Artificial Intelligence

3.4: Solving Problem by Searching

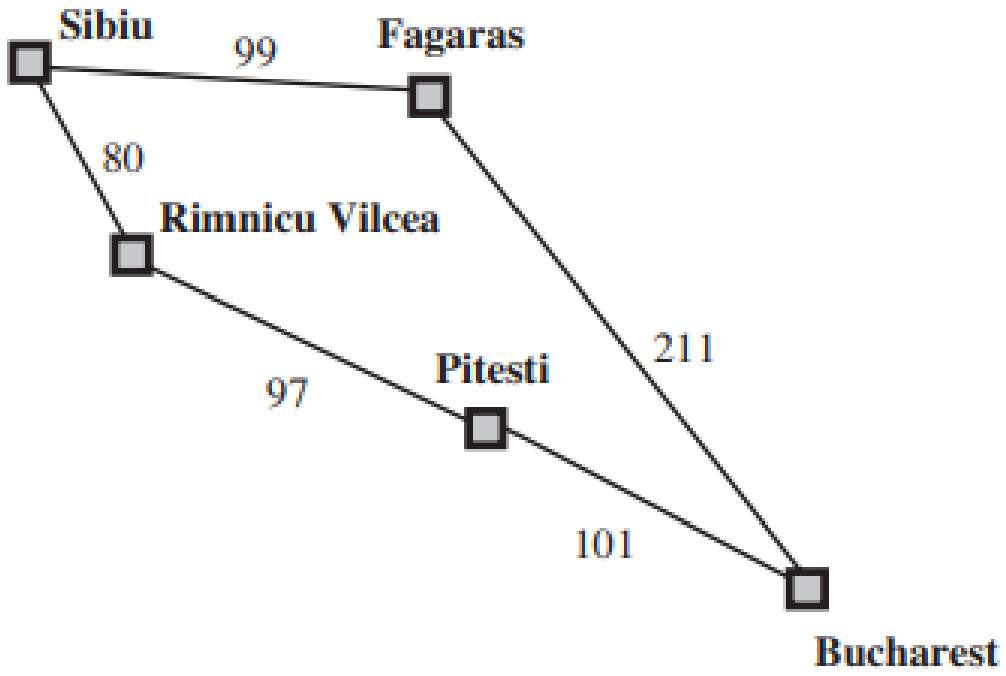
Uniform-cost search

- Expand least-cost unexpanded node-Cheapest-First Search
- Implementation:
  - *fringe* = queue ordered by path cost
- Equivalent to breadth-first if path cost of all edges is same

Insert the root into the queue
While the queue is not empty
Dequeue the maximum priority element from the queue
(If priorities are same, alphabetically smaller path is chosen)
If the path is ending in the goal state, print the path and exit
Else
Insert all the children of the dequeued element, with the cumulative costs as priority

Uniform Cost Search

The successors of Sibiu are Rimnicu Vilcea and Fagaras, with costs 80 and 99, respectively.

