Best-First Search Algorithm

Best-first search is a systematic control strategy, combining ***the strengths of breadth-first and depth-first search into one algorithm***. The main difference between best-first search and the brute-force search techniques is that we make use of an evaluation or heuristic function to order the **SearchNode** objects on the queue. In this way, we choose the SearchNode that appears to be best, before any others, regardless of their position in the tree or graph.

It tries to expand the node that is closest to the goal, on the: grounds that this is likely to lead to a solution quickly. Thus, it evaluates nodes by using just the heuristic function: f (n) = h(n).

Main Program of Best-Search Algorithm

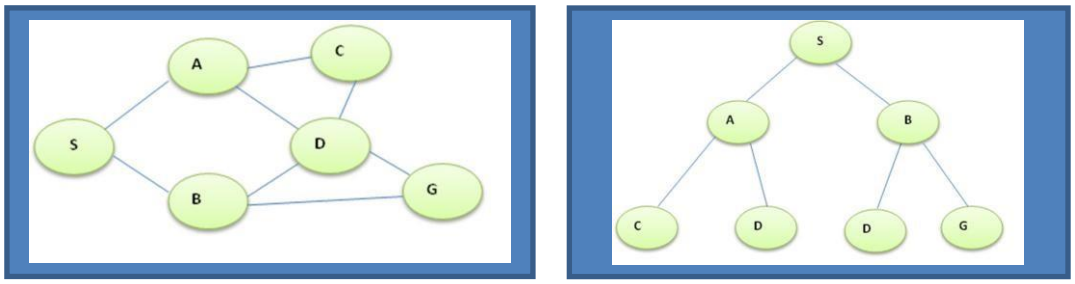
```
Function best_first_search
    open := [Start]
    closed := [ ]
    while open ≠ [ ]
       remove leftmost state from open, call it X
       if X is a goal then return SUCCESS
       else visit(X)
       put X on closed
        sort open by heuristic merit (best leftmost)
    return FAIL
End function
```

```
Subroutine visit(X)
    generate children of X
    for each child of X
      case  % 3 cases
            the child is not on open or closed:    child has never been seen before
                assign the child a heuristic value
                add the child to open

            the child is already on open:          child is waiting
                if the child was reached by a shorter path
                then give the state on open the shorter path

            the child is already on closed:        child been visited
                if the child was reached by a shorter path
                then remove the child from closed
                     add the child to open
End Subroutine
```

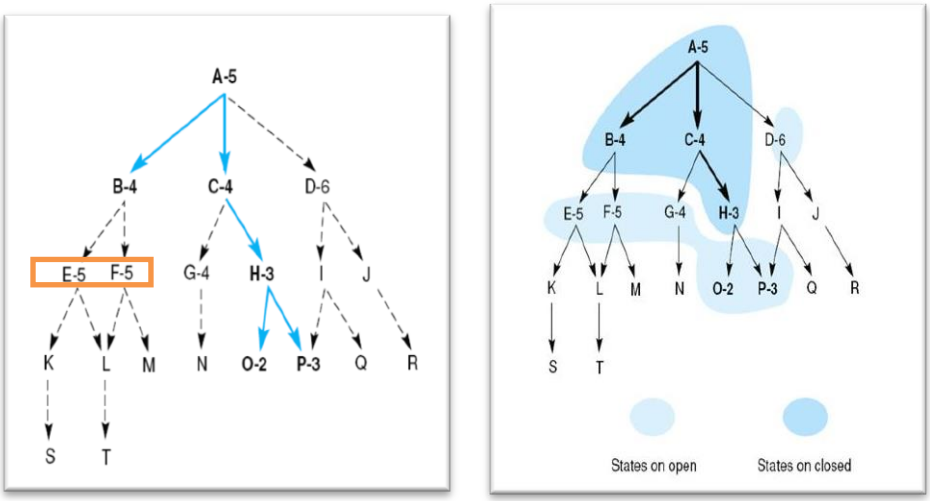
Example



Heuristic Values : A=2; B=3; C=1; D=4; S=10; G=0

Item	Open	Closed
1	open = [s10]	closed=[ ]
2	open =[2AS,3BS]	closed=[s10]
3	open=[1CAS,3BS,4DAS]	closed=[A2,S10]
4	open=[3BS,4DAS]	closed =[1CAS, A2,S10]
5	open=[0GBS,4DAS]	closed=[3BS,1CAS, A2,S10]
6	open=[0GBS,4DAS]	Goal

