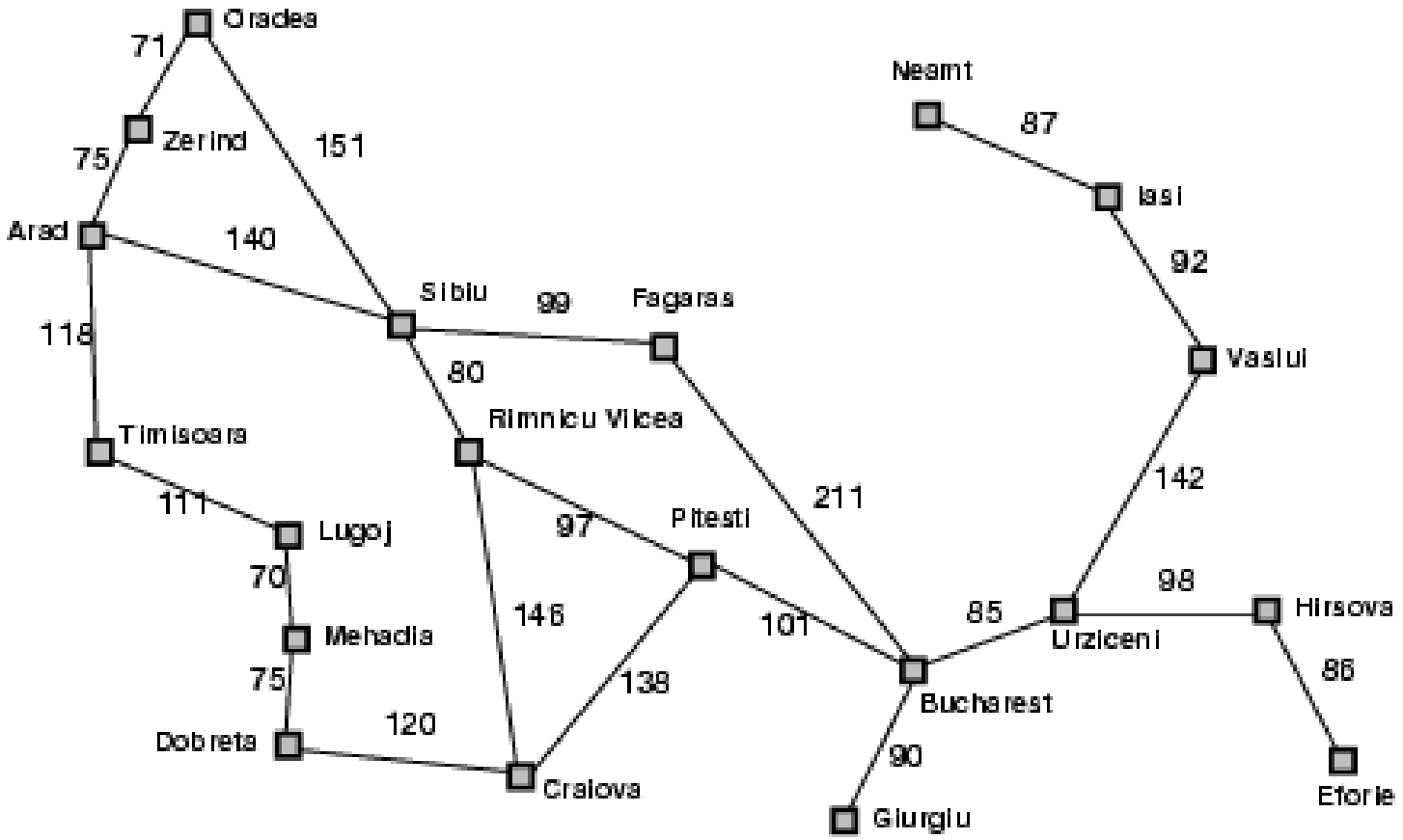
Example

A* Algorithm

- An example presented here to enable your reading the textbook

Romania with step costs in km

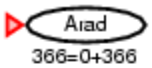


Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Open List:

Arad

A* search example



We start with our initial state Arad. We take a node and add it to the open list. Since it's the only thing on open list, we expand the node.

Open list is sorts the nodes inside of it according to their g()+h() score.

Open List:

Sibiu

Timisoara

Zerind

A* search example

We add the three nodes we found to the open list.

We sort them according to the g()+h() calculation.

A* search example

Open List:

Rimricu Vicea

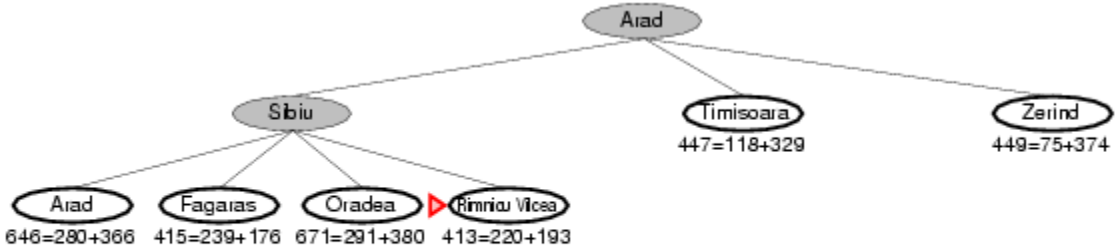
Fagaras

Timisoara

Zerind

~~Arad~~

Oradea



We've been to Arad before. Don't list it again on the open list.

When we expand Sibiu, we run into Arad again. But we've already expanded this node once; so, we don't add it to the open list again.