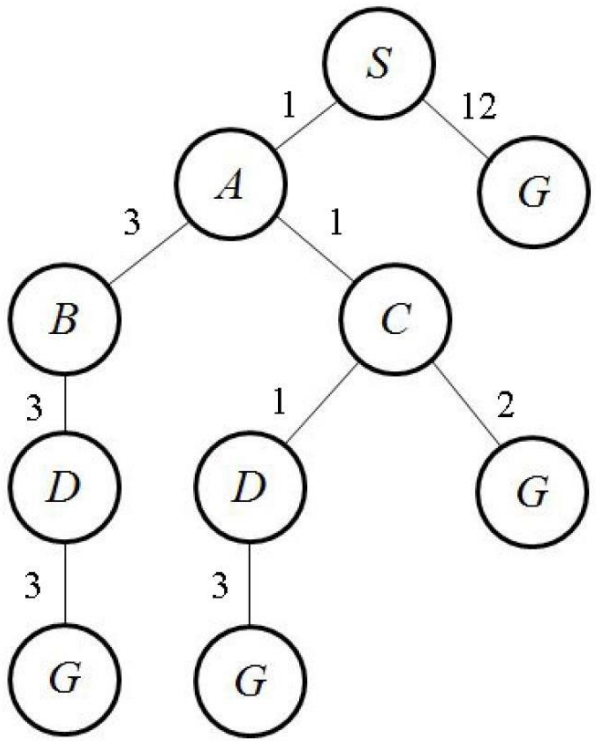


Part of the Romania state space, selected to illustrate uniform-cost search.

The least-cost node, Rimnicu Vilcea, is expanded next, adding Pitesti with cost 80 + 97=177.

The least-cost node is now Fagaras, so it is expanded, adding Bucharest with cost 99+211=310.

Now a goal node has been generated, but uniform-cost search keeps going, choosing Pitesti for expansion and adding a second path to Bucharest with cost 80+97+101= 278.



Initialization: {[S, 0]}

Iteration1: {[S->A, 1], [S->G, 12]}

Iteration2: {[S->A->C, 2], [S->A->B, 4], [S->G, 12]}

Iteration3: {[S->A->C->D, 3], [S->A->B, 4], [S->A->C->G, 4], [S->G, 12]}

Iteration4: {[S->A->B, 4], [S->A->C->G, 4], [S->A->C->D->G, 6], [S->G, 12]}

Iteration5: {[S->A->C->G, 4], [S->A->C->D->G, 6], [S->A->B->D, 7], [S->G, 12]}

Iteration6 gives the final output as S->A->C->G.