Example 2:



1. open=[A5]; closed=[ ]
2. Visit A5; open=[B4,C4,D6]; closed=[A5] Ordered by heuristic values
3. Visit B4; open=[C4,E5,F5,D6]; closed=[B4,A5]
4. visit C4; open=[H3,G4,E5,F5,D6]; closed=[C4,B4,A5]
5. visit H3; open=[O2,P3,G4,E5,F5,D6]; closed=[H3,C4,B4,A5]

……

Notes on Breadth, depth and Best Search Strategies

- A heuristic search – use heuristic (evaluation) function to select the best state to explore.
- Can be implemented with a priority queue, best one lines up in the front of the queue
- Breadth-first implemented with a queue, First In First Out (FIFO)
- Depth-first implemented with a stack, Last In First Out (LIFO)

Breadth-first	FIFO
Depth-first	LIFO
Best-first	???

Characteristics of Best First Search

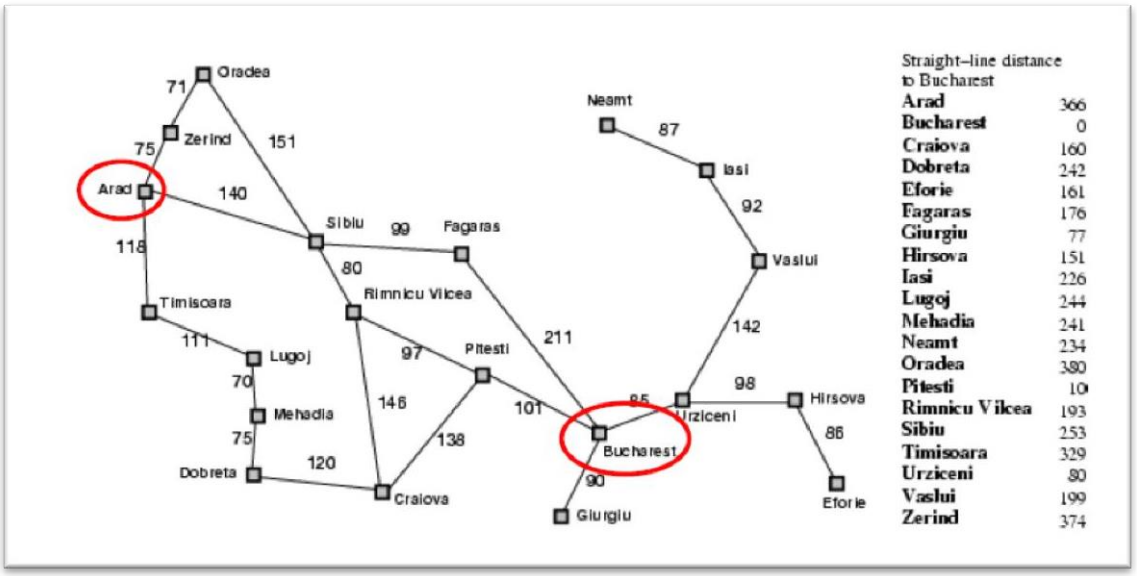
- Like the depth-first and breadth-first search, best-first search uses two-lists. open: to keep track of the frontier of the search. closed: to record states already visited.
- Order the states on open according to some heuristic estimate of their closeness to a goal.
- At each iteration through the loop, consider the most promising state on the open list next.
- When visiting a child state, if it is already on open or closed, the algorithm checks if the child is reached by a shorter path this time compared with the last time it arrived at this child.

