Solving problems by searching
Uninformed search algorithms

Discussion Class CS 171

Friday, October, 2nd

(Please read lecture topic material before and after each lecture on that topic)

Thanks to professor Kask

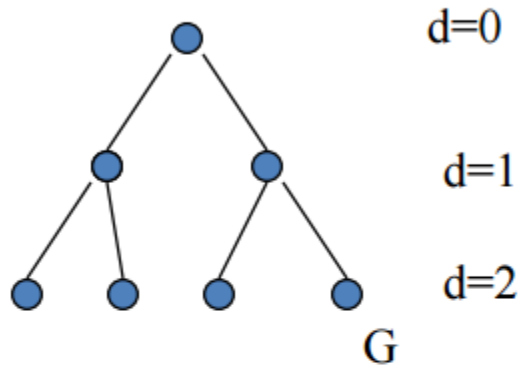
Some of the slides (page 2-7) were copied from his lectures.

Complexity of Depth-First Search?

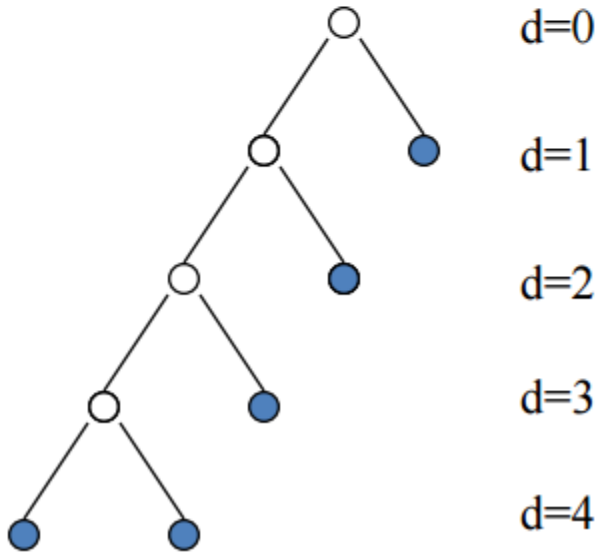
- Time Complexity
  - assume d is deepest path in the search space
  - assume (worst case) that there is 1 goal leaf at the RHS
  - so DFS will expand all nodes

$$=1 + b + b^2 + \ldots\ldots\ldots + b^d$$

$$= \mathbf{O(b^d)}$$



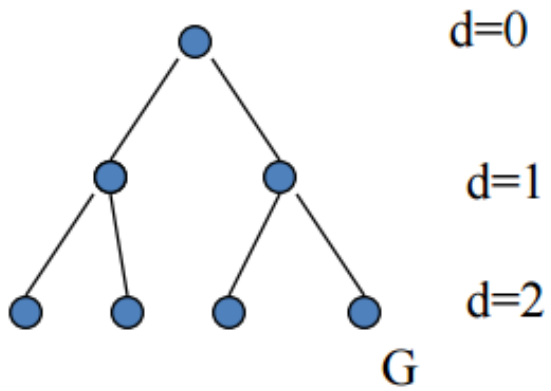
- Space Complexity (for tree-search)
  - how many nodes can be in the queue (worst-case)?
  - **O(bd) if deepest node at depth d**



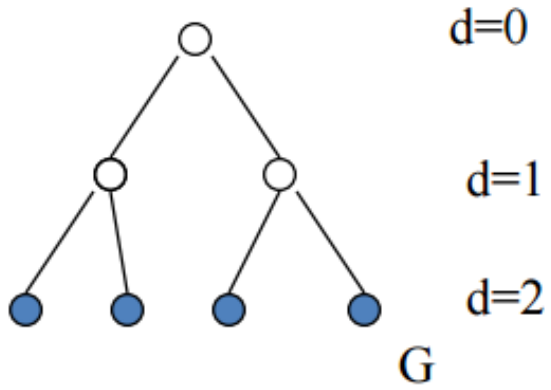
Complexity of Breadth-First Search

- **Time Complexity**
  - assume (worst case) that there is 1 goal leaf at the RHS
  - so BFS will expand all nodes

