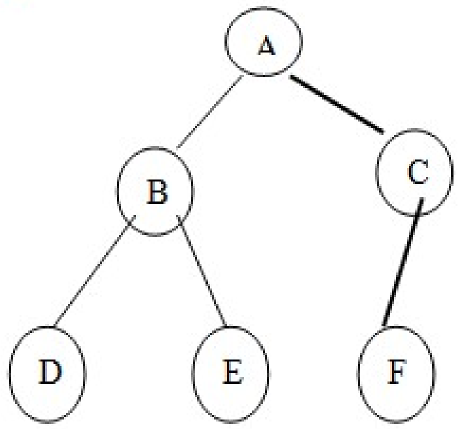
Solution path: A-B-C-G

Given the following state space, starting from A, find the goal(s) using BFS, DFS, UCS, Greedy, and A* search strategies. Note that numbers on the arcs represent step cost and numbers to the left of each node represent heuristic value for that node. Ties break alphabetically.

- What are the goals?
- For each strategy (BFS, DFS, UCS), show the first goal that will be found, and the cost (without using frontier and explored).
- For each strategy (BFS, DFS, UCS, Greedy, and A*), give the search tree, the frontier, the explored, the order in which the nodes are visited, the goal found, and the cost.



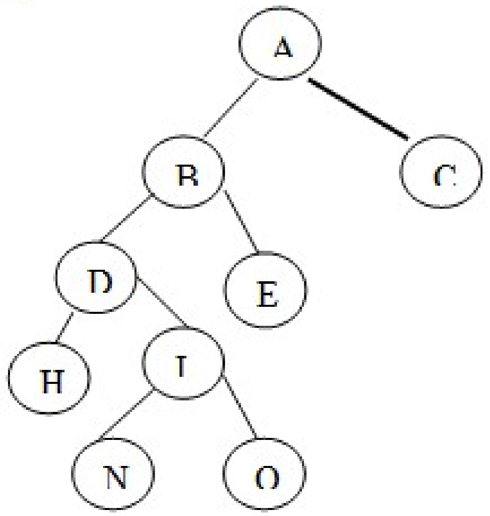
	Order	Goal	Cost
BFS	A, B, C, D , E, F	F	7
DFS	A, B, D, H, I, N, O	O	9
UCS	A, C, G, M	M	4
Greedy	**A C F**	**F**	**7**
A*	**A C G M**	**M**	**4**

BFS **Frontier**: A ~~B~~ C ~~C~~ D E	**Tree:** 
Explored list: A B C	**Cost:** 7
Solution path: A-C-F	

Frontier:

A
~~B~~ C
~~D~~ E-C
~~H~~ ~~I~~ E C
~~N~~ O E C
~~O~~ E C

Tree:



Explored list: A B D H I N

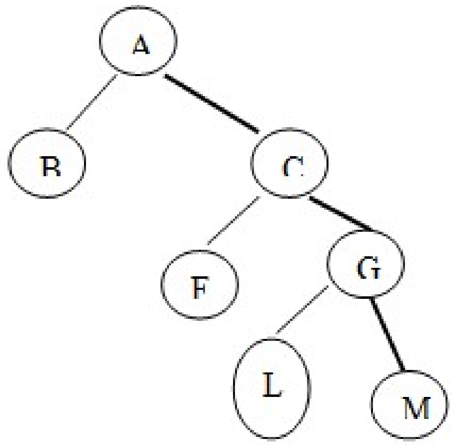
Cost: 9

Solution path: A-B-D-I-O

Frontier:

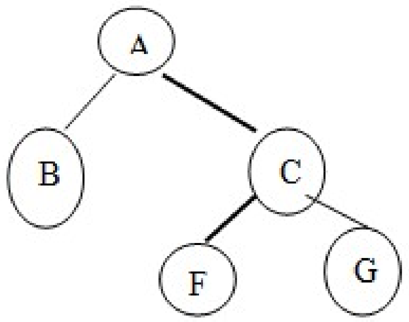
A
~~C(2)~~ B(5)
~~G(3)~~ B(5) F(7)
~~M(4)~~ B(5) L(6) F(7)

Tree:



Explored list: A C G	**Cost:** 4
Solution path: A-C-G-M	

Greedy

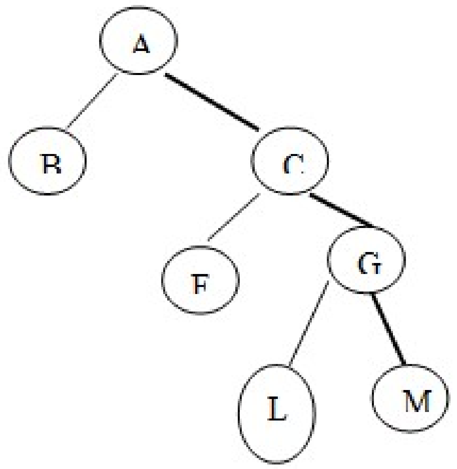
Frontier:	Tree:
A ~~C(3)~~ B(6) ~~F(0)~~ G(1) B(6)	
Explored list: A C	**Cost: 7**
Solution path: A-C-F	

A*

Frontier:

A
~~C(5)~~ B(11)
~~G(4)~~ F(7) B(11)
~~M(4)~~ F(7) L(8) B(11)

Tree:

