Graph and Heuristic Search

Lecture 2

Ning Xiong

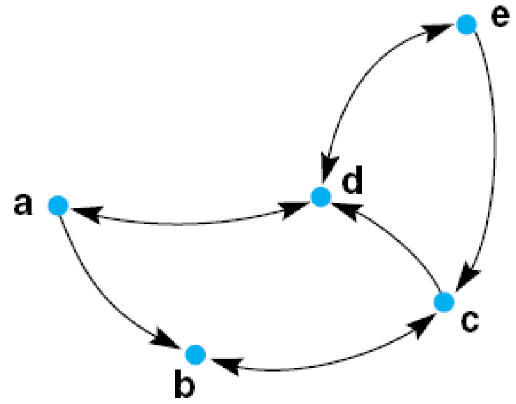
Mälardalen University

Agenda

- Uninformed graph search
  - breadth-first search on graphs
  - depth-first search on graphs
  - uniform-cost search on graphs

- General informed (heuristic) search algorithms
  - greedy best-first
  - A* search

Loop with graph search

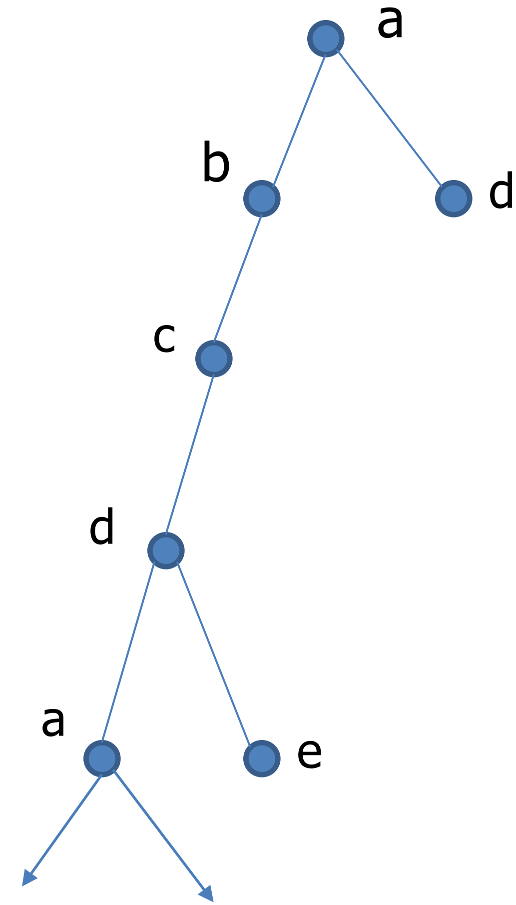


Nodes = {a,b,c,d,e}

Arcs = {(a,b),(a,d),(b,c),(c,b),(c,d),(d,a),(d,e),(e,c),(e,d)}

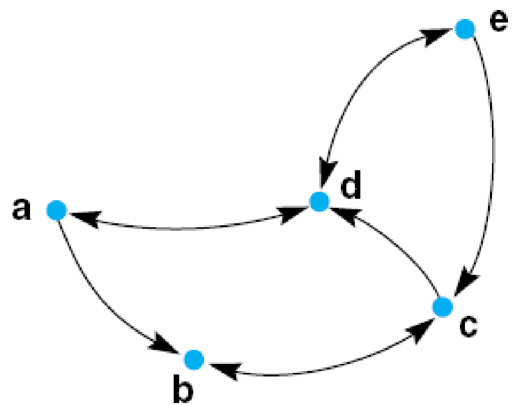
Initial state: a
Goal state: e

Depth-first



- In a general graph, there exists the risk of **loop or**
- **Cycle** making the search process continue for ever

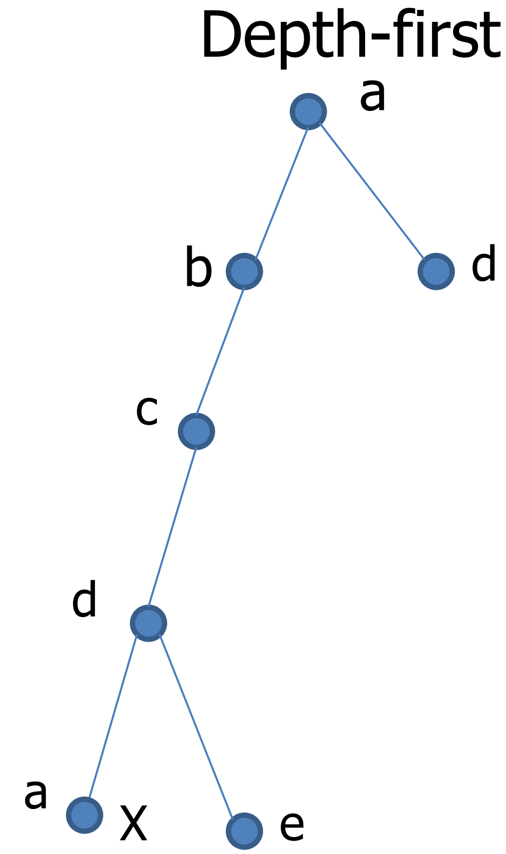
Method to avoid loop



Nodes = {a,b,c,d,e}

Arcs = {(a,b),(a,d),(b,c),(c,b),(c,d),(d,a),(d,e),(e,c),(e,d)}

Initial state: a
Goal state: e



- If we memorize the nodes that previously have been expanded, we can avoid picking state ’**a**’ for expansion for the second time. Consequently the search can terminate and find the goal **e**.

Avoiding Loop

- Memorize all expanded nodes in a Closed list
- All nodes in Closed list should be blocked for processing again

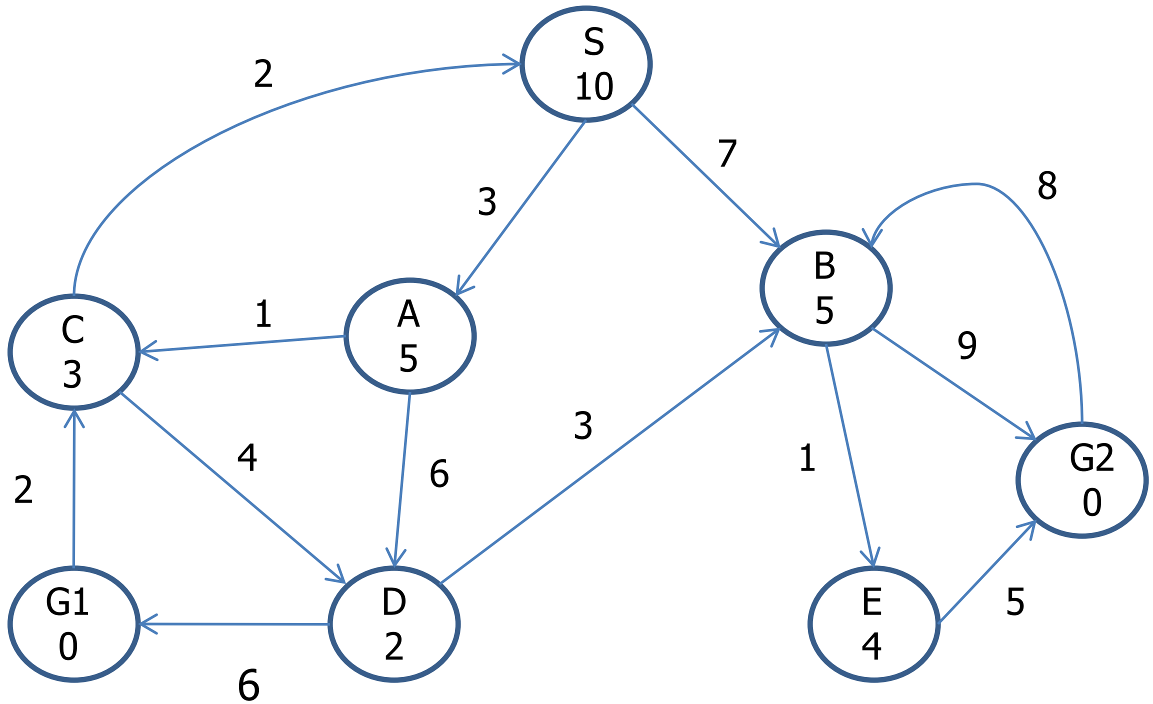
General graph search strategy:

1. Remove the first node from the Fringe list, assign it to s

2. If s is the goal, terminate with found solution

3.1 Add s to the Closed list
3.2 Expand s, discard the children of s that are in the Closed list
3.3 Insert the remaining children of s into Fring, with some order

Worked Example

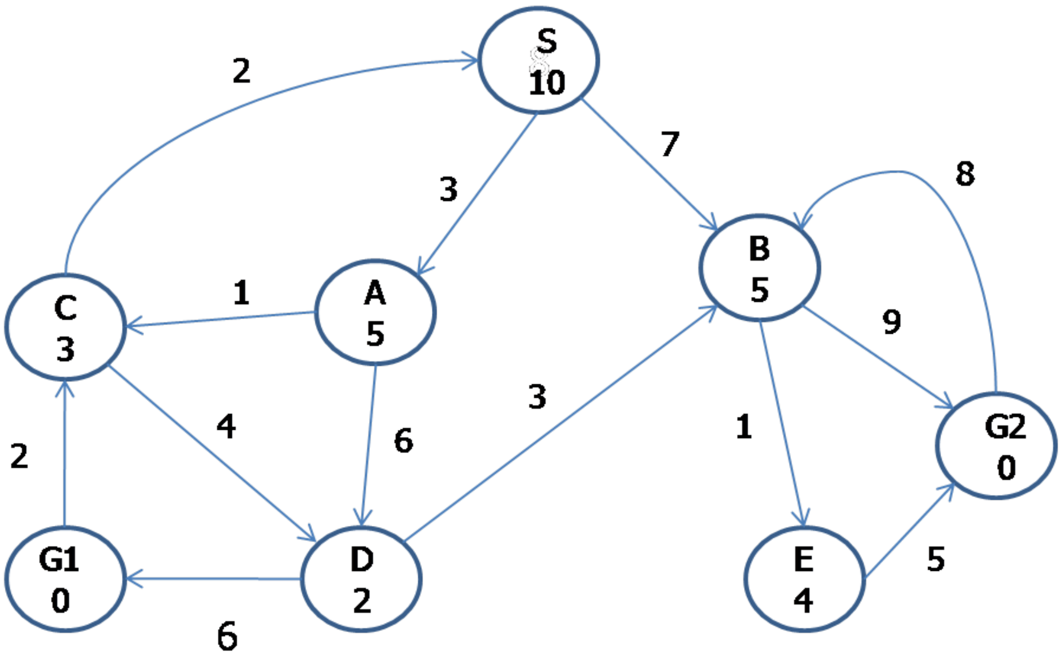


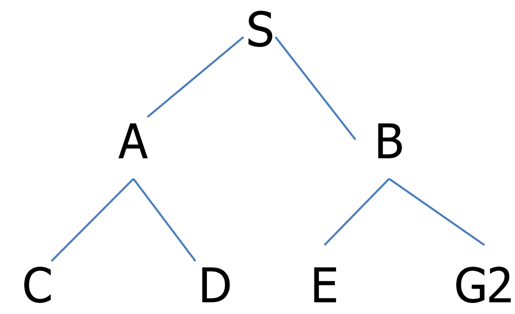
Initial: S
Goal: G1, G2
Other states: A, B, C, D, E

Arc from C to S: 2
Arc from S to A: 3
Arc from A to C: 1
Arc from A to D: 6
Arc from C to D: 4
Arc from D to G1: 6
Arc from G1 to C: 2

Arc from S to B: 7
Arc from D to B: 3
Arc from B to E: 1
Arc from B to G2: 9
Arc from E to G2: 5
Arc from G2 to B: 8