$= 1 + b + b^2 + \ldots\ldots\ldots + b^d$
$= \mathbf{O(b^d)}$



- **Space Complexity**
  - how many nodes can be in the queue (worst-case)?
  - at depth d there are b^d unexpanded nodes in the Q = $\mathbf{O(b^d)}$



time complexity analysis

- Suppose goal is at level g:
- DFS:
  - Best time : g
  - Worst time : $1+ b + b^2 + \ldots + b^g$
- BFS:
  - Best time : $1+ b + b^2 + \ldots + b^{g-1} +1$
  - Worst time : $1+ b + b^2 + \ldots + b^g$

Comparing BFS and DFS

- DFS is not optimal, BFS optimal if path cost is a non-decreasing function of depth, but BFS is not optimal in general.
- Time Complexity worse-case is the same, but

– In the worst-case BFS is always better than DFS

– Sometime, on the average DFS is better if:

  - many goals, no loops and no infinite paths
  - BFS is much worse memory-wise
  - DFS can be linear space
  - BFS may store the whole search space.

- In general
  - BFS is better if goal is not deep, if long paths, if many loops, if small search space
  - DFS is better if many goals, not many loops
  - DFS is much better in terms of memory

Iterative-Deepening Search (DFS)

- Every iteration is a DFS with a depth cutoff.

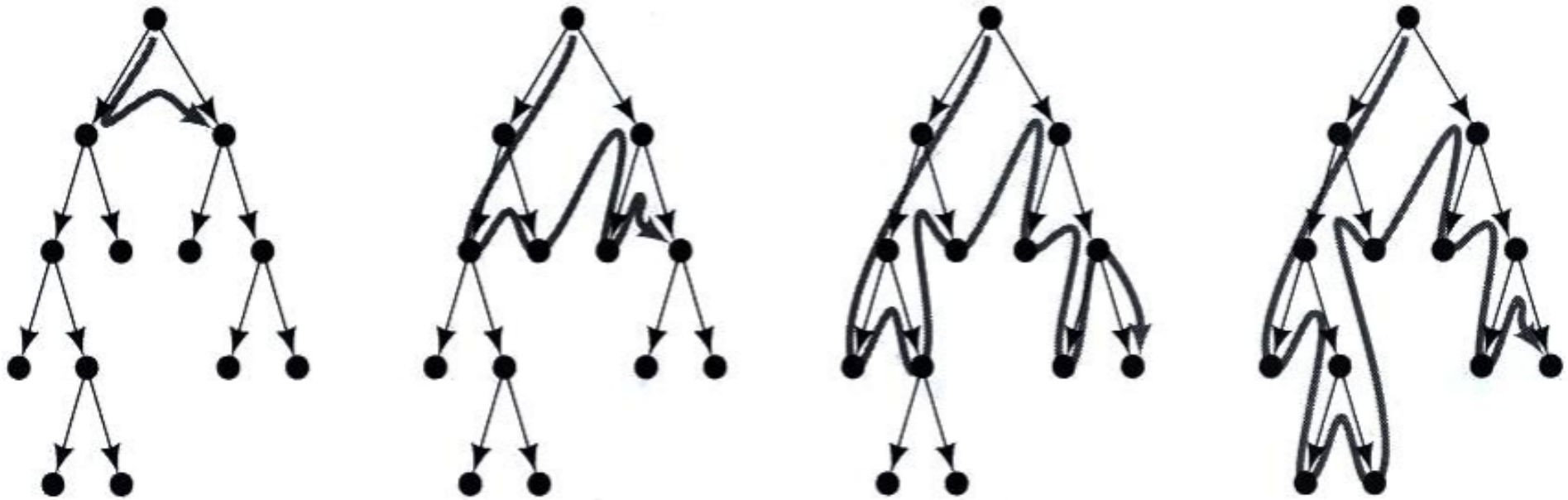
Iterative deepening (ID)

1. i = 1
2. While no solution, do
3. DFS from initial state S_0 with cutoff i
4. If found goal, stop and return solution, else, increment cutoff

Comments:

- IDS implements BFS with DFS
- Only one path in memory
- BFS at step i may need to keep 2^i nodes in OPEN