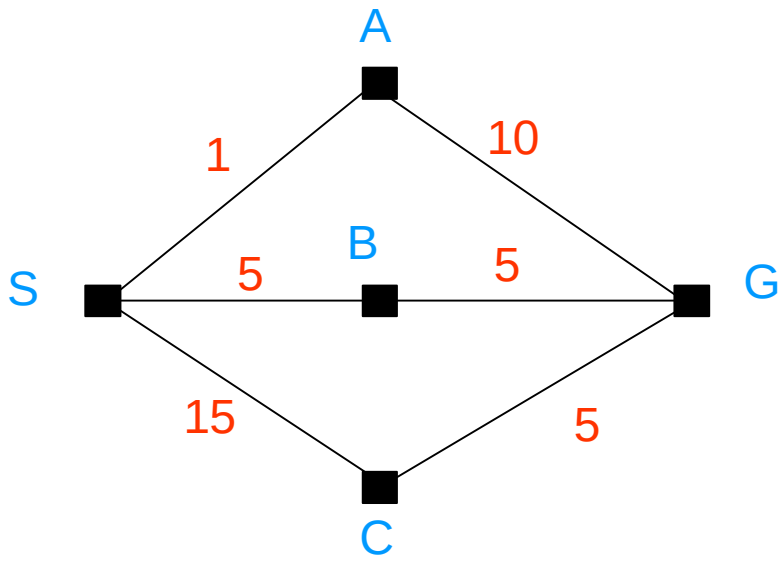
Uniform-cost Search

Uniform-cost search is guided by path costs rather than depths

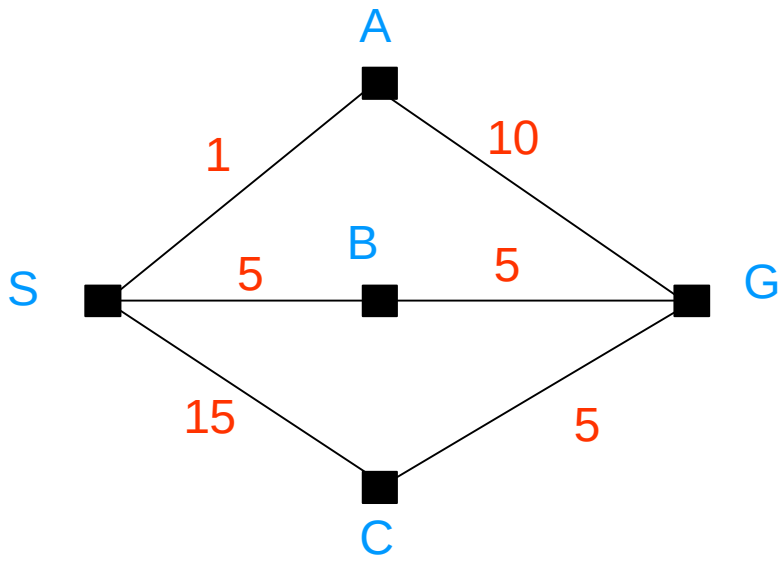
- Complete? Yes, if step cost ≥ ε (0 not allowed due to recursion/looping)
- Time? # of nodes with *g(path cost)* ≤ cost of optimal solution, $O(b^{ceiling(C^*/\varepsilon)})$ where *C^* is the cost of the optimal solution, ε some small positive constant*
- Space? # of nodes with *g* ≤ cost of optimal solution $O(b^{ceiling(C^*/\varepsilon)})$
- Optimal? Yes – nodes expanded in increasing order of *g(n)*

When all step costs are the same, uniform-cost search is similar to breadth-first search, except that the latter stops as soon as it generates a goal, whereas uniform-cost search examines all the nodes at the goal's depth to see if one has a lower cost