Greedy Search Algorithm

Greedy search is a best-first strategy where we try to minimize the estimated cost to reach the goal. Since we are greedy, we always expand the node that is estimated to be closest to the goal state. Unfortunately, the exact cost of reaching the goal state usually can't be computed, but we can estimate it by using a cost estimate or heuristic function *h()*. When we are examining node **n**, then *h*(**n**) gives us the estimated cost of the cheapest path from n's state to the goal state. Of course, the better an estimate *h()* gives, the better and faster we will find a solution to our problem. Greedy search has similar behavior to depth-first search. Its advantages are delivered via the use of a quality heuristic function to direct the search.

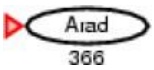
Example

Let us see how this works for route-finding problems in Romania, using the straight line distance heuristic, which we will call h_{SLD}. If the goal is Bucharest, initials heuristic function is Arad. We will need to know the straight-line distances to Bucharest, which are shown in Figure.

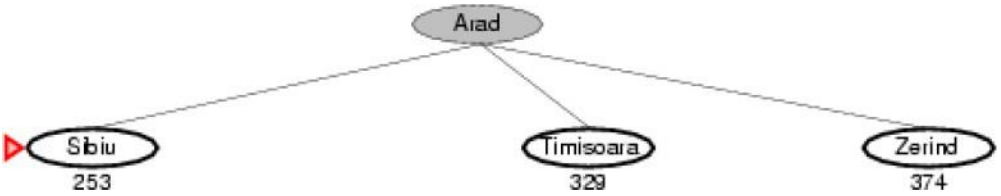


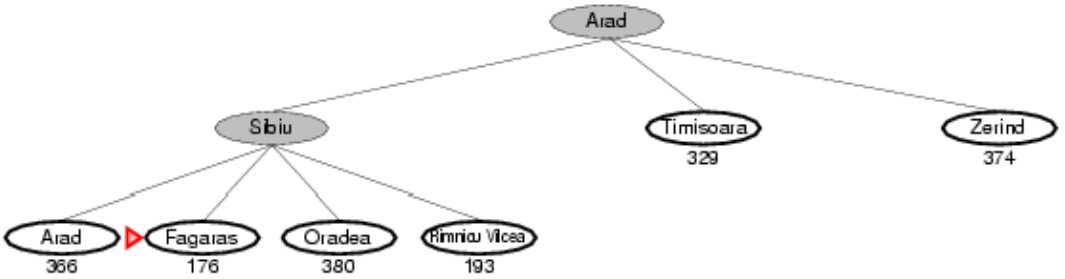
Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

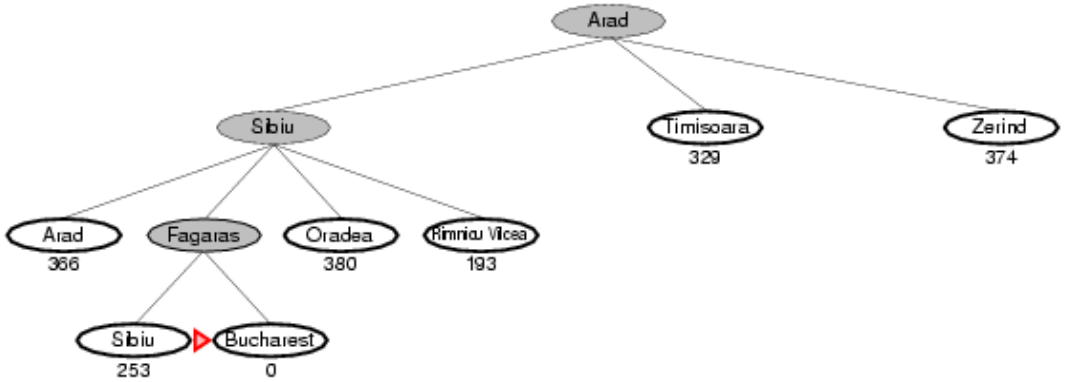
- Choose the heuristic function h(n)
  : initial state:



- Expand the node that has the lowest value of the heuristic function h(n)







- It is not complete and nor optimal