Iterative deepening search

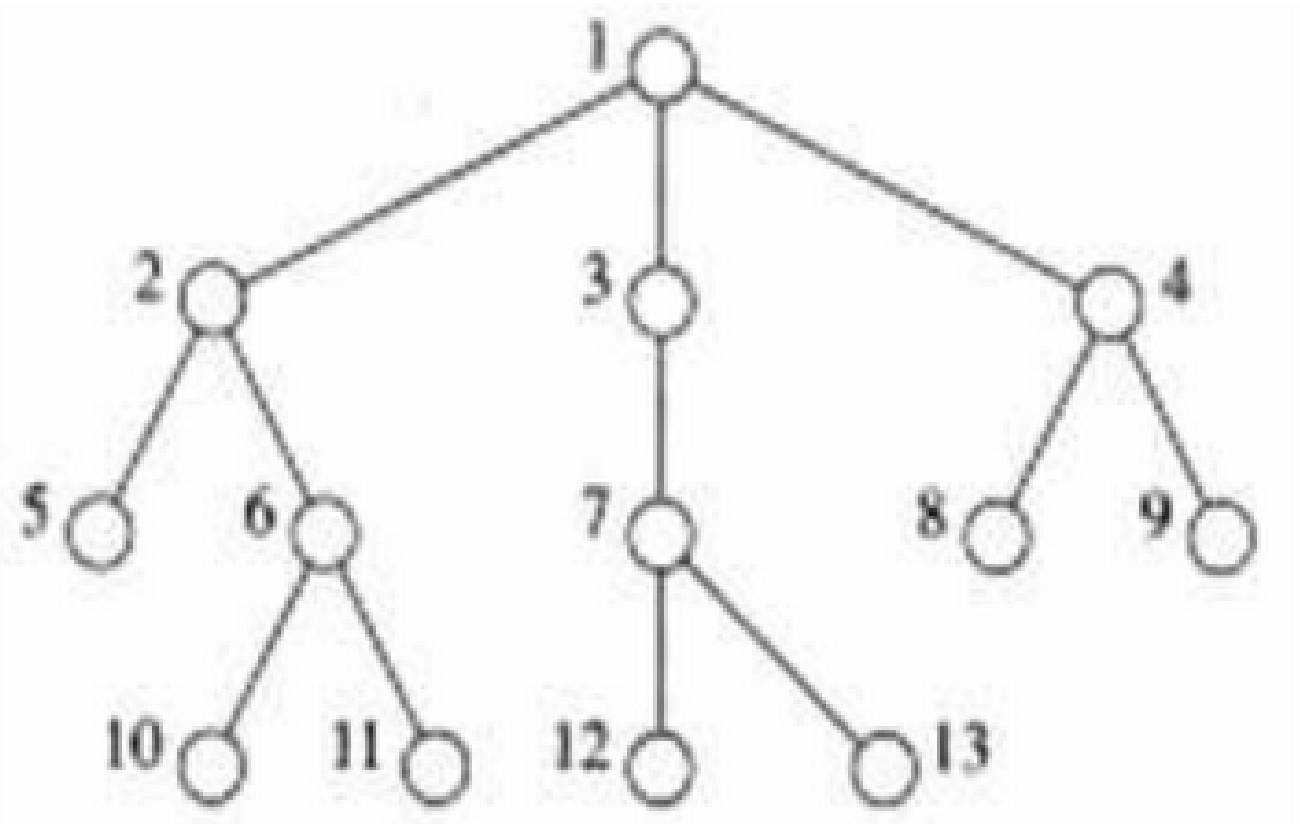


Stages in Iterative-Deepening Search

Iterative deepening search - Complexity

- Number of expanded nodes at each iteration:
- d = 0 : 1
- d = 1 : 1 + b
- d = 2 : 1 + b + b^2
- d = 3 : 1 + b + b^2 + b^3
- ….
- Total ?

Find BFS, DFS and Iterative deepening orders?



Djikestra (uniformed cost search)

```
1- create vertex set Queue
dist[source] = 0;
2- for each vertex v in Graph:
       if v ≠ source:
              dist[v] ← INFINITY
              prev[v] ← UNDEFINED
              add v to Q

3- while Q is not empty:                          // RUN THIS PART FOR V TIMES
       u ← vertex in Q with min dist[u]           // IT TAKES AT LEAST LOG(V)
       remove u from Q
       for each neighbor v of u:                  // RUN THIS PART FOR E TIMES
       alt ← dist[u] + length(u, v)
              if alt < dist[v]:
                     dist[v] ← alt
                     prev[v] ← u
```