Open List:

Rimricu Vicea

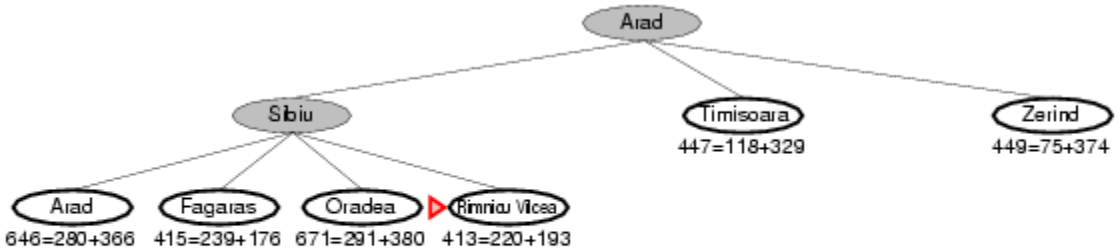
Fagaras

Timisoara

Zerind

Oradea

A* search example



We see that Rimricu Vicea is at the top of the open list; so, it's the next node we will expand.

Open List:

Fagaras

Pitesti

Timisoara

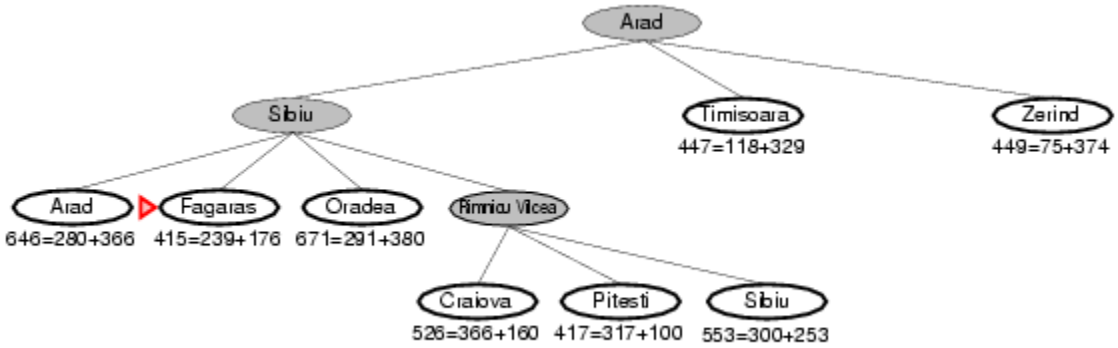
Zerind

Craiova

~~Sibiu~~

Oradea

A* search example



When we expand Rimricu Vincea, we run into Sibiu again.

But we've already expanded this node once; so, we don't add it to the open list again.

Open List:

Fagaras

Pitesti

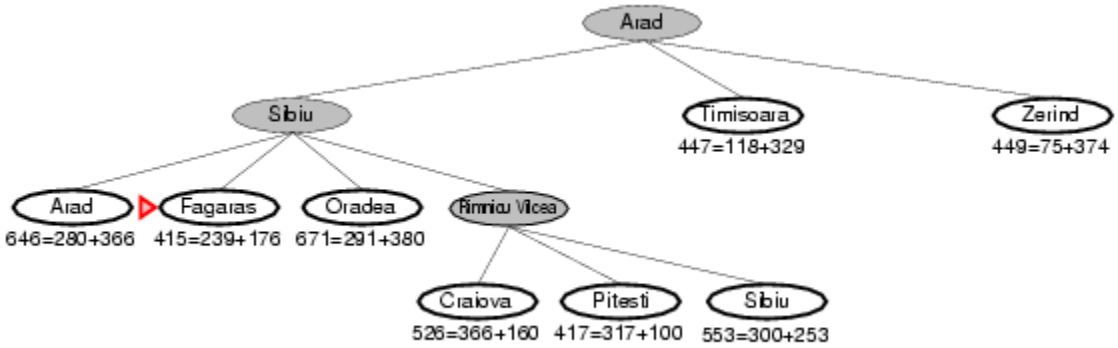
Timisoara

Zerind

Craiova

Oradea

A* search example



Fagaras will be the next node we should expand – it's at the top of the sorted open list.

Open List:

Pitesti

Timisoara

Zerind

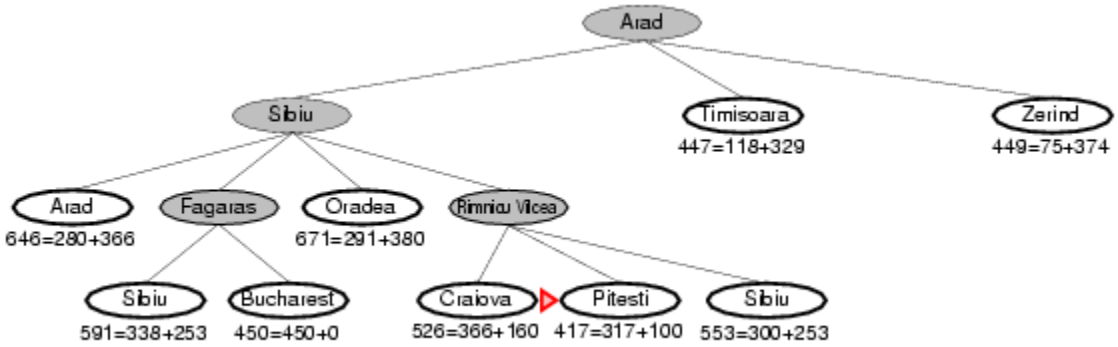
Bucharest

Craiova

~~Sibiu~~

Oradea

A* search example



When we expand Fagaras, we find Sibiu again. We don't add it to the open list.

We also find Bucharest, but we're not done. The algorithm doesn't end until we "expand" the goal node – it has to be at the top of the open list.

Open List:

Pitesti

Timisoara

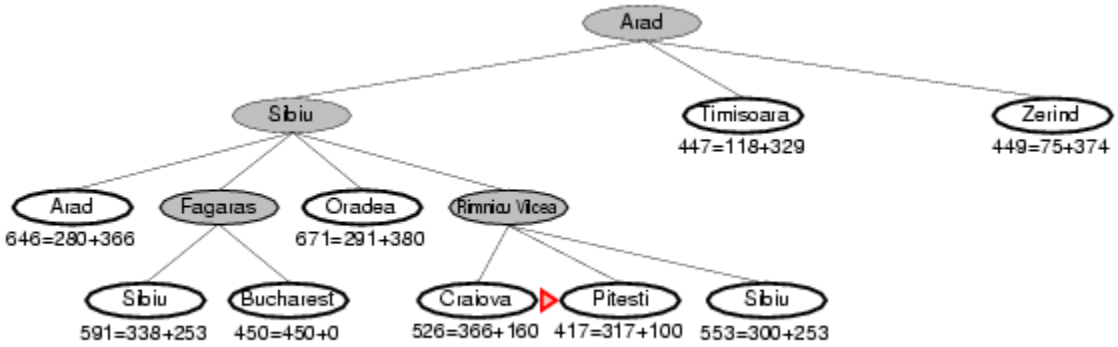
Zerind

Bucharest

Craiova

Oradea

A* search example



It looks like Pitesti is the next node we should expand.