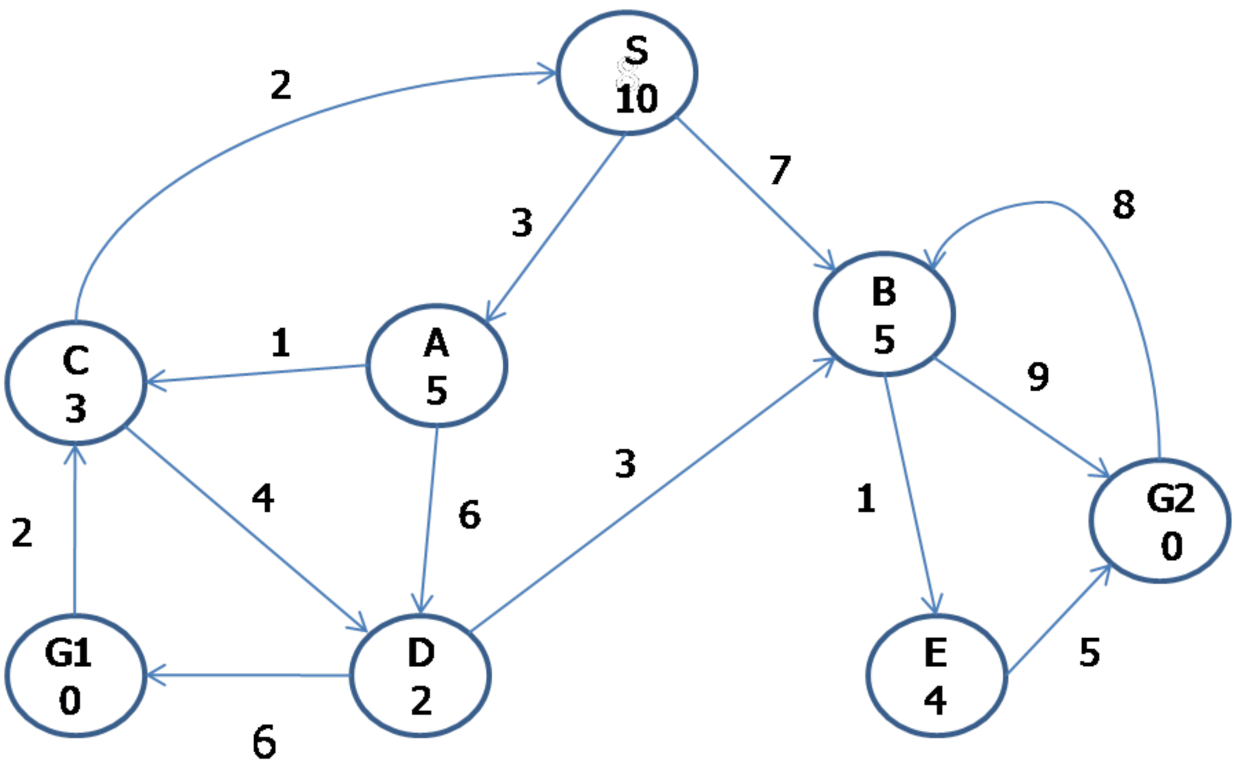
Breadth-First Graph Search

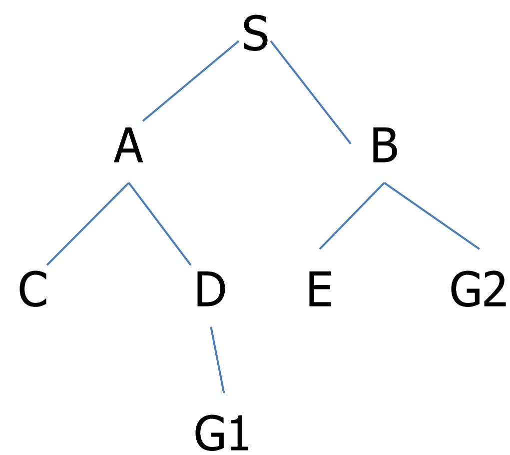




1. Expand S: generate A and B
2. Expand A: generate C and D
3. Expand B: generate E and G2

Breadth-First Graph Search





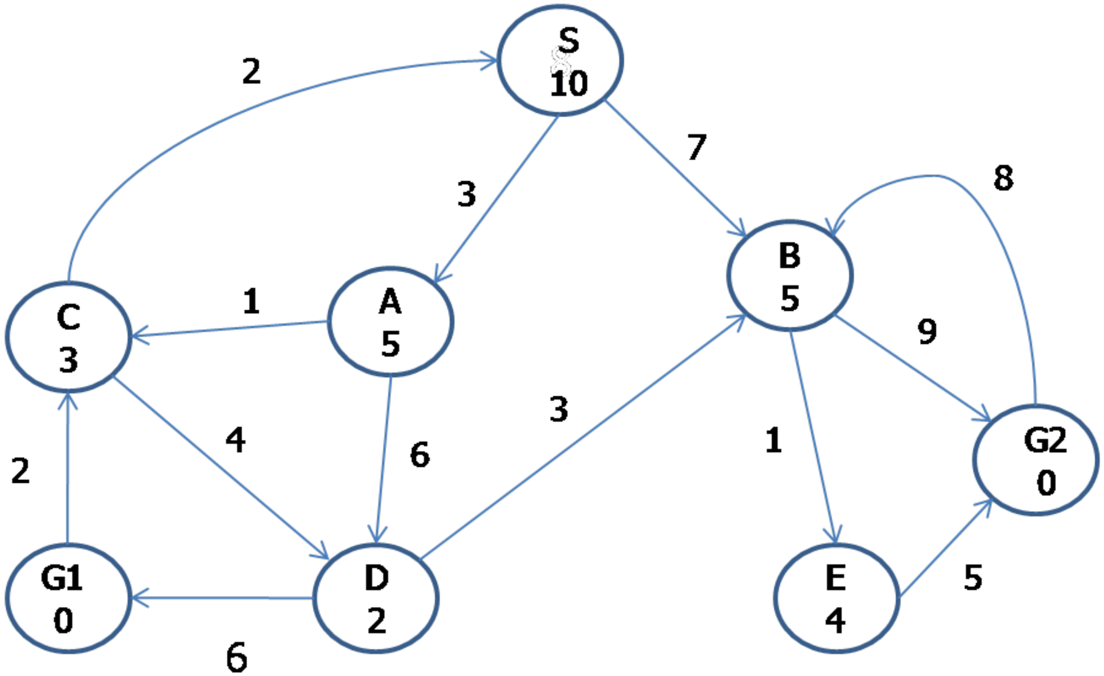
4. Expand C: no new

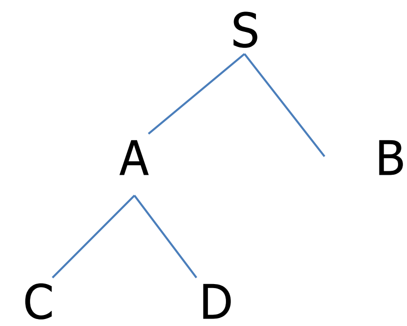
5. Expand D: generate G1

6. Expand E: no new

7. G2 is found as goal, trace back from G2 to get the path

Depth-First Graph Search

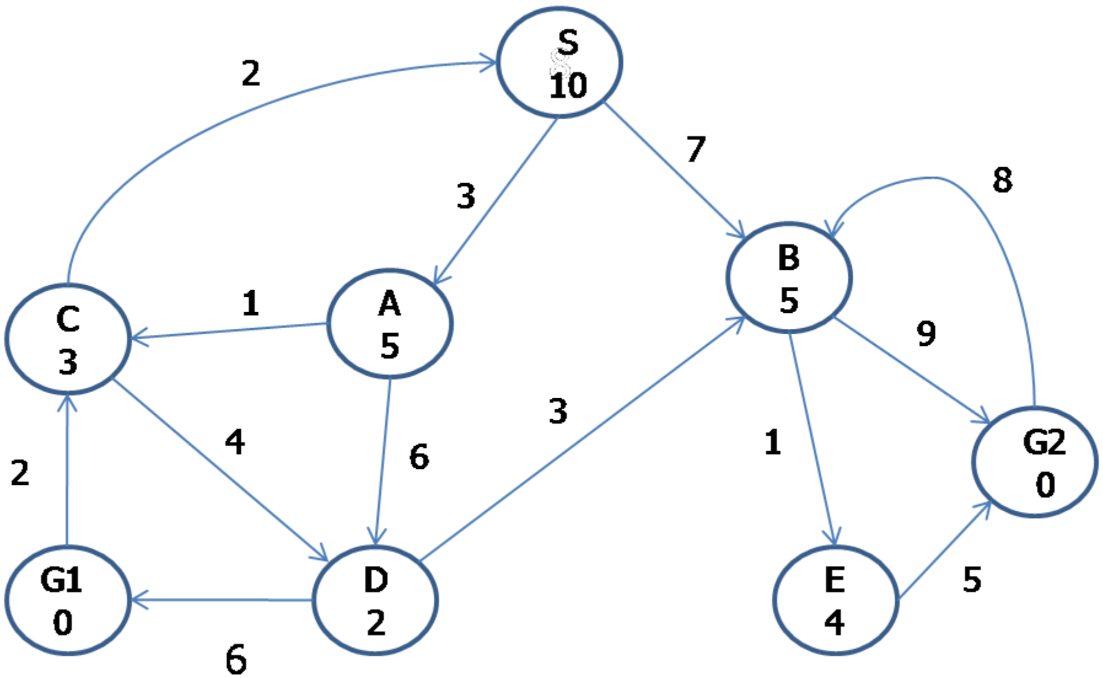


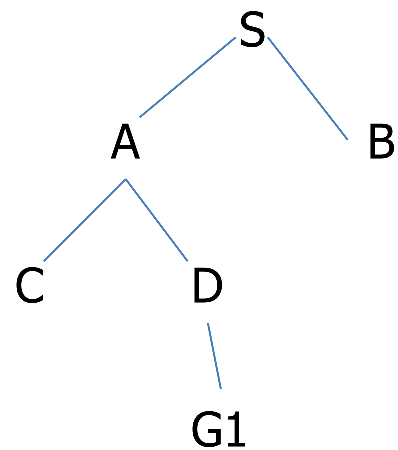


1. Expand S: generate A and B

2. Expand A: generate C and D

Depth-First Graph Search





3. Expand C: no new

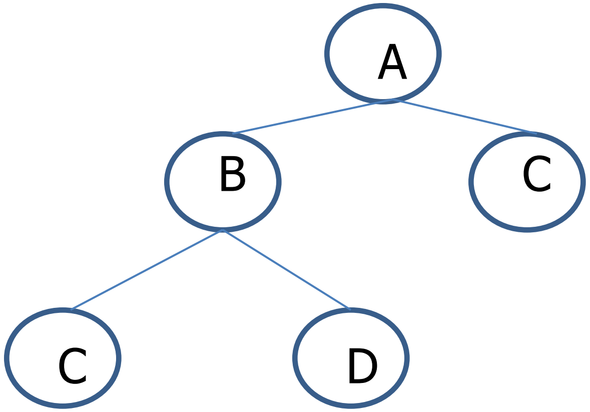
4. S is in the Closed List

4. Expand D: generate G1

5. G1 is checked as the goal, trace back to get the path

Uniform-Cost Graph Search

A child node of B has the state already in fringe, meaning a second path. We may discard one path based on path cost. The node and path with larger cost can be removed without altering final solution