Uniform Cost Search

- Expand lowest-cost OPEN node ($g(n)$)
- In BFS $g(n) = depth(n)$

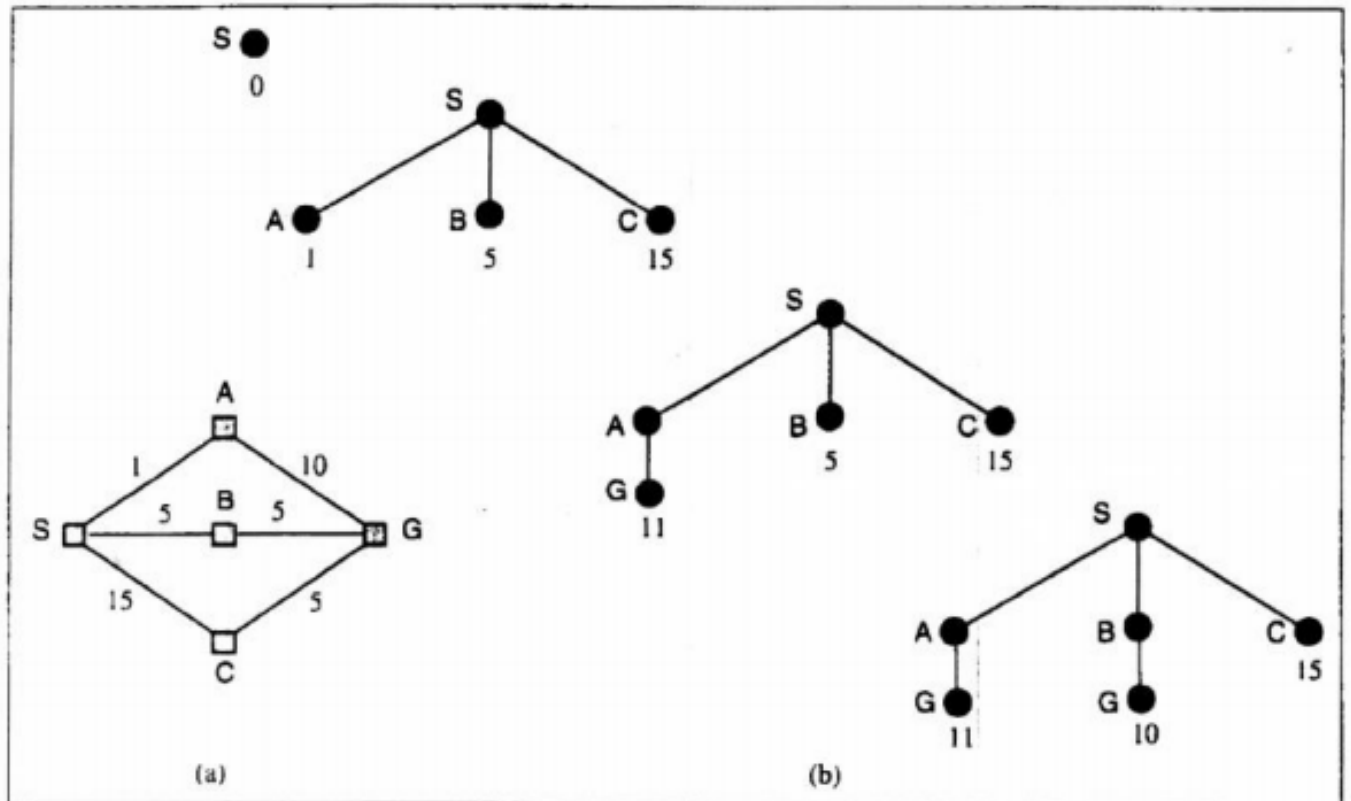
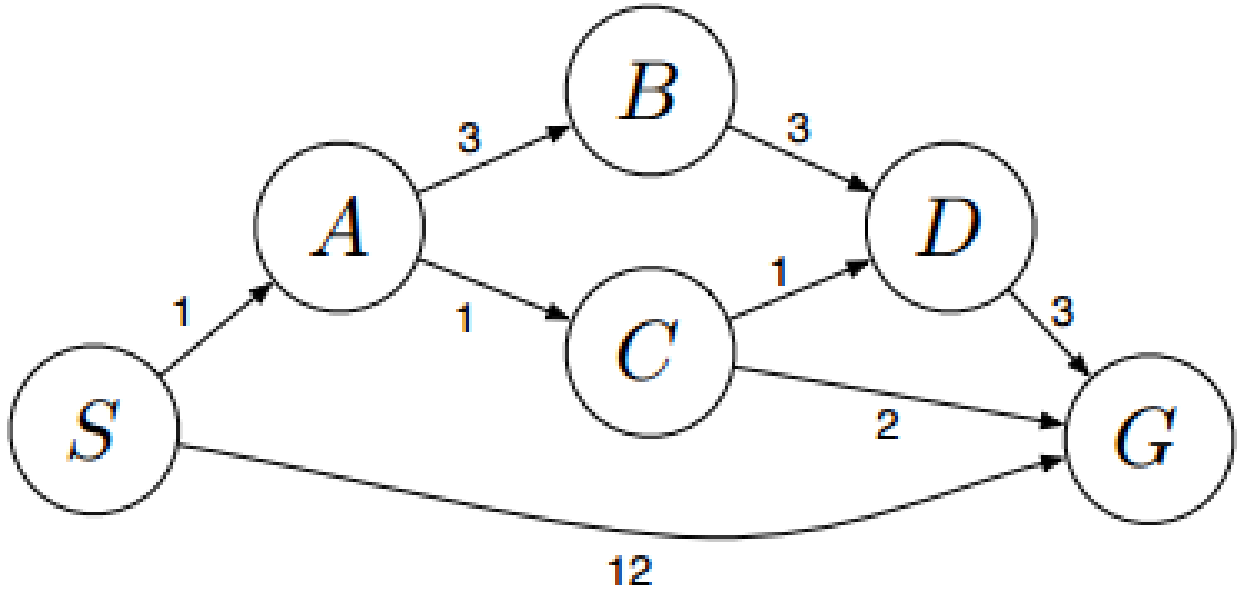