https://www.youtube.com/watch?v=dRMvK76xQJI



- BFS vs UCS
- BFS?
- UCS?



- BFS will find the path SAG, with a cost of 11, but SBG is cheaper with a cost of 10
- Uniform Cost Search will find the cheaper solution (SBG).

Depth First Search

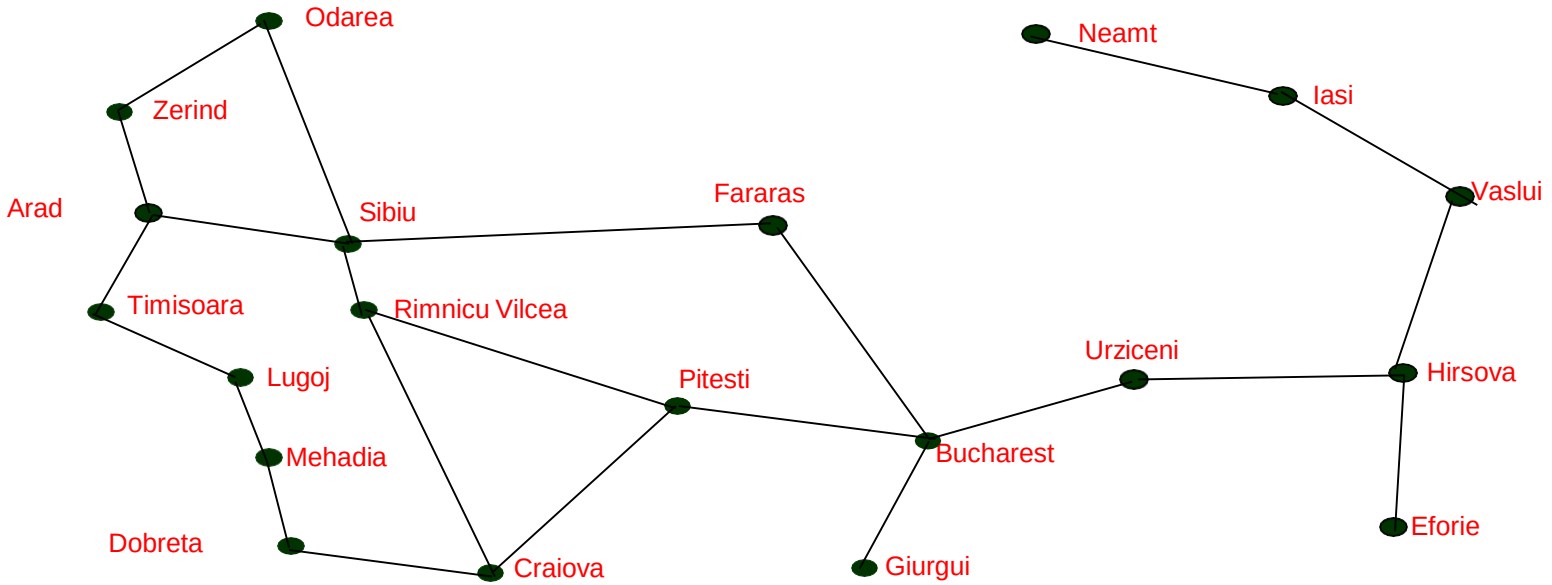
- If DFS goes down an infinite branch it will not terminate if it does not find a goal state.
- If it does find a solution, there may be a better solution at a lower level in the tree. Therefore, depth first search is neither complete nor optimal.
- DFS may never terminate as it could follow a path that has no solution (e.g., Chess: repeated moves result in similar configuration)
- **Depth Limited Search (DLS)** solves this by imposing a depth limit, at which point the search terminates that particular branch.