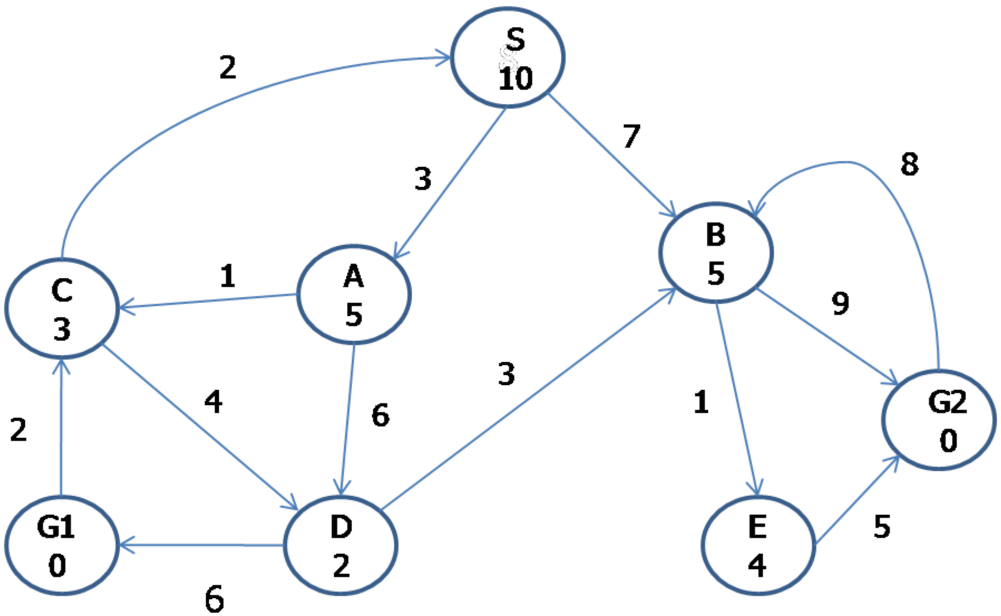
Fringe=(B, C)

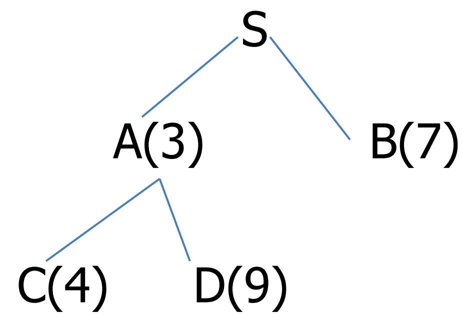
1. Remove the first node from the Fringe list, assign it to s

2. If s is the goal, terminate with found solution

3.1 Add s to the Closed list
3.2 Expand s, discard the children of s that are in the Closed list
3.3 Insert the remaining children of s into Fring, order the nodes in Fringe with increasing path cost. **Check nodes with same state and remove those having larger path costs.**

Uniform-Cost Graph Search

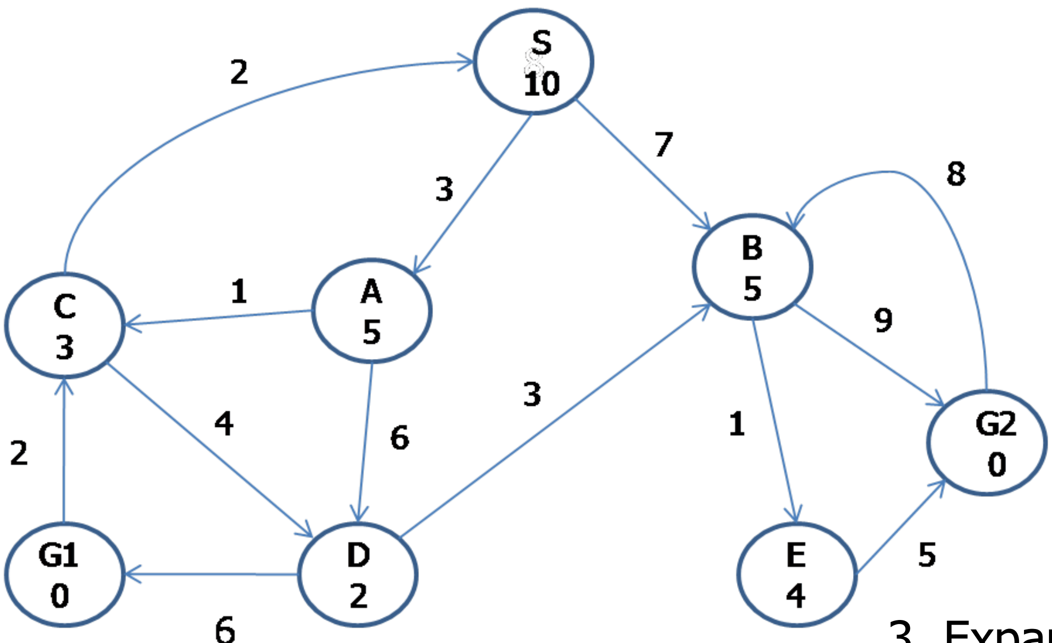


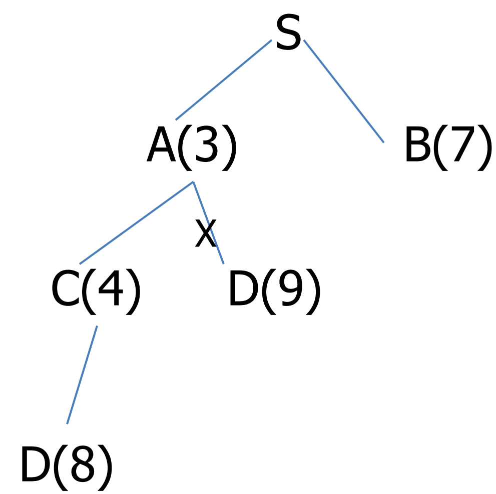


1. Expand S: generate A(3) and B(7)
2. Expand A: generate C(4) and D(9)

Fringe=[C(4), B(7), D(9)]

Uniform-Cost Graph Search

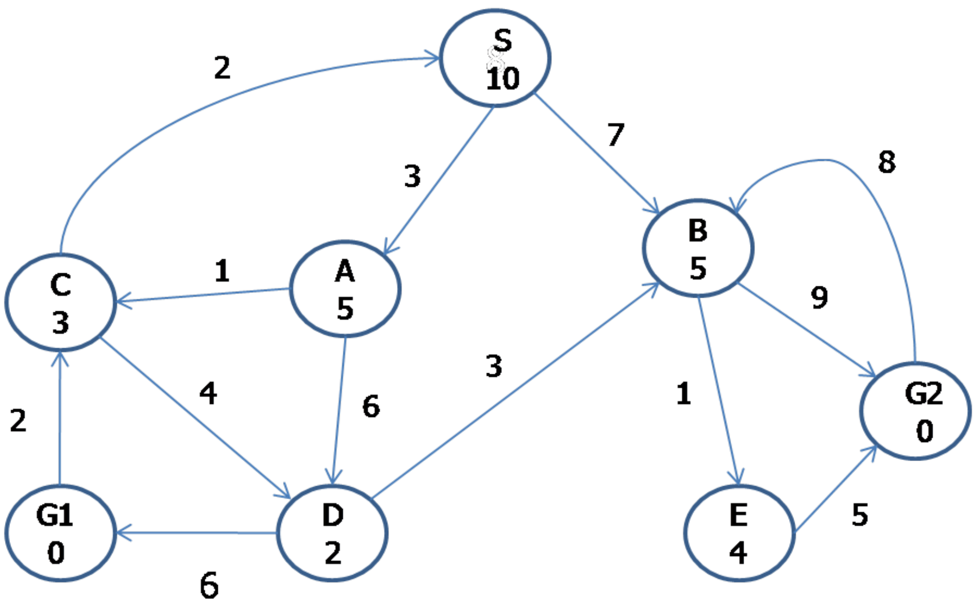


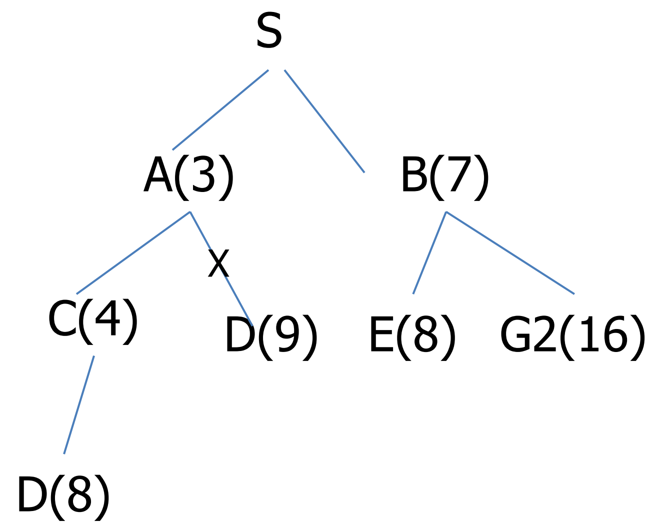


3. Expand C: D(8) and remove D(9) in Fringe

Fringe=[B(7), D(8)]

Uniform-Cost Graph Search

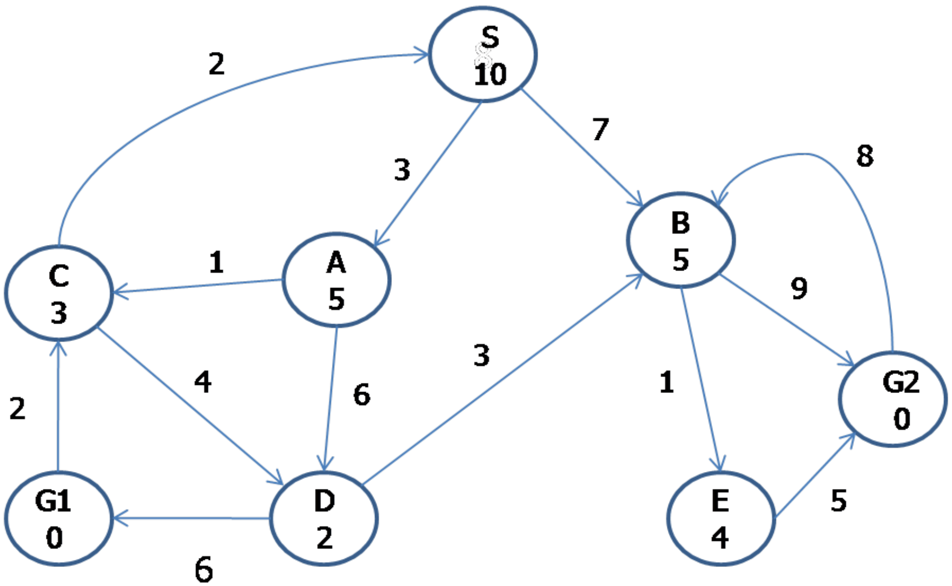


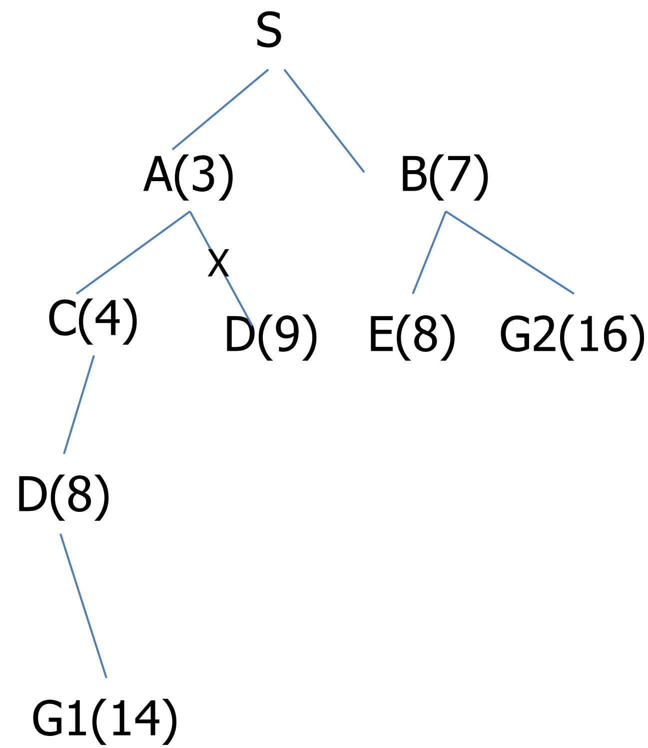


4. Expand B: generate E(8) and G2(16)

Fringe=[D(8), E(8), G2(16)]