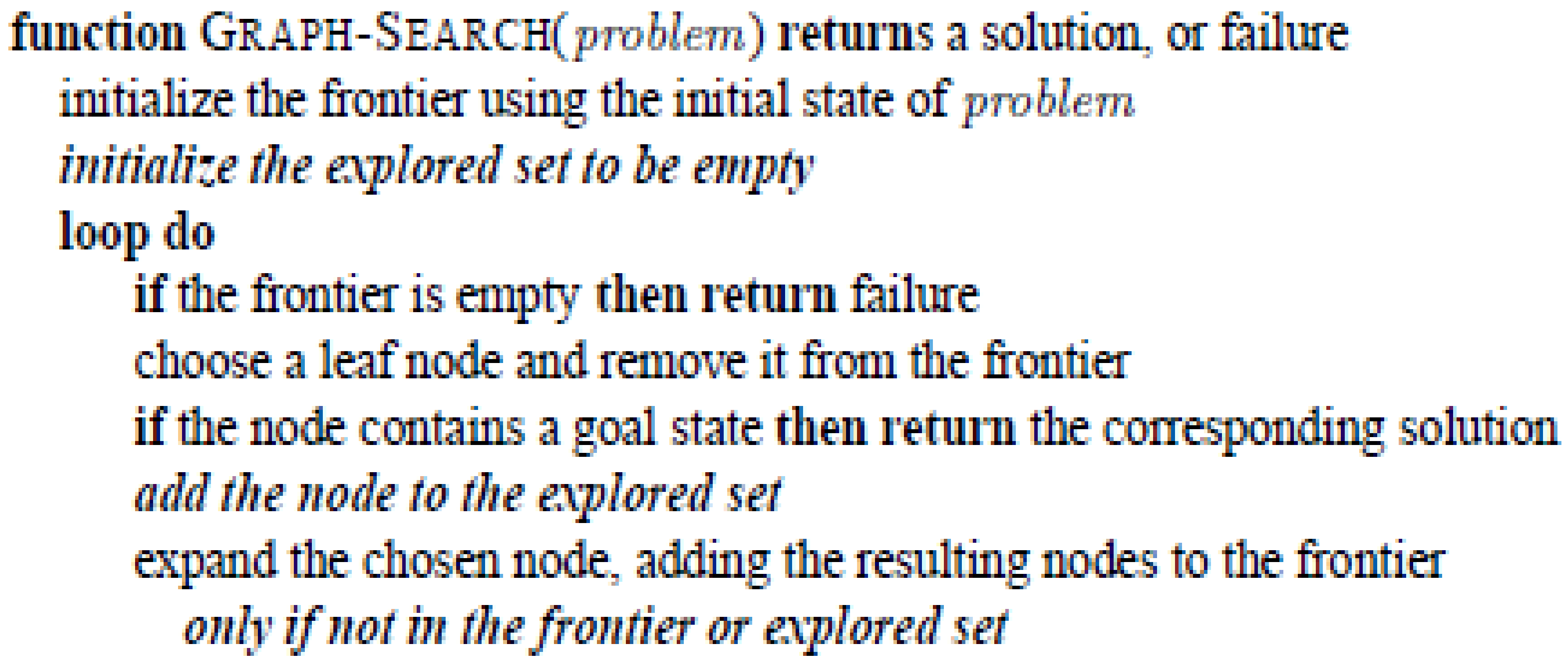


Explored list: A C G	**Cost:** 4
Solution path: A-C-G-M	

UCS algorithm on a graph

```
function UNIFORM-COST-SEARCH(problem) returns a solution, or failure
    node ← a node with STATE = problem.INITIAL-STATE, PATH-COST = 0
    frontier ← a priority queue ordered by PATH-COST, with node as the only element
    explored ← an empty set
    loop do
        if EMPTY?(frontier) then return failure
        node ← POP(frontier)   /* chooses the lowest-cost node in frontier */
        if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
        add node.STATE to explored
        for each action in problem.ACTIONS(node.STATE) do
            child ← CHILD-NODE(problem, node, action)
            if child.STATE is not in explored or frontier then
                frontier ← INSERT(child, frontier)
            else if child.STATE is in frontier with higher PATH-COST then
                replace that frontier node with child
```

Depth-first search on a graph



```
function GRAPH-SEARCH(problem) returns a solution, or failure
  initialize the frontier using the initial state of problem
  initialize the explored set to be empty
  loop do
     if the frontier is empty then return failure
     choose a leaf node and remove it from the frontier
     if the node contains a goal state then return the corresponding solution
     add the node to the explored set
     expand the chosen node, adding the resulting nodes to the frontier
       only if not in the frontier or explored set
```

Formal description of Best-First Search algorithm

```
Function Best-First Graph-Search(problem,frontier,f) returns a solution or a failure
// f: evaluation function
children← an empty set;
explored← an empty set;
frontier← Insert (Make-Node(Initial-state[problem],NULL,NULL,d,c),frontier)

Loop do
    If  Empty?(frontier) then return failure
    node ← POP(frontier)
    If Goal-Test[ problem] applied to State[node] succeeds  then return Solution(node)
    add State[node] to explored
    children←Expand (node,problem)
   for each child in children
        If state [child] is not in explored or frontier then
            frontier←insert(State [child], frontier)  // sort frontier in ascending order of  f-values
        else if  child[state] is in frontier with higher f-values  then
            replace that frontier node with child
End Loop
```