Open List:

Bucharest

Timisoara

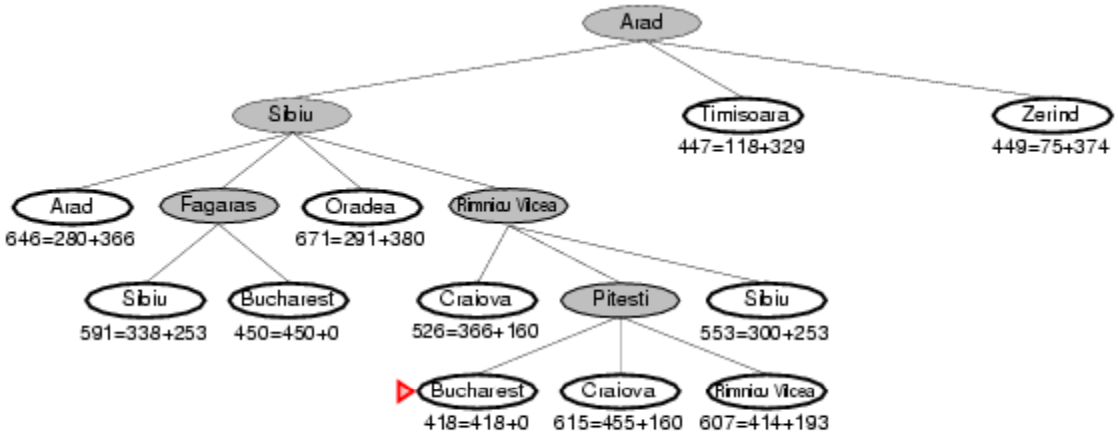
Zerind

Craiova

~~Rimricu Vicea~~

Oradea

A* search example



We just found a better value for Bucharest; so, it got moved higher in the list. We also found a worse value for Craiova – we just ignore this.

And of course, we ran into Rimricu Vicea again. Since it's already been expanded once, we don't re-add it to the Open List.

Open List:

Bucharest

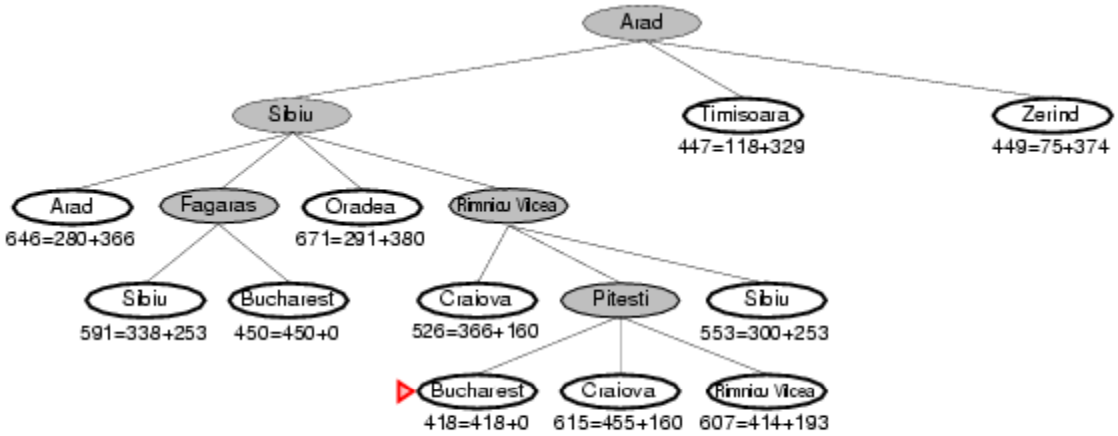
Timisoara

Zerind

Craiova

Oradea

A* search example



Now it looks like Bucharest is at the top of the open list…

Open List:

Bucharest

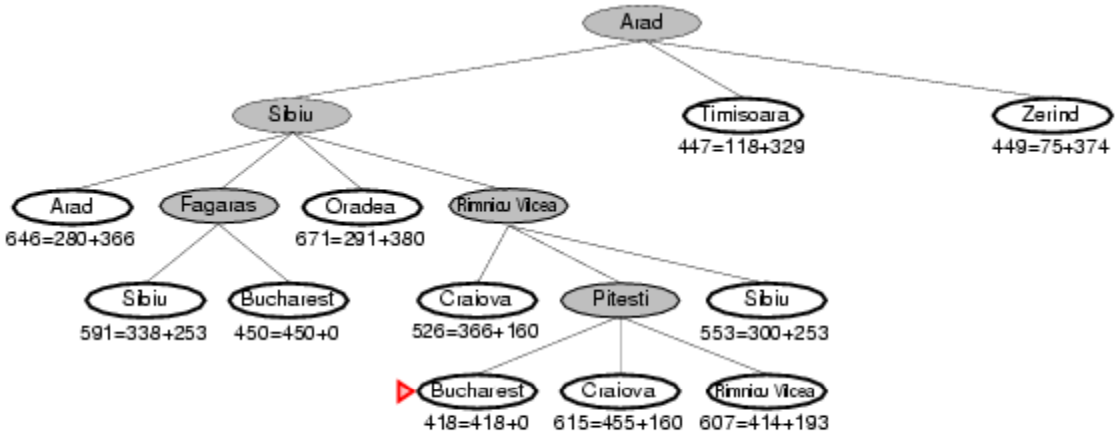
Timisoara

Zerind

Craiova

Oradea

A* search example



Now we “expand” the node for Bucharest.