Uniform-Cost Graph Search

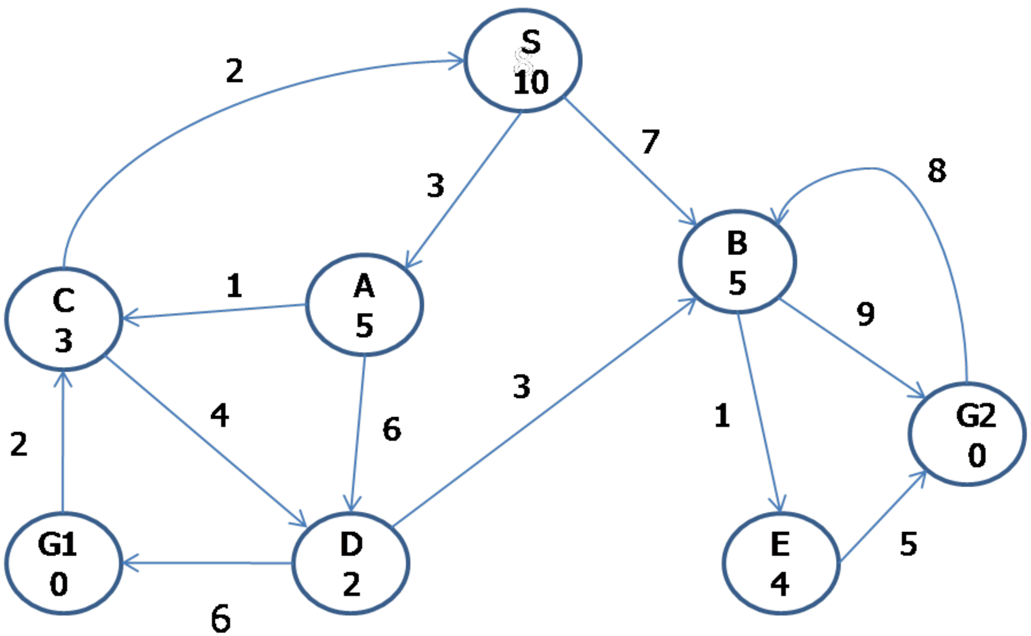


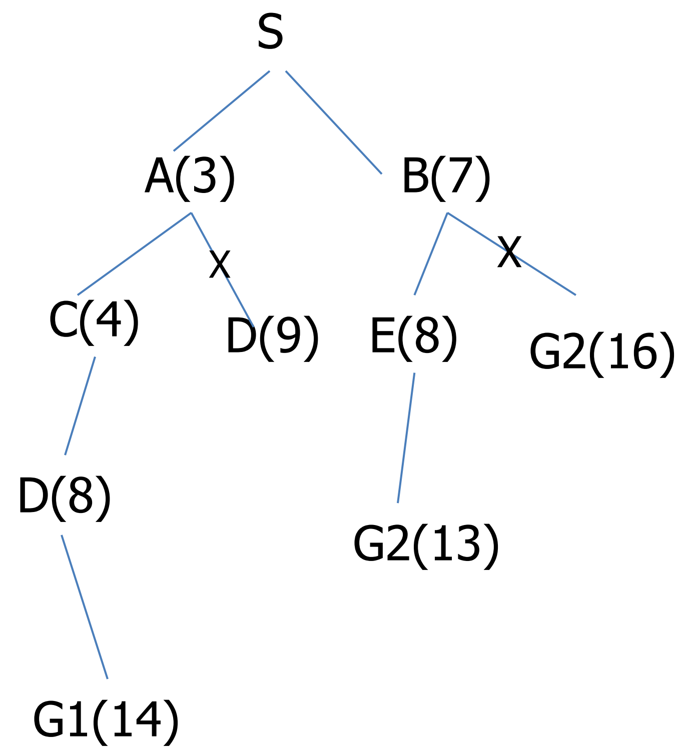


5. Expand D: generate G1(14)

Fringe=[E(8), G1(14), G2(16)]

Uniform-Cost Graph Search





6. Expand E: generate G2(13), remove G2(16) in Fringe

Fringe=[G2(13), G1(14)]

G2(13) is checked as goal, optimal path: S → B → E → G2

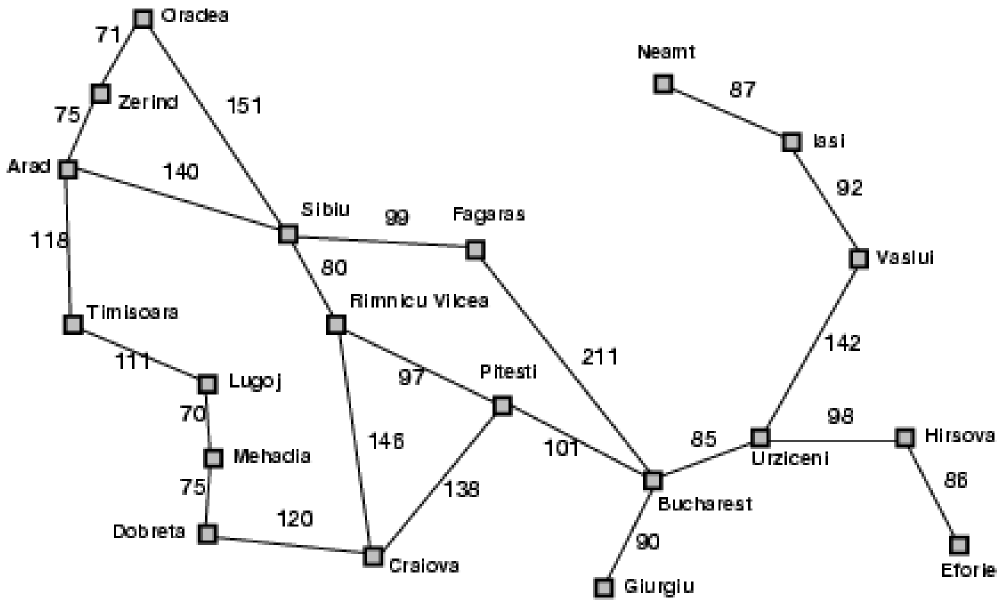
Heuristic (Informed) Search

- The main idea is to utilize heuristic knowledge from the domain in the search procedure
- The heuristic knowledge is embedded into an evaluation function $f(n)$, *which assesses* the "desirability“ of nodes
- The evaluation function is used to choose the most desirable leaf node for expansion
- Order the nodes in fringe in terms of the evaluation function (most desirable node appears the first)
- General approach for informed search is also called best-first search
- Two categories of the best-first search strategy:
  - greedy best-first search
  - A^* search

Greedy best-first search

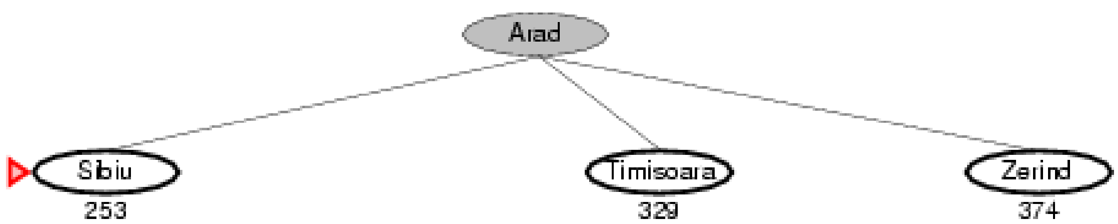
- Evaluation function $f(n) = h(n)$ (heuristic function)
- h(n): estimate of cost of the cheapest path from *n* to *goal*
- e.g., $h(n)$ = straight-line distance from *n* to goal
- Greedy best-first search expands the node that is estimated to be closest to goal

Greedy best-first search example



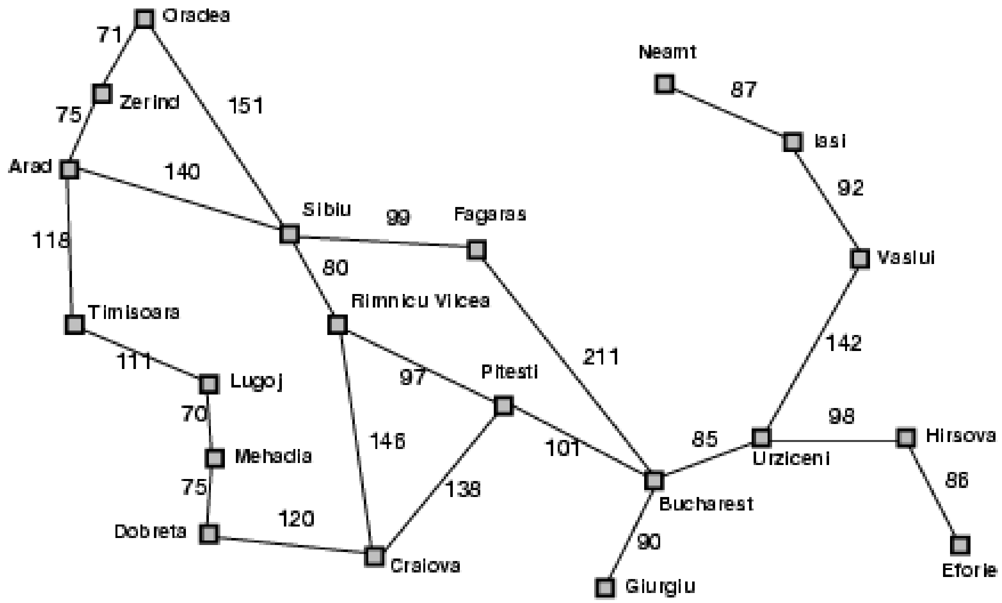
Straight–line distance to Bucharest

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

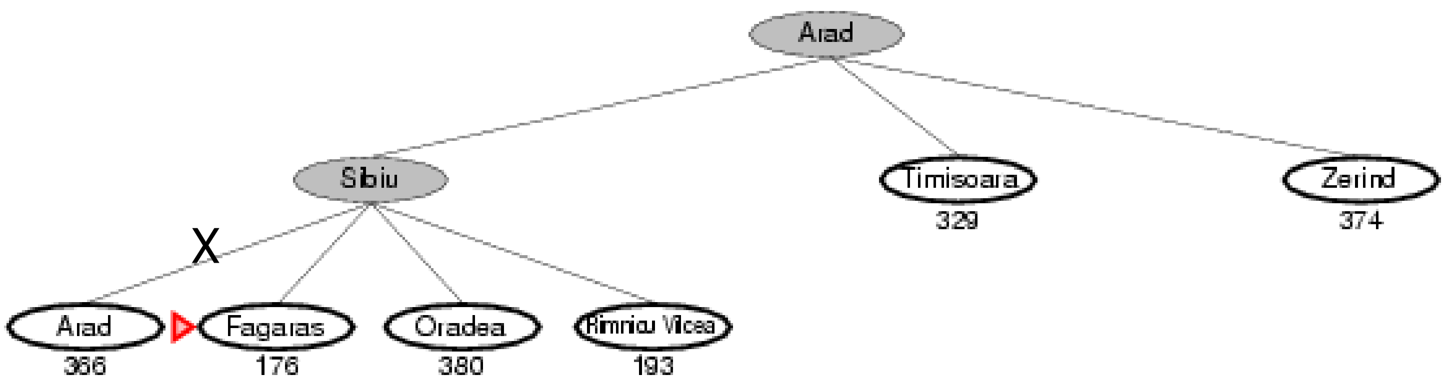


Expand Arad: generate Sibiu(253), Timisoara(329), Zerind(374)