Depth Limited Search - Observations

- If the depth parameter, l, is set deep enough then we are guaranteed to find a solution if one exists
  - Therefore, it is complete if l >=d (d=depth of solution)

- Space requirements are O(bl)
- Time requirements are $O(b^l)$
- DLS is not optimal

Map of Romania



On the Romania map there are 20 towns so any town is reachable in 19 steps

diameter of state space: any city can be reached from any other city in at most 9 steps

Depth-limited search

* depth-first search with depth limit *l*,

i.e., nodes at depth *l* have no successors

- Depth-first search can be viewed as a special case of depth-limited search with *l* =∞

•Depth-limited search can terminate with two kinds of failure:

- Standard failure: no solution
- Cutoff: no solution within depth limit

Recursive implementation:

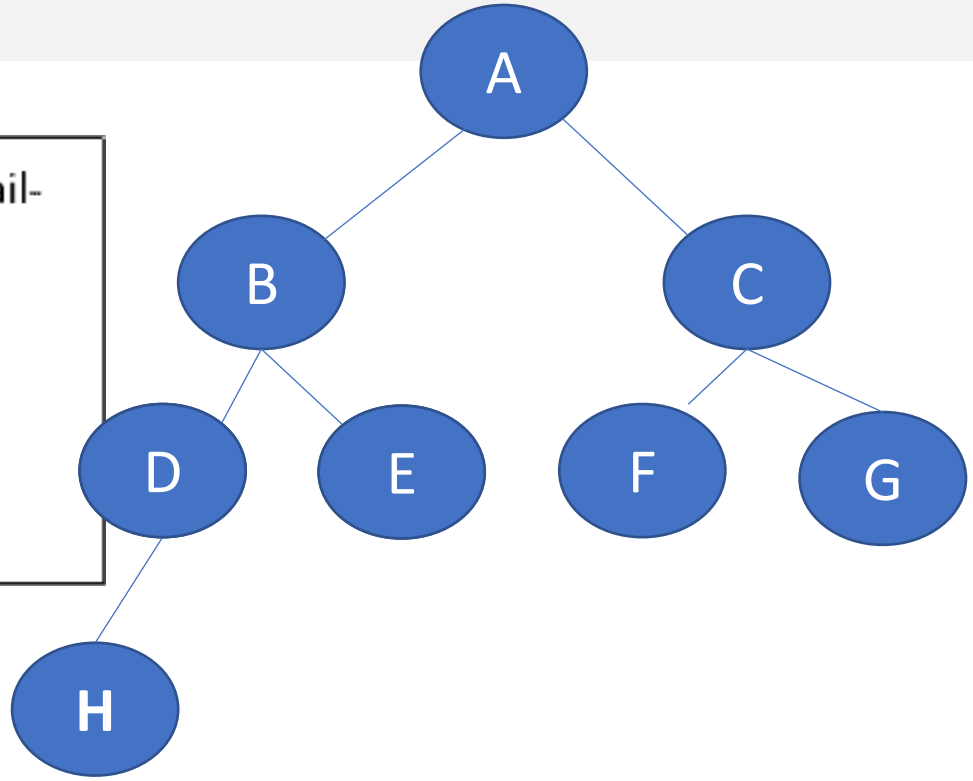
```
function DEPTH-LIMITED-SEARCH( problem, limit) returns soln/fail/cutoff
  RECURSIVE-DLS(MAKE-NODE(INITIAL-STATE[problem]), problem, limit)

function RECURSIVE-DLS(node, problem, limit) returns soln/fail/cutoff
  cutoff-occurred? ← false
  if GOAL-TEST[problem](STATE[node]) then return SOLUTION(node)
  else if DEPTH[node] = limit then return cutoff
  else for each successor in EXPAND(node, problem) do
      result ← RECURSIVE-DLS(successor, problem, limit)
      if result = cutoff then cutoff-occurred? ← true
      else if result ≠ failure then return result
  if cutoff-occurred? then return cutoff else return failure
```