We’re done! (And we know the path that we’ve found is optimal.)

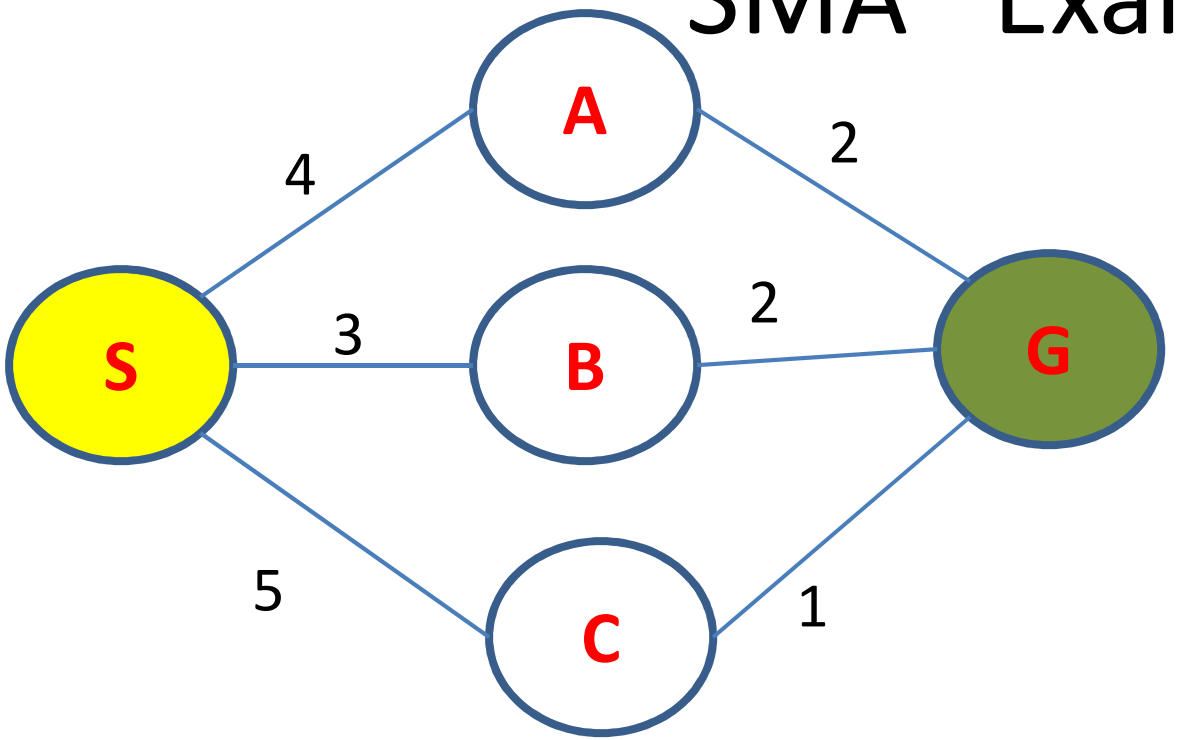
Memory bounded heuristics

- SMA* example presented here to help you understand the material better

Memory bounded heuristic search

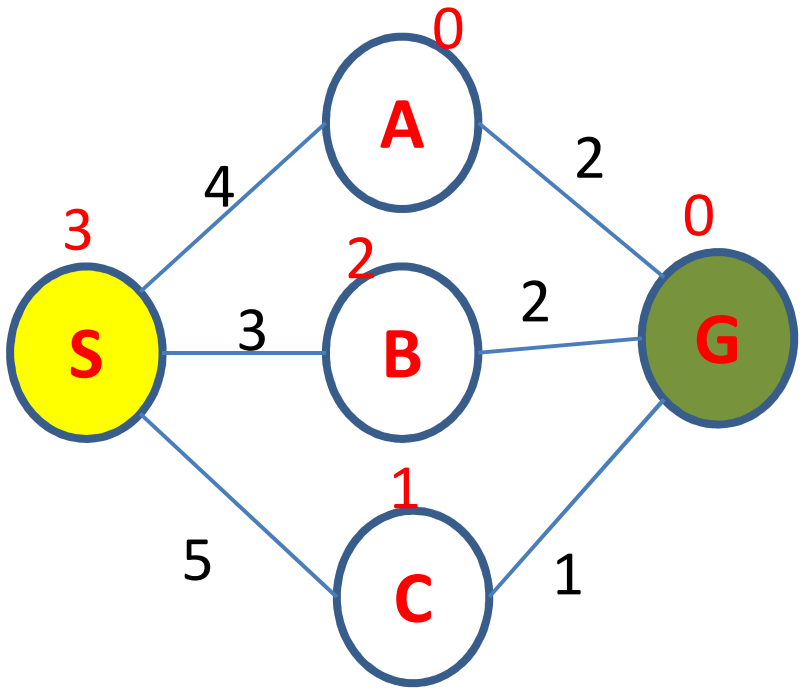
- **Iterative deepening A* (IDA*)**
  - Similar to iterative deepening algorithm
  - **f-cost (g+h)** used as cutoff rather than depth
  - At each iteration, *cutoff value is smallest f-cost of any node that exceeded the cutoff on previous iteration*
- **Recursive best-first search (RBFS)**
  - Similar to recursive depth first search in structure
  - Uses f-limit variables to keep track of f-values
  - Is an optimal algorithm if h(n) is admissible
  - Space complexity linear in the depth of the deepest optimal solution
- IDA* and RBFS suffer from using **too little** memory
- **How to use all memory available ?**