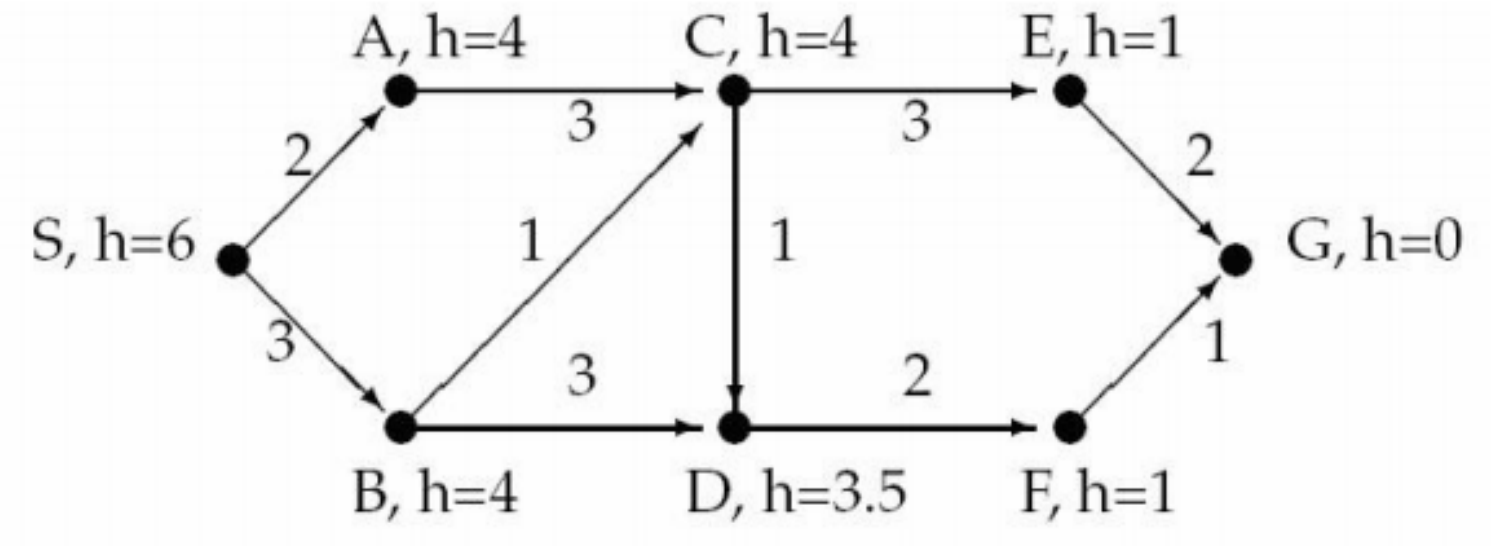
Figure 3.13 A route-finding problem. (a) The state space, showing the cost for each operator. (b) Progression of the search. Each node is labelled with $g(n)$. At the next step, the goal node with $g = 10$ will be selected.