Greedy best-first search example

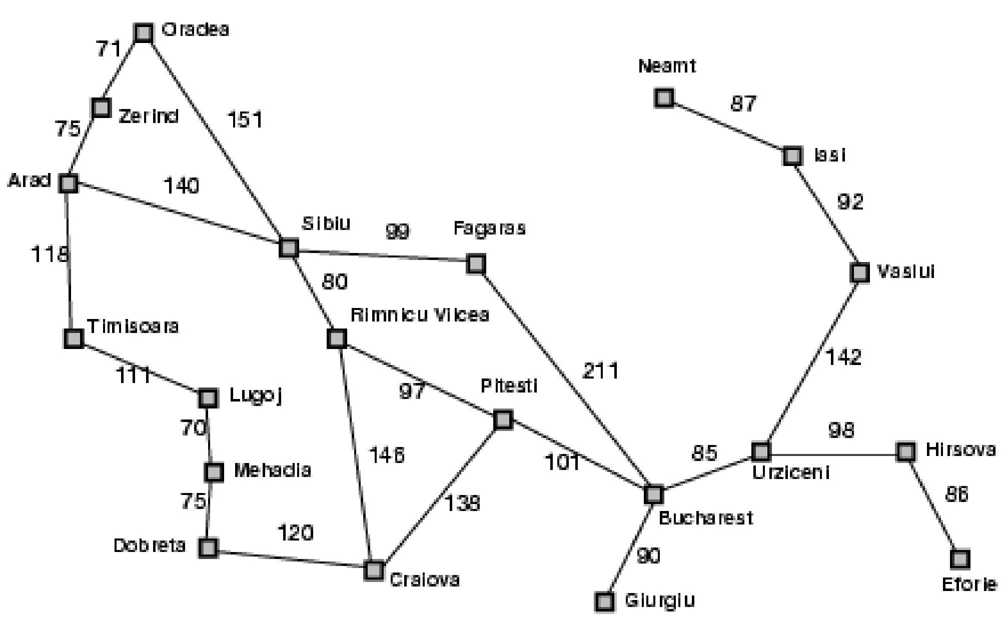


Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

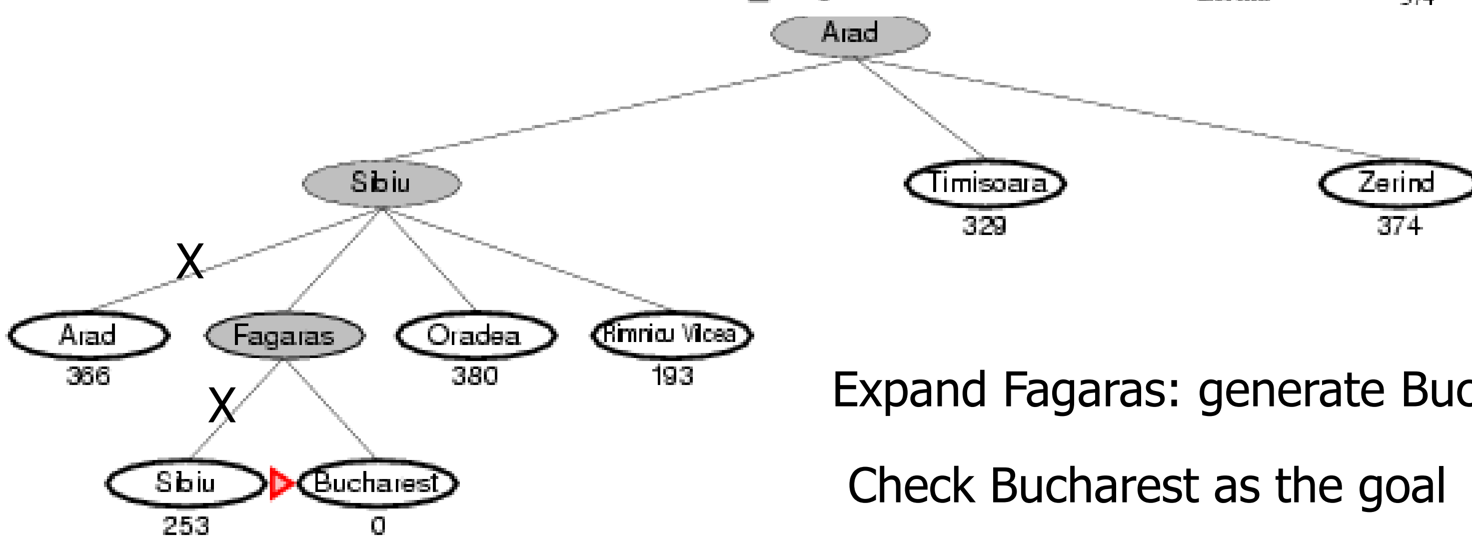


Expand Sibiu: generate Fagaras(176), Oradea(380), Rimnicu Vilcea(193)

Greedy best-first search example



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



Expand Fagaras: generate Bucharest(0)

Check Bucharest as the goal

A^* search

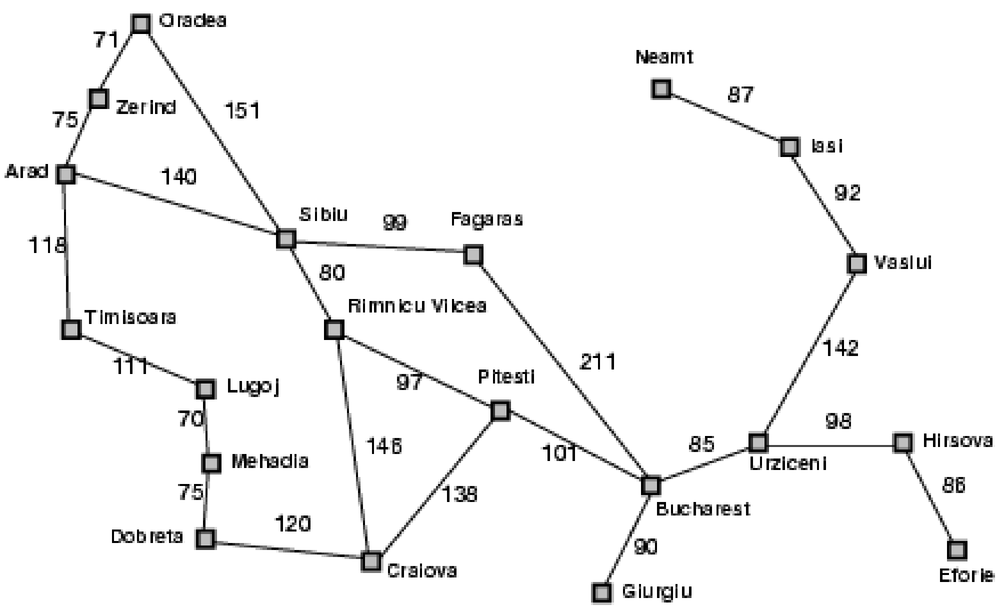
- Purpose: avoid expanding paths that are already expensive, combine known cost and predicted cost
- Evaluation function $f(n) = g(n) + h(n)$

  -- $g(n)$ = cost so far to reach n

  -- $h(n)$ = estimated cost of the cheapest path from n to goal
- Order nodes in Fringe according the values of the f function.

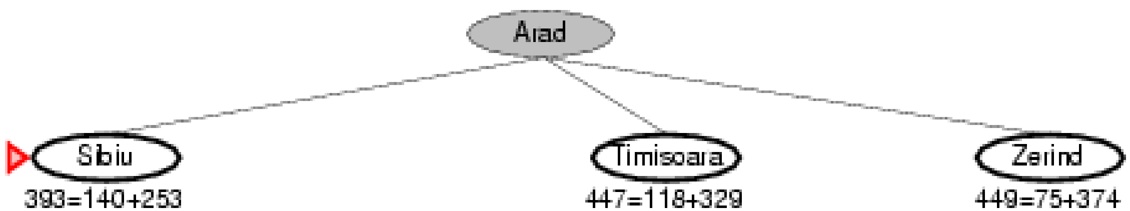
- When h(n)=0, A^* search turns to be uniform cost search