SMA* Example



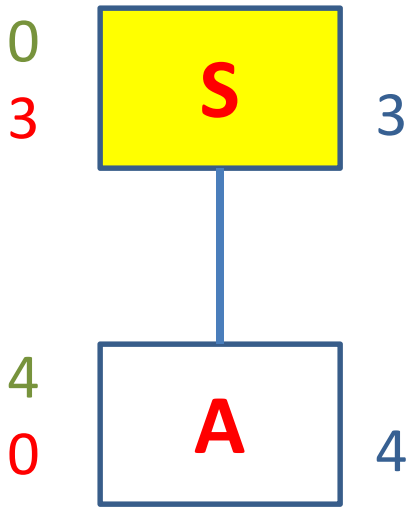
	S	A	B	C	G
Heuristic	3	0	2	1	0

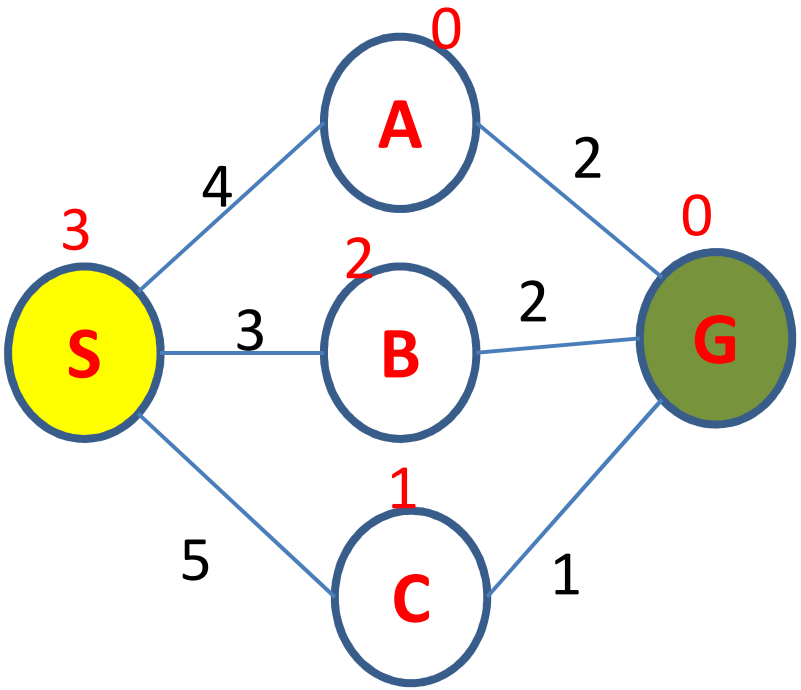
SMA* Example



Perform SMA* with memory = 3 nodes

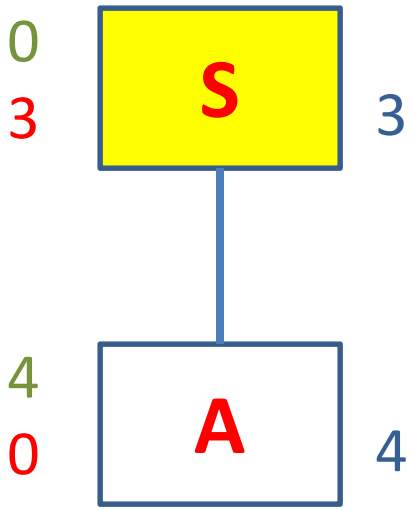
SMA* Example