Example

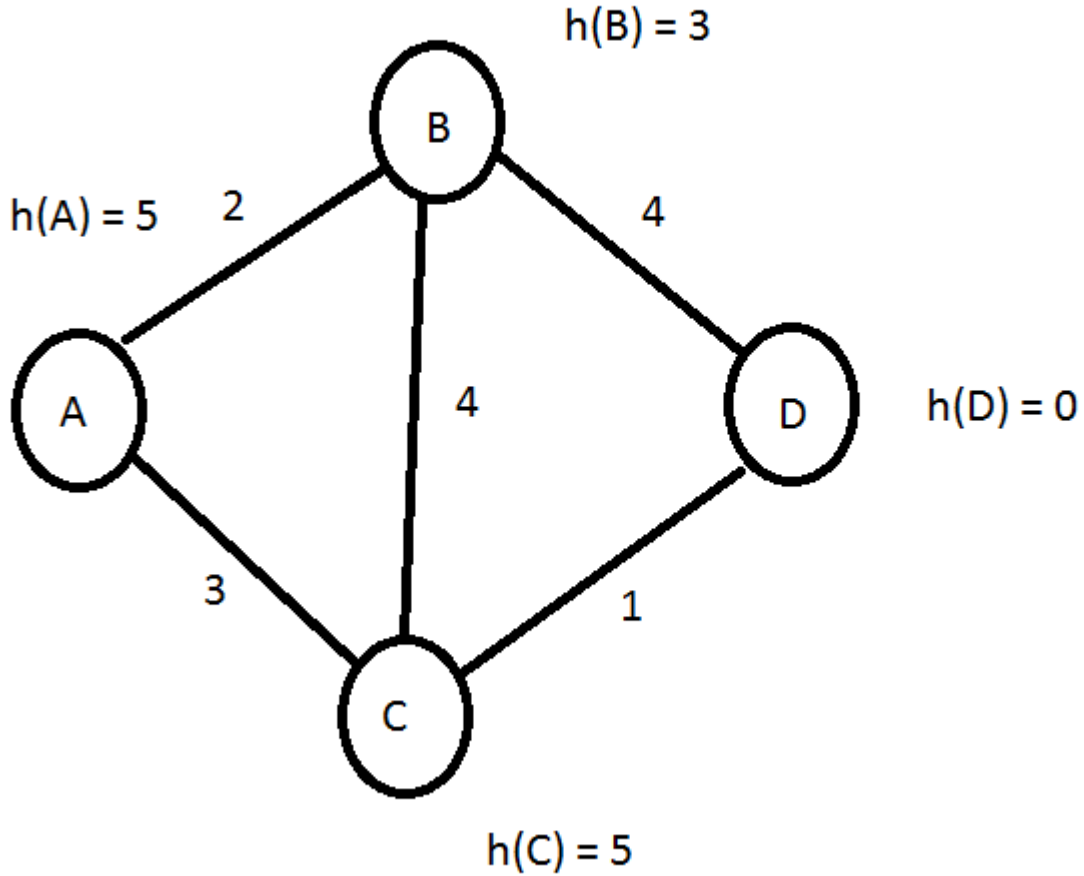
- Answer the following questions about the search problem shown above. Break any ties alphabetically. For the questions that ask for a path? (DFS, BFS and uniform cost)