A* search example

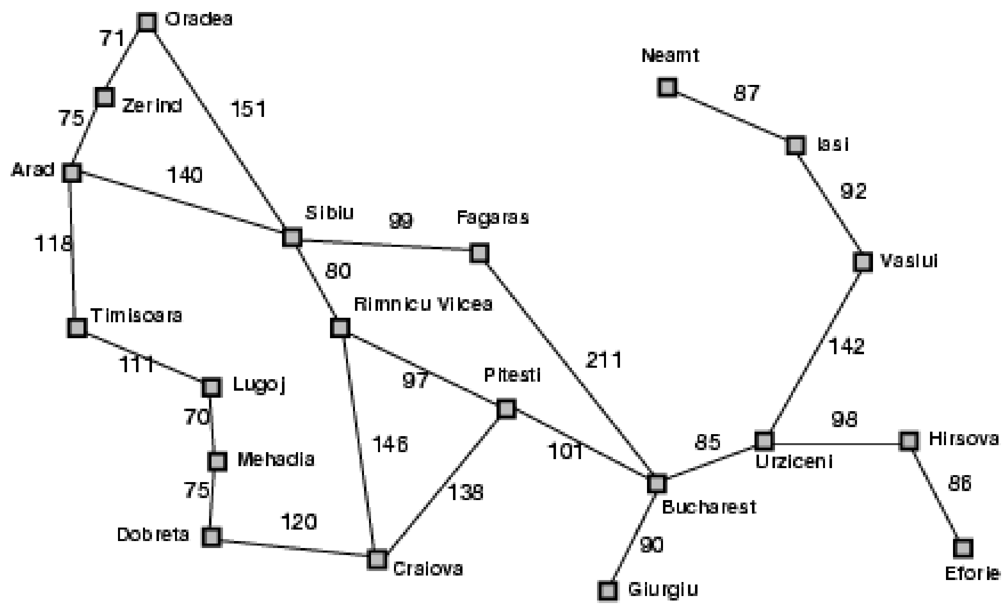


Straight–line distance to Bucharest

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

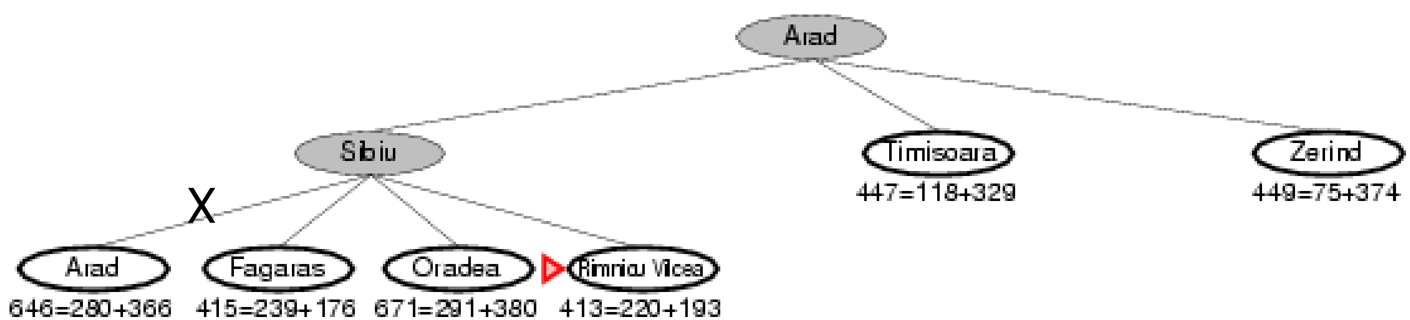


A* search example

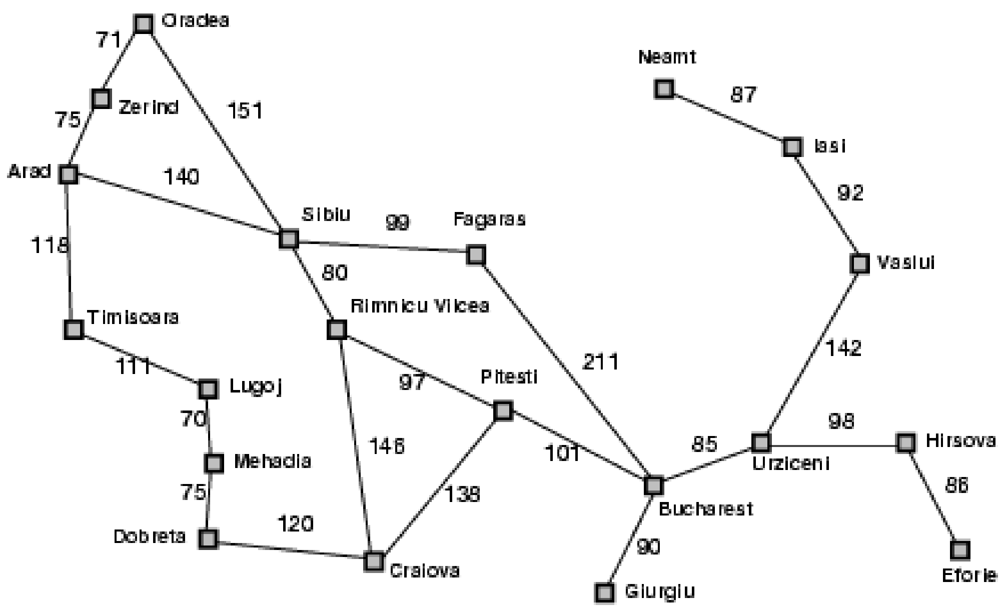


Straight–line distance to Bucharest

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

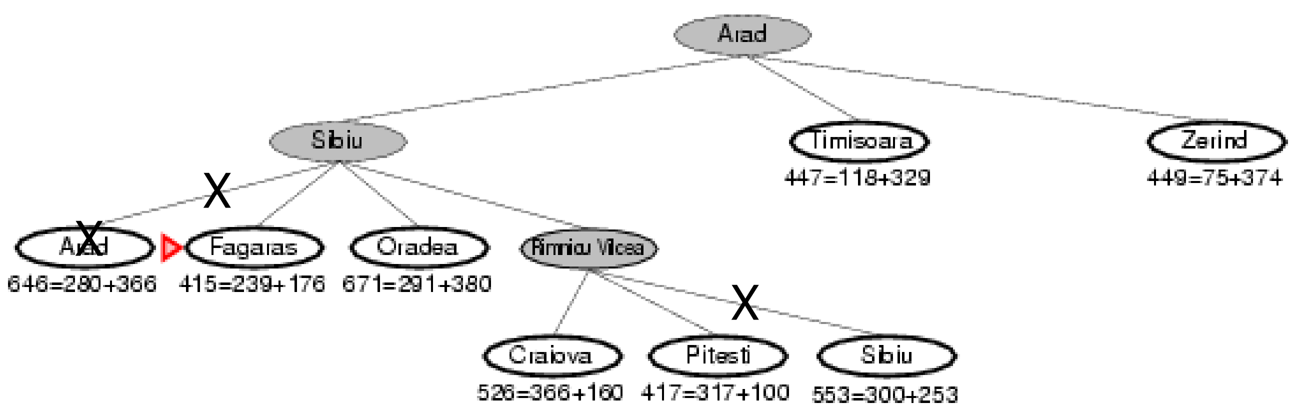


A* search example

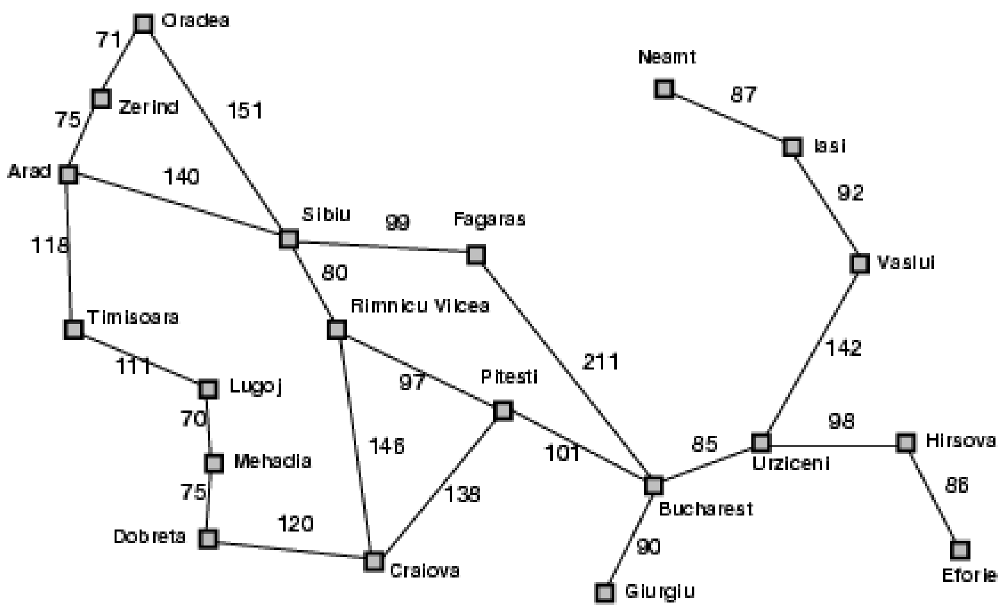


Straight–line distance to Bucharest

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

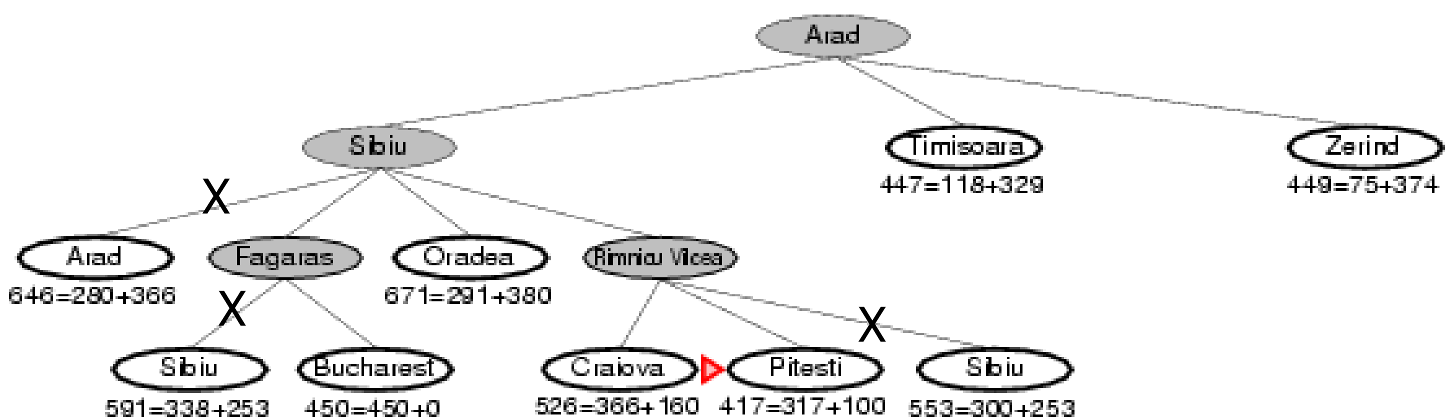


A* search example

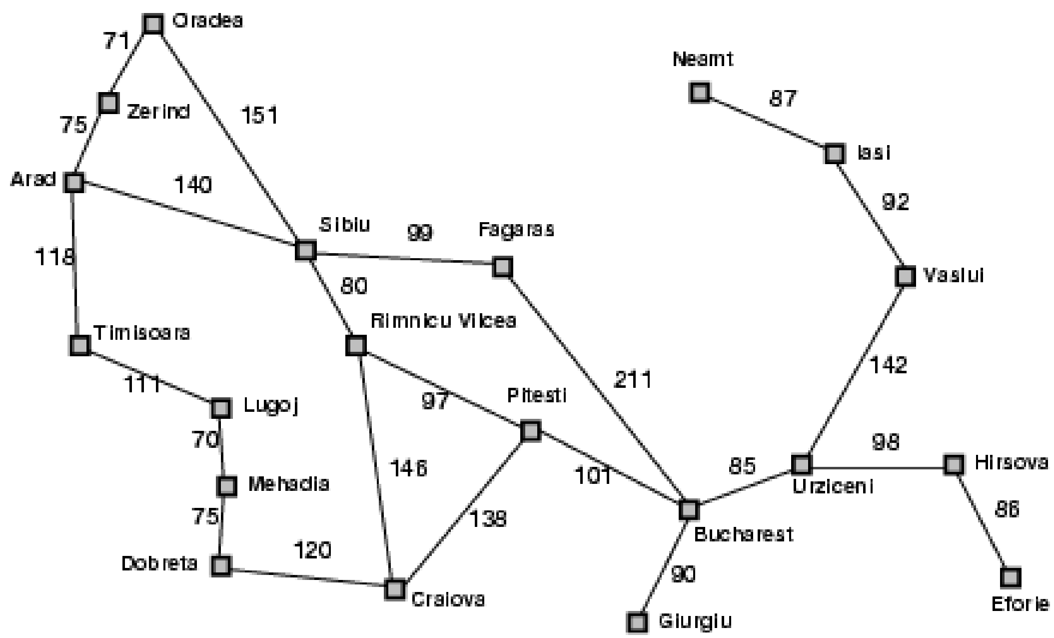


Straight–line distance to Bucharest

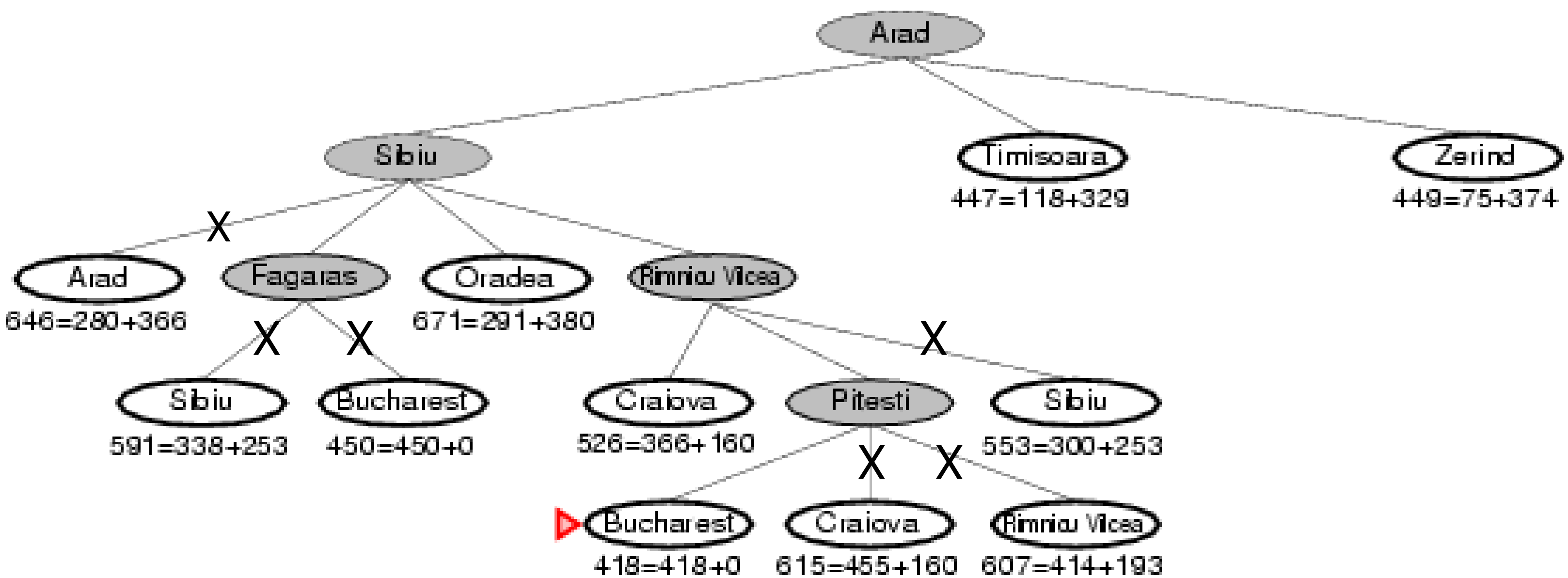
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374