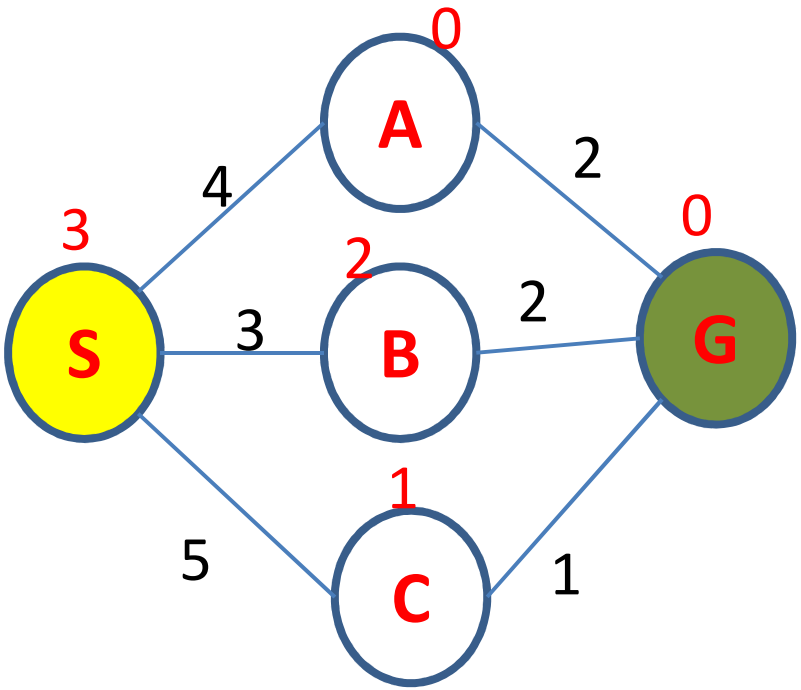




Generate children

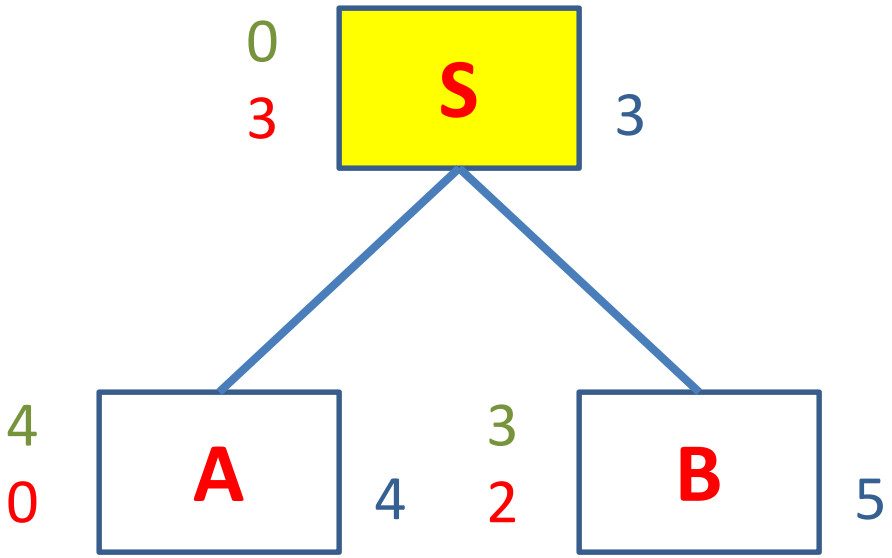
SMA* Example

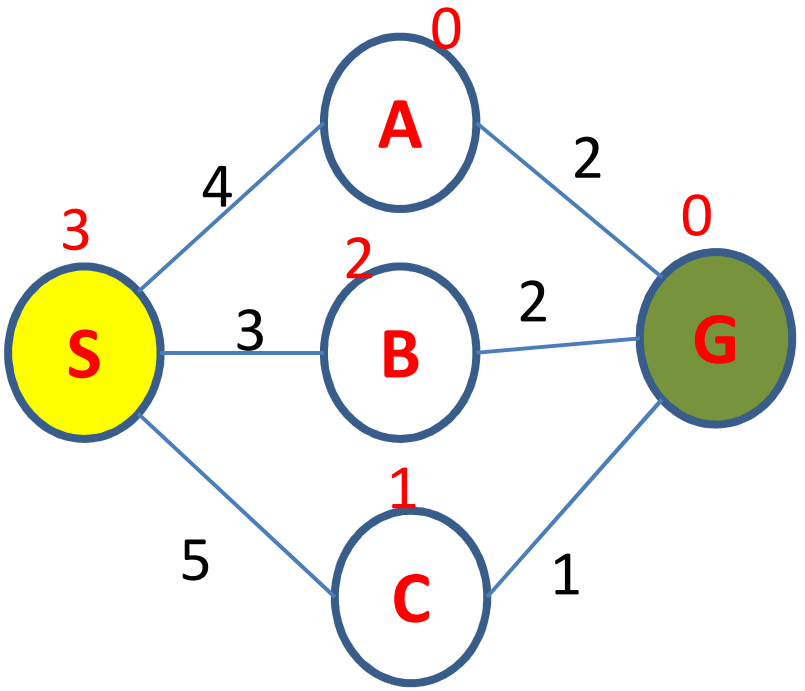




Generate children

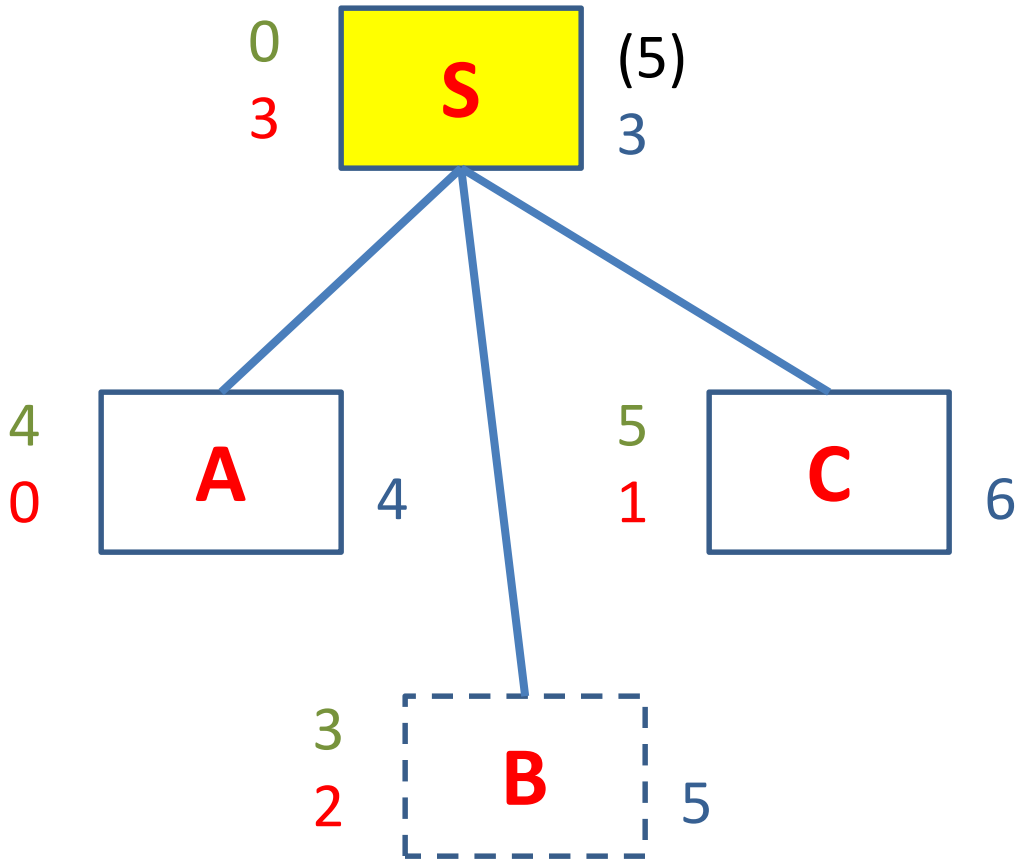