A* algorithm



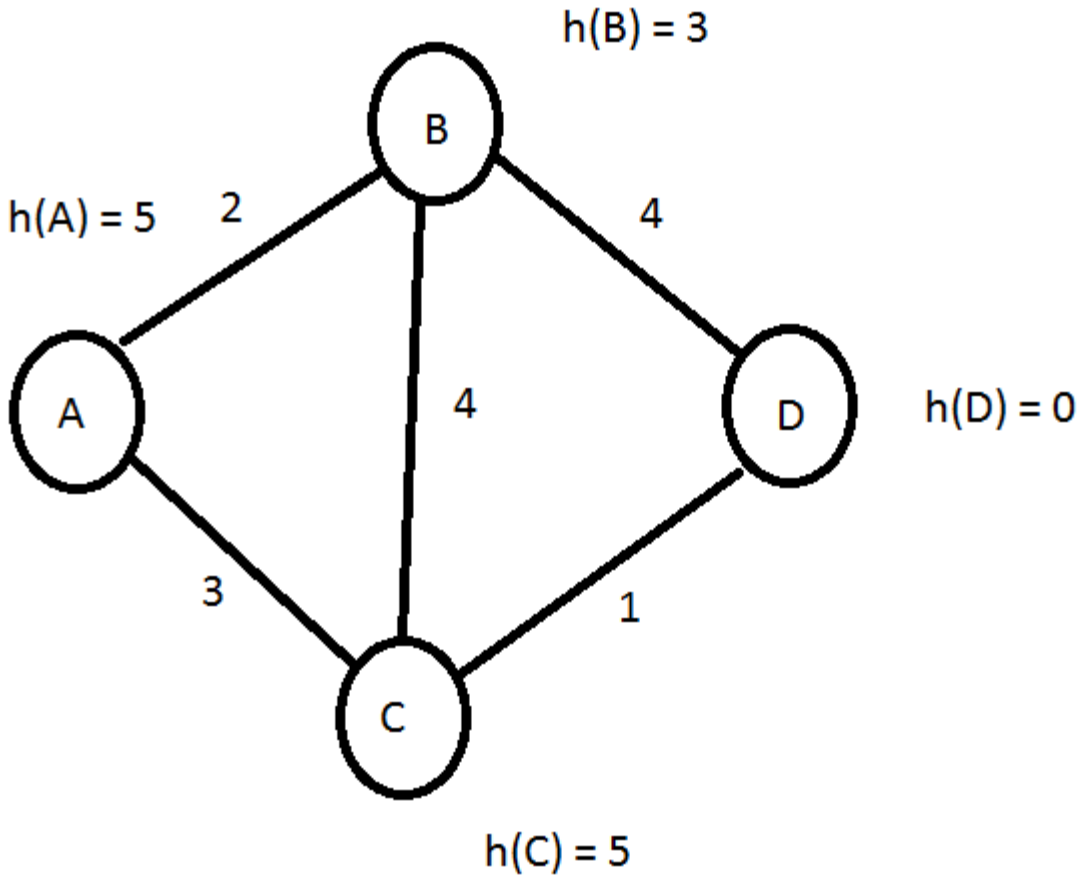
A* algorithm- Run it on this example?



A* algorithm

- Step1:
  - A: 5 , Select A
- Step2:
  - B=5 , C=8 , Select B
- Step 3:
  - C=8, D=6, Select D
- Step 4:
  - D is the goal.

  - Why A* doesn't work correctly?



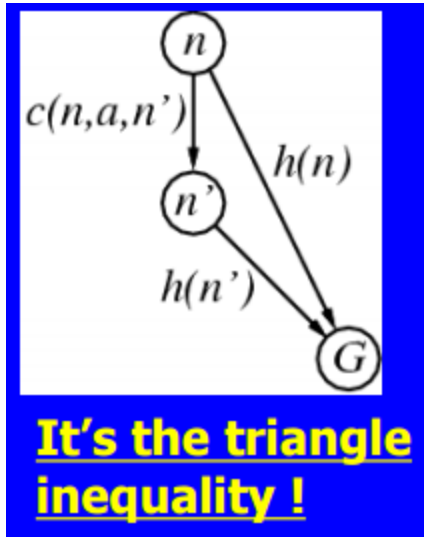
Consistent heuristics

- A heuristic is consistent if for every node n, every successor n' of n generated by any action a,
  - h(n) ≤ c(n,a,n') + h(n')
  - If h is consistent, we have
  - f(n’) = g(n’) + h(n’) (by def.)

    = g(n) + c(n,a,n') + h(n’) (g(n’)=g(n)+c(n.a.n’))

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

    f(n’) ≥ f(n)
  - i.e., f(n) is non-decreasing along any path.



- Theorem: If h(n) is consistent, A* using GRAPH-SEARCH is optimal

Admissible heuristics

- A heuristic h(n) is admissible if for every node n,

  $h(n) \leq h^*(n)$,

  where $h^*(n)$ is the true cost to reach the goal state from n.
- An admissible heuristic never overestimates the cost to reach the goal, i.e., it is optimistic (or at least, never pessimistic)

– Example: hSLD(n) (never overestimates actual road distance)

- Theorem: If h(n) is admissible, A* using TREE-SEARCH is optimal

Question

- Provide an example of a graph, which is not admissible so A* cannot find optimal answer?