Iterative Deepening Search (vs DLS)

- The problem with DLS is choosing a depth parameter
- Setting a depth parameter to 19 is obviously wasteful if using DLS
- IDS overcomes this problem by trying depth limits of 0, 1, 2, …, n. In fact, it is combining BFS and DFS

Iterative deepening search

```
function ITERATIVE-DEEPENING-SEARCH( problem) returns a solution, or failure
    inputs: problem, a problem
    for depth ← 0 to ∞ do
        result ← DEPTH-LIMITED-SEARCH( problem, depth)
        if result ≠ cutoff then return result
```

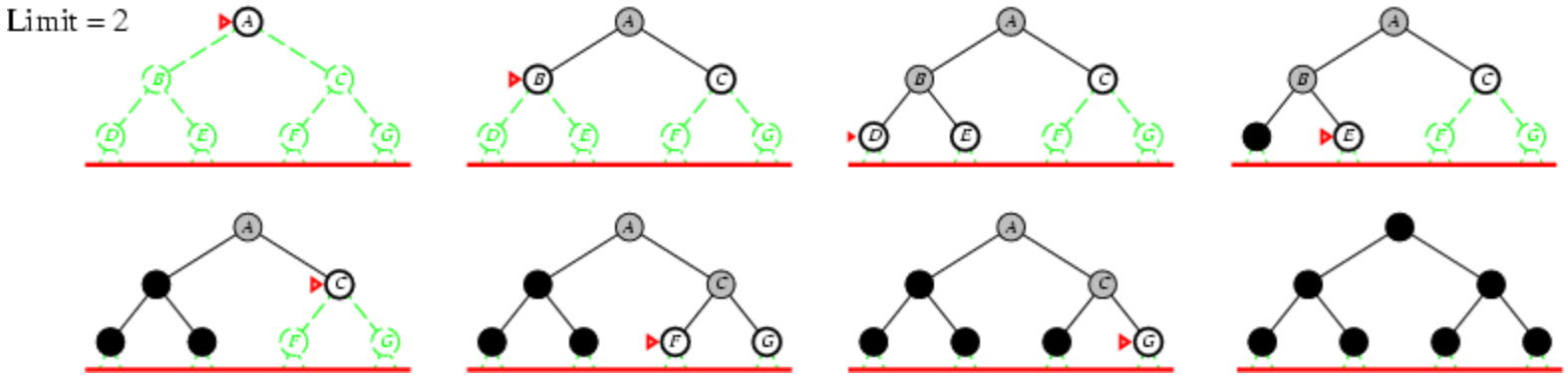


Iterative deepening search *l* =0

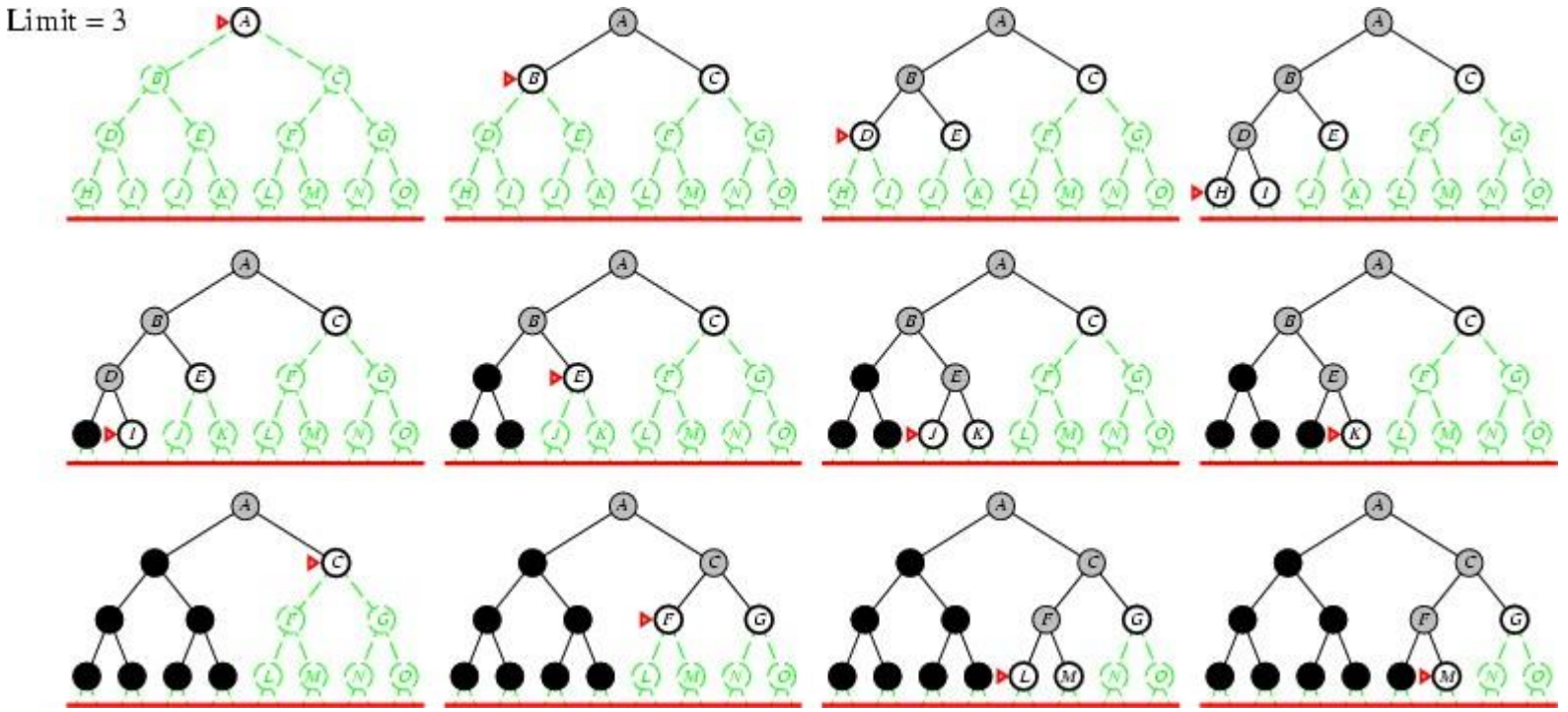
Limit = 0

Iterative deepening search *l* =1

Iterative deepening search *l* =2



Iterative deepening search *l* =3



Iterative Deepening Search - Observations

- IDS may seem wasteful as it is expanding the same nodes many times. In fact, when b=10 only about 11% more nodes are expanded than for a BFS or a DLS down to level d
- Time Complexity = $O(b^d)$
- Space Complexity = O(bd)
- *For large search spaces, where the depth of the solution is not known, IDS is the preferred uninformed search method*