SMA* Example

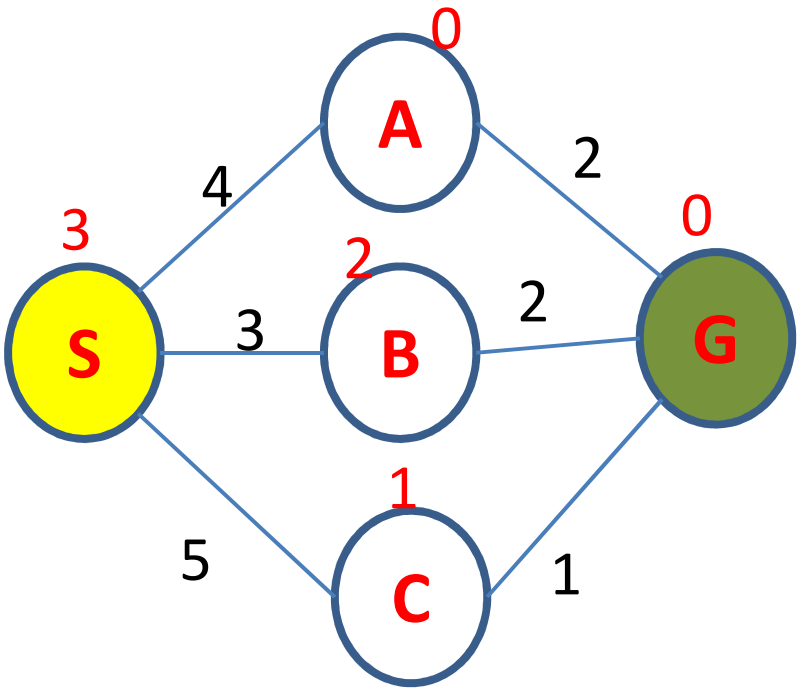




Generate children

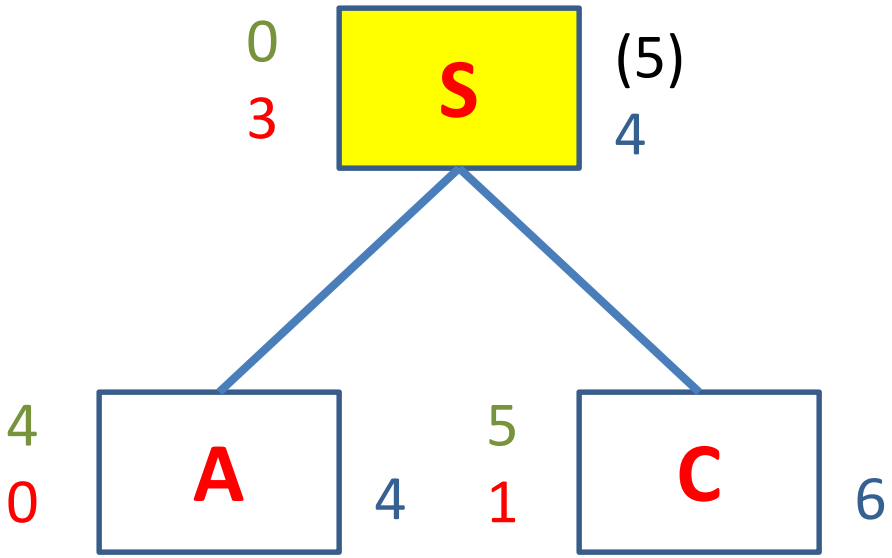
SMA* Example

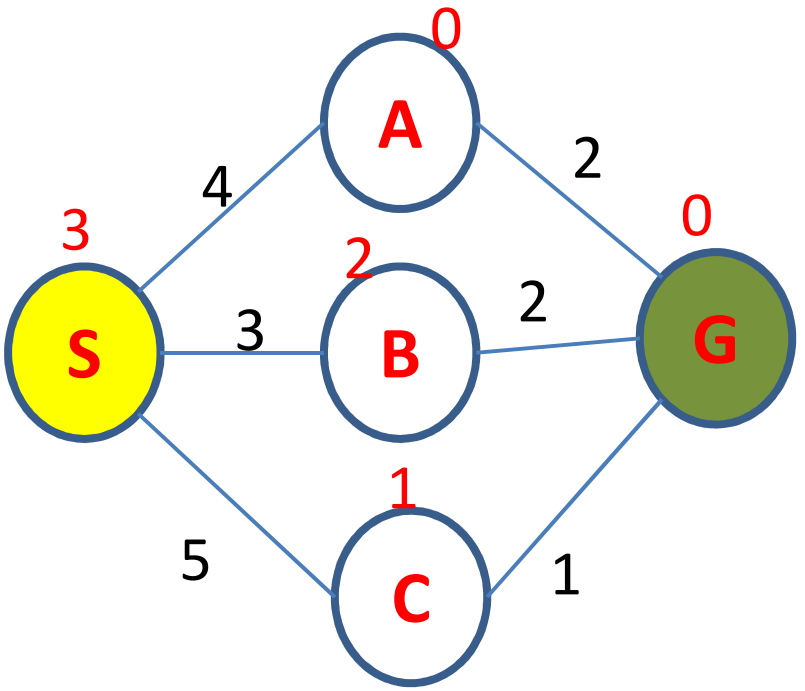




Memory Full
Forget B

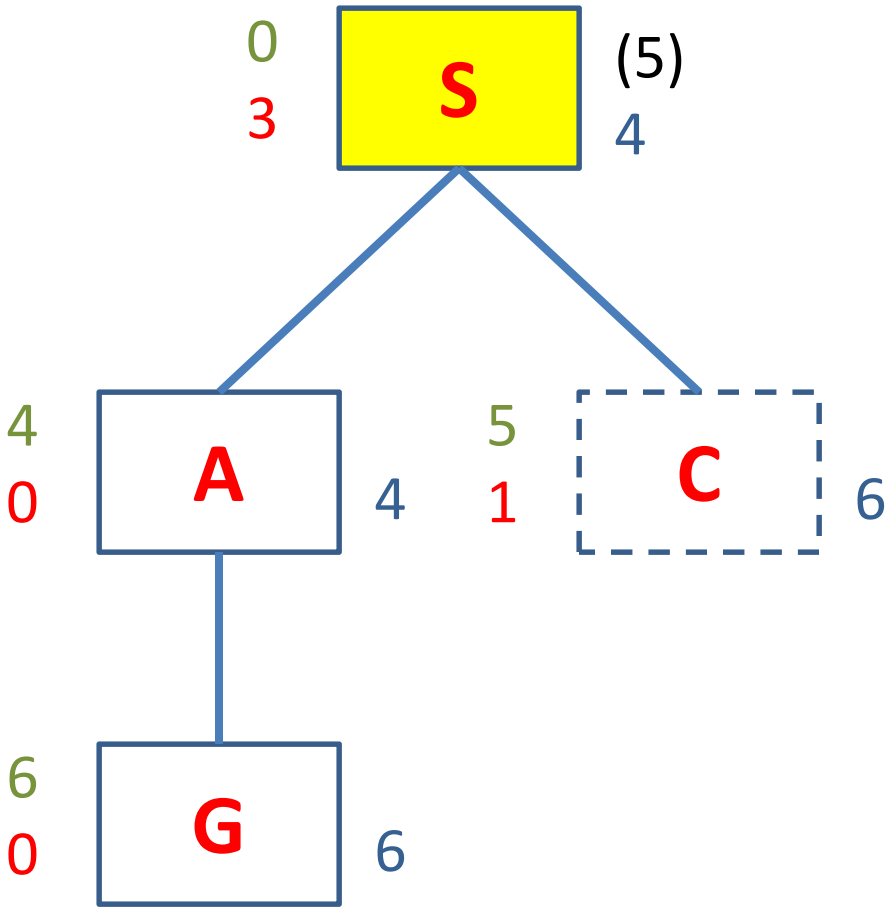