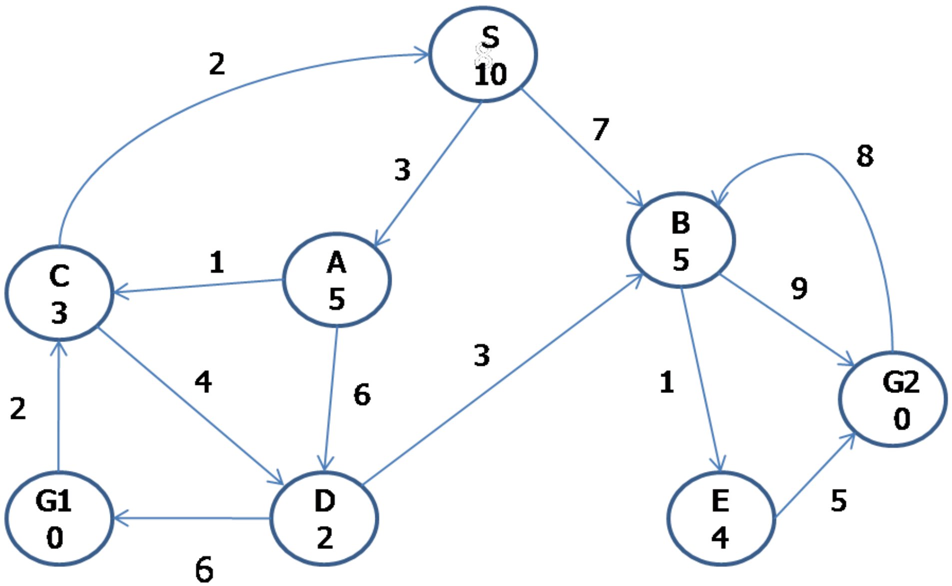
A* search example

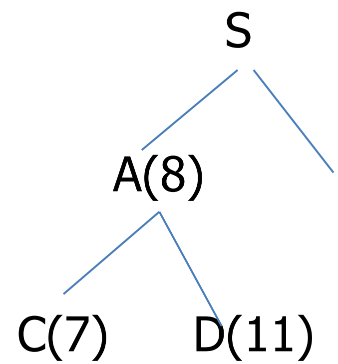


Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374



Another Example: A* Search





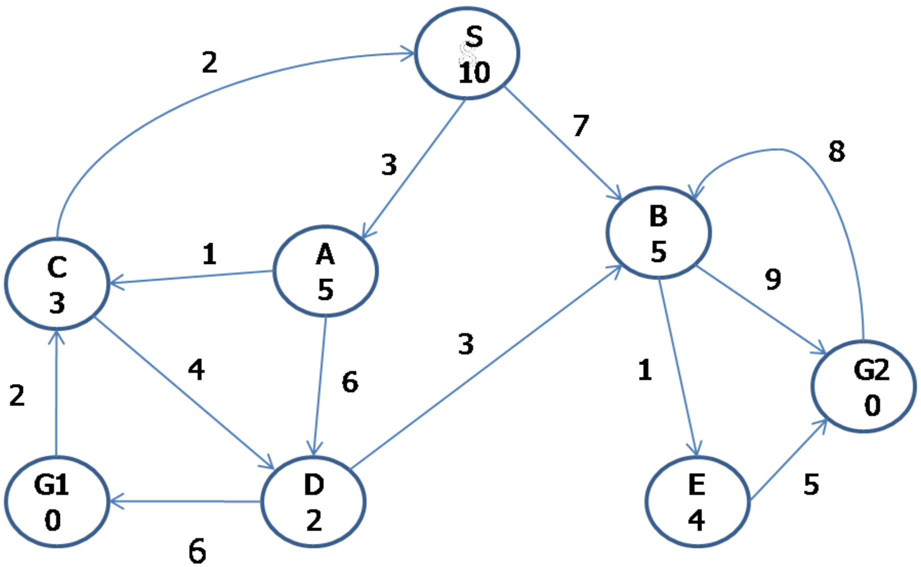
1. Expand S: generate A(8) and B(12)
2. Expand A: generate C(7) and D(11)

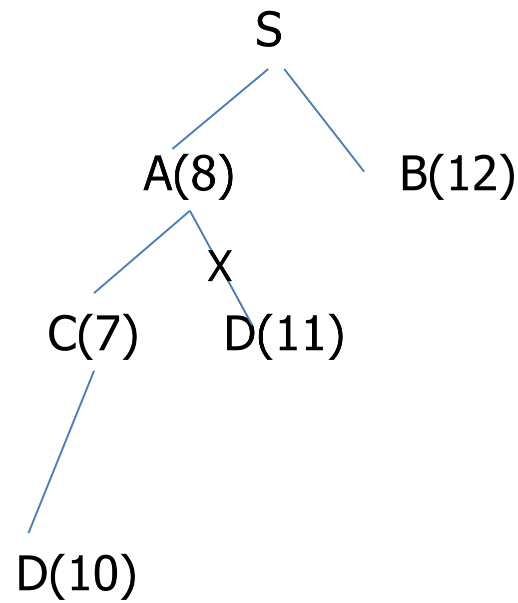
h(A)=5, h(B)=5, h(C)=3, h(D)=2, h(E)=4

h(G1)=h(G2)=0

Fringe=[C(7), D(11), B(12)]

Another Example: A* Search

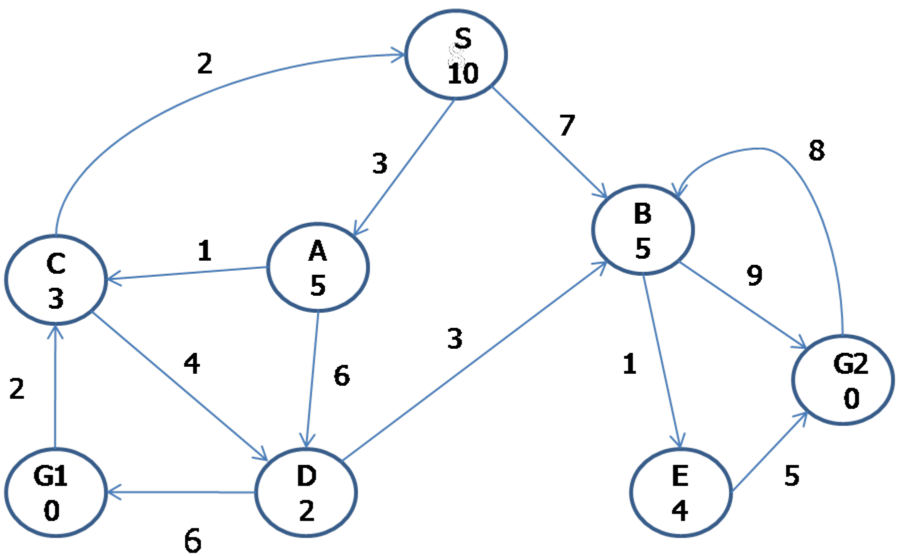


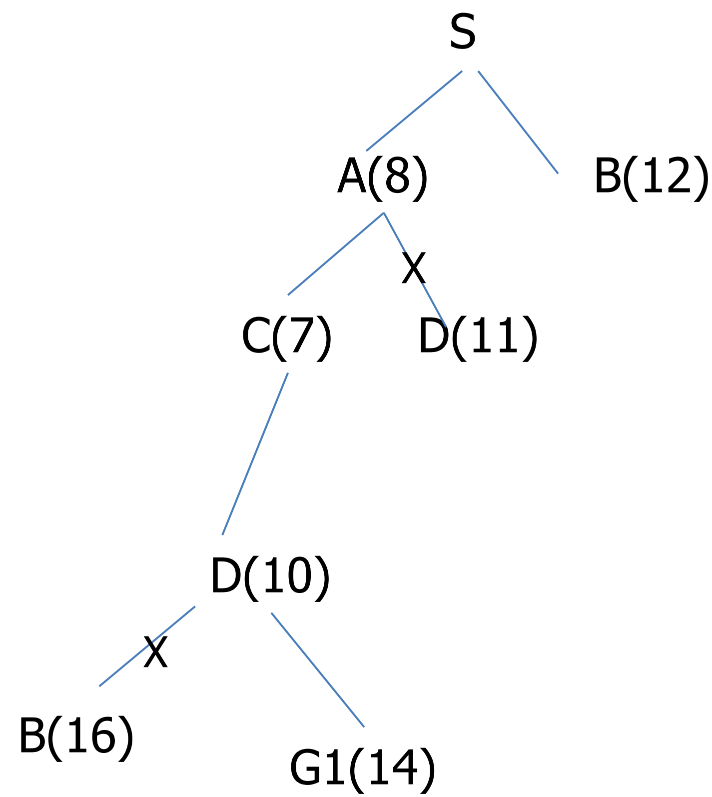


3. Expand C: generate D(10), remove D(11) from Fringe

Fringe=[D(10), B(12)]