Iterative deepening search

- Number of nodes generated in a depth-limited search to depth *d* with branching factor *b*:

$$N_{DLS} = b^0 + b^1 + b^2 + \ldots + b^{d-2} + b^{d-1} + b^d$$

- Number of nodes generated in an iterative deepening search to depth *d* with branching factor *b*:

$$N(\mathrm{IDS}) = (d)b + (d-1)b^2 + \cdots + (1)b^d$$

$$N_{IDS} = (d+1)b^0 + d\,b^1 + (d-1)b^2 + (d-2)b^3 + (d-3)b^4 + b^d$$

- In an **iterative deepening search**, the nodes **on the bottom level** (depth d) are generated once, those on the next-to-bottom level are generated twice, and so on, up to the children of the root, which are generated d times.

Iterative deepening search

- For *b* = *10*, *d* = *5*,
  - $N_{DLS} = 1 + 10 + 100 + 1{,}000 + 10{,}000 + 100{,}000 = 111{,}111$
  -
  - $N_{IDS} = 6 + 50 + 400 + 3{,}000 + 20{,}000 + 100{,}000 = 123{,}456$
- Overhead = (123,456 - 111,111)/111,111 = 11%

Properties of iterative deepening search

- Complete? Yes
- Time? $(d+1)b^0 + d\,b^1 + (d-1)b^2 + \ldots + b^d = O(b^d)$
- Space? $O(bd)$
- Optimal? Yes, if step cost = 1

Blind Searches – Summary

Evaluation	Breadth First	Uniform Cost	Depth First	Depth Limited	Iterative Deepening
Time	B^D	B^D	B^M	B^L	B^D
Space	B^D	B^D	BM	BL	BD
Optimal?	Yes	Yes	No	No	Yes
Complete?	Yes	Yes	No	Yes, if L>= D	Yes

B = Branching factor

D = Depth of solution

M = Maximum depth of the search tree

L = Depth Limit

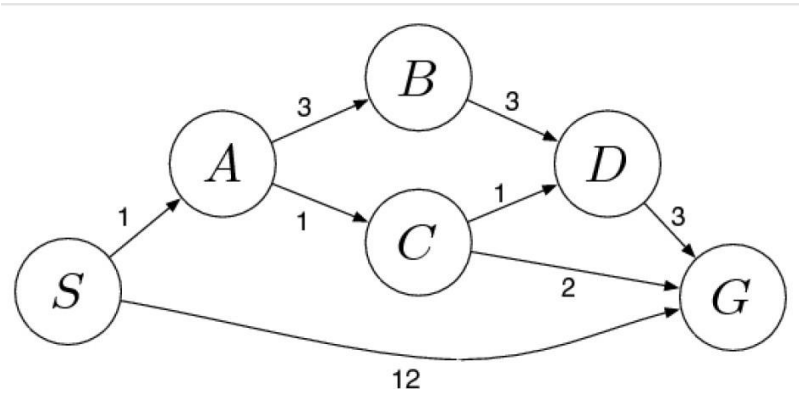
Uninformed Search

- Why are all these strategies called uninformed?

  Because they **do not consider any information about the states (end nodes)** to decide which path to expand first on the frontier

  In other words, they are general they do not take into account the **specific nature of the problem**.

Repeated states / Loops



Summary

- Problem formulation usually requires abstracting away real-world details to define a state space that can feasibly be explored
- Variety of uninformed search strategies
- Iterative deepening search uses only linear space and not exceedingly more time than other uninformed algorithms

Homework

- Readings
  - CH 3 Section 3.1-3.4.5