Another Example: A* Search

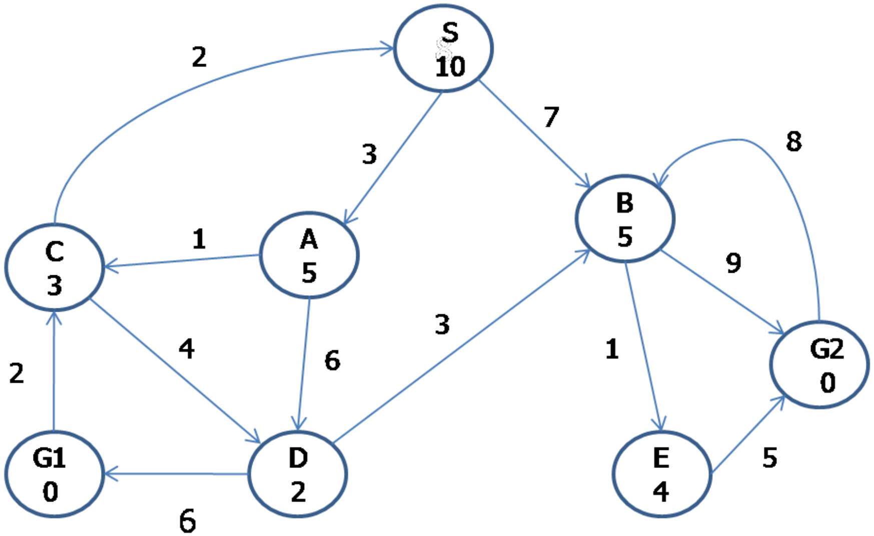


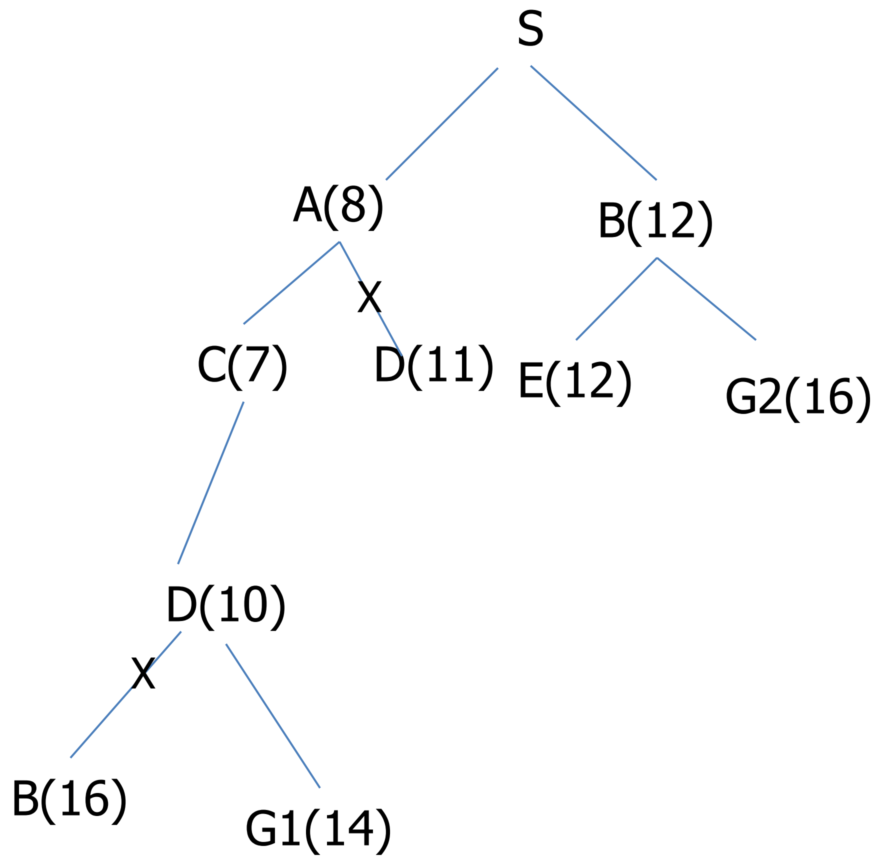


4. Expand D: generate G1(14) and B(16), B(16) is then discarded

Fringe=[B(12), G1(14)]

Another Example: A* Search

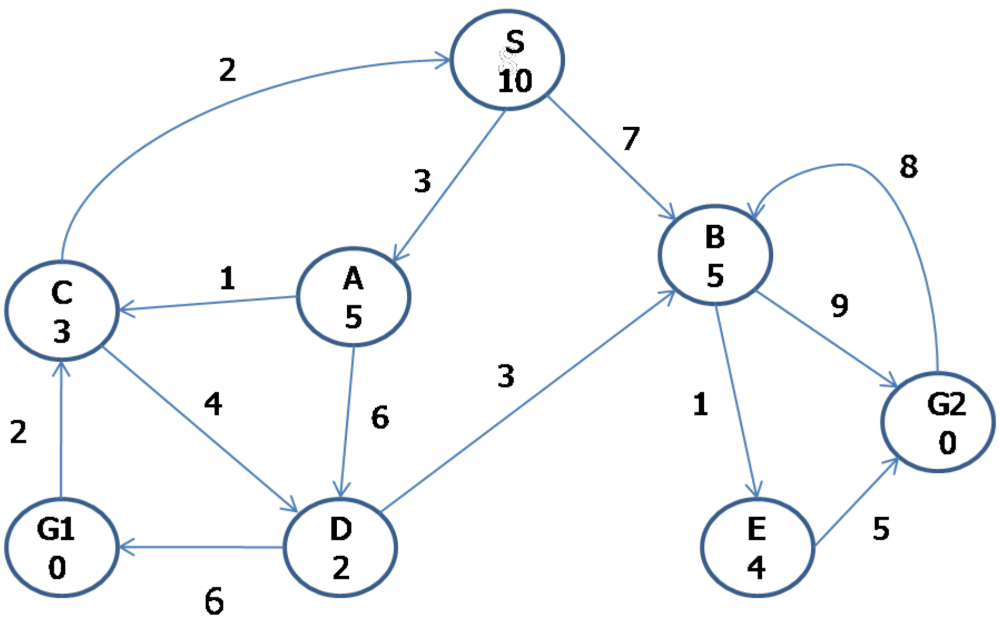


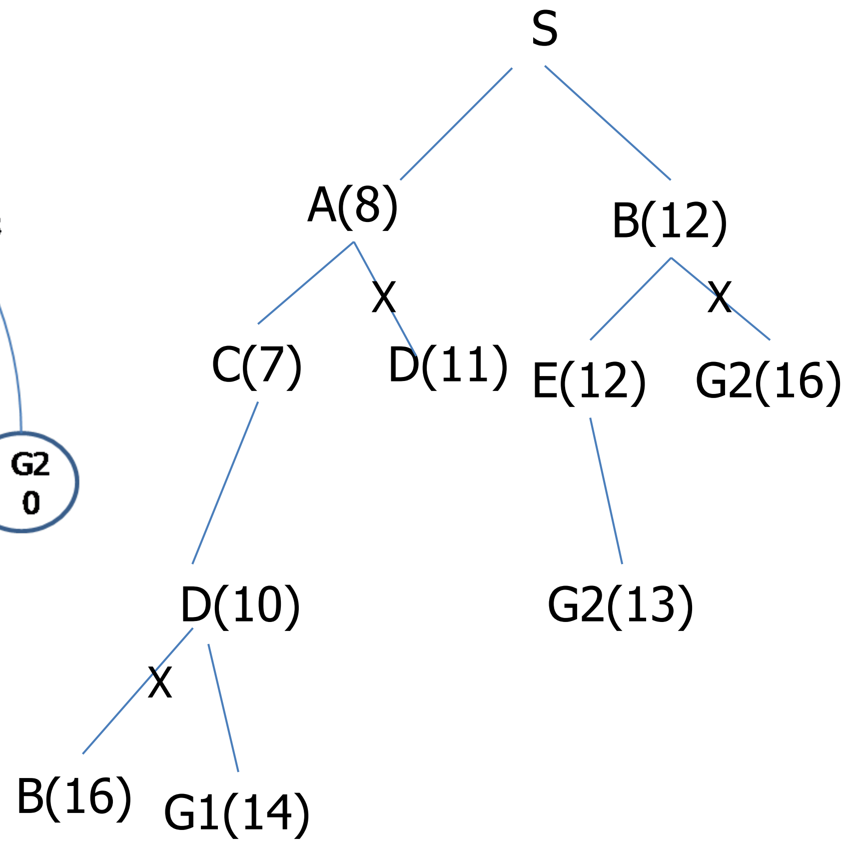


5. Expand B: generate E(12) and G2(16)

Fringe=[E(12), G1(14), G2(16)]

Another Example: A* Search





6. Expand E: generate G2(13), remove G2(16) in Fringe

Fringe=[G2(13), G1(14)]

Recognize G2 as the goal, Path: S → B → E → G2

Optimality of A*?

- Whether an A* algorithm is optimal **depends on**

the used heuristic function h.

- Some specific property of h is required to ensure optimality

- We will discuss such property for tree and graph searches respectively

When Is A* Optimal on Trees

- Theorem 1: If *h(n)* is admissible, A^* using `TREE-SEARCH` is optimal

- A heuristic *h(n)* is admissible if for every node *n*, $h(n) \leq h^*(n)$, where $h^*(n)$ is the true least cost to reach the goal state from *n*.

- An admissible heuristic never overestimates the least cost to reach the goal.

Proof of Optimality of A* on trees?

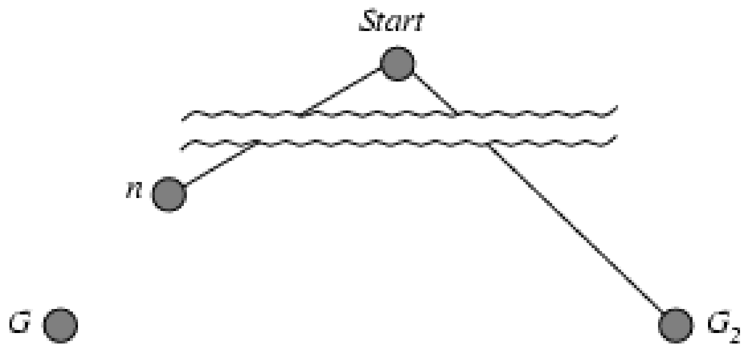
Suppose a goal state G appears as the first element in the fringe, the search will terminate and return the path from the initial to G .

We only need to prove that G is the optimal goal state

Proof of Optimality on Trees with admissible heuristics

Prove: **a suboptimal goal G2 will never appear first in Fringe**

- Suppose some suboptimal goal G_2 has been generated and is in the fringe. Let *n* be an unexpanded node in the fringe such that *n* is on a shortest path to an optimal goal *G*.
-



We have:

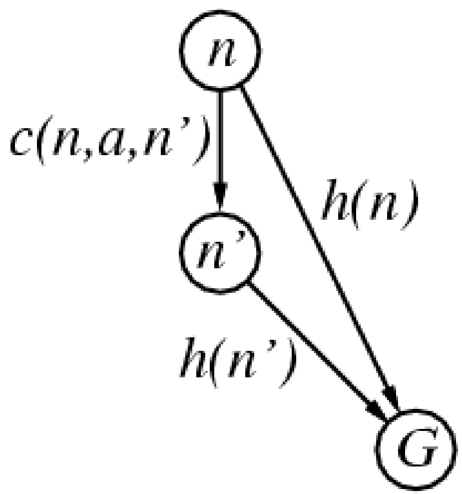
1. $f(G_2)=g(G_2)>g(G)$
2. $g(G)=g(n)+h^*(n)>g(n)+h(n)=f(n)$

   Hence we have $f(G_2)>f(n)$

When is A* Optimal on Graphs

- Theorem 2: If *h(n)* is consistent, A* using `GRAPH-SEARCH` is optimal

- A heuristic is consistent if for every node *n*, every successor *n'* of *n* generated by any action *a*, $h(n) \leq c(n,a,n') + h(n')$



Proof of Optimality for A* on graphs?

Suppose a goal state G appears as the first element in the fringe, the search will terminate and return the path from the initial to G .

We have to prove:

1. G is optimal goal state

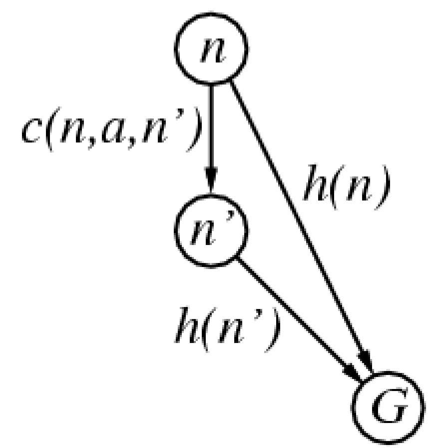
Since consistence ensures admissibility, this can be proved in the same way as for A* tree search

2. It is not possible to find another path to G yielding lower cost?

Proof of optimality on graphs with consistent heuristics

To Prove: **any other path to G will have more cost**

Suppose *n* is a node in the Fringe and appears on another path to G, and n' is the successor of *n* on the path, we have



$$
\begin{aligned}
f(n') &= g(n') + h(n') \\
&= g(n) + c(n,a,n') + h(n') \\
&\geq g(n) + h(n) \\
&= f(n)
\end{aligned}
$$

Therefore *f(n)* is non-decreasing along the path

$f(n) > f(G) = g(G)$

$f(G) = g'(G) \geq f(n)$

$\Longrightarrow$ $g'(G) > g(G)$