SMA* Example

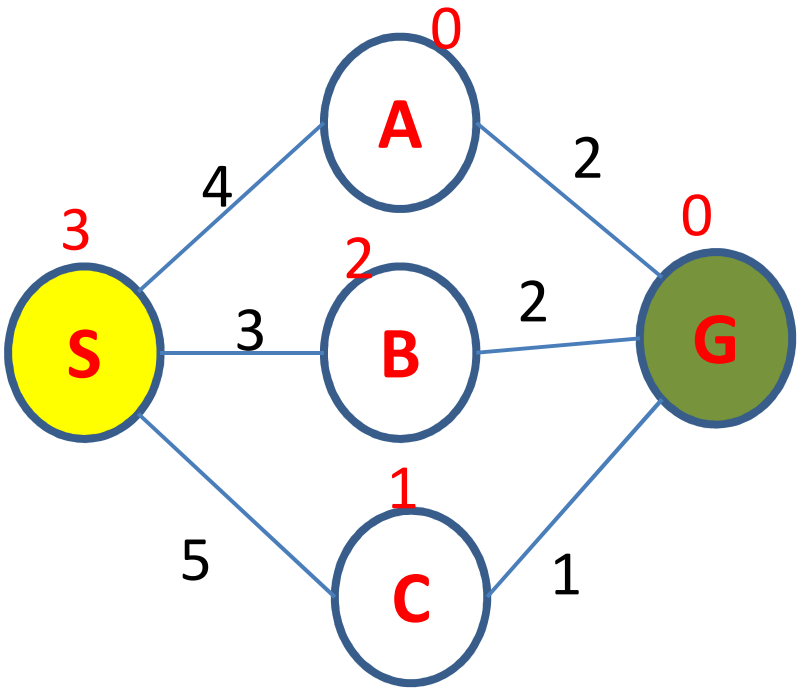




All children explored
Adjust f-values

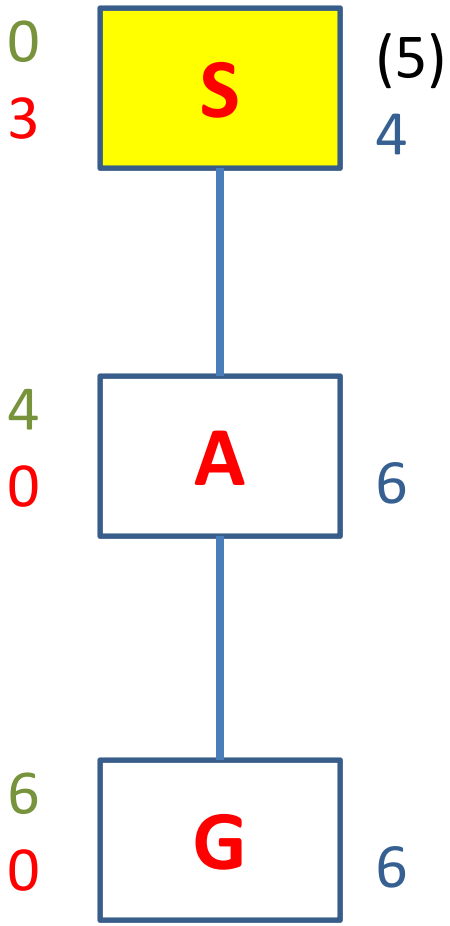
SMA* Example

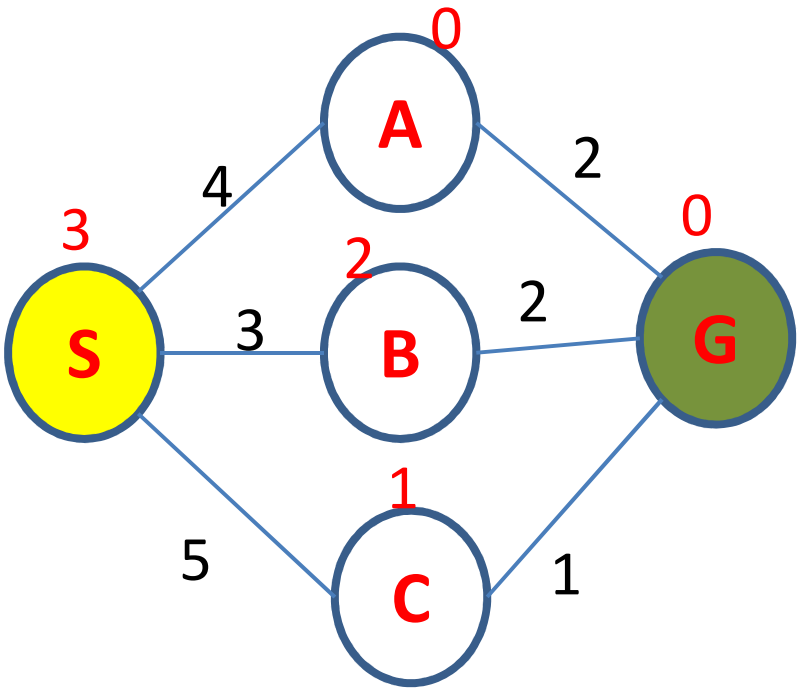




Memory Full
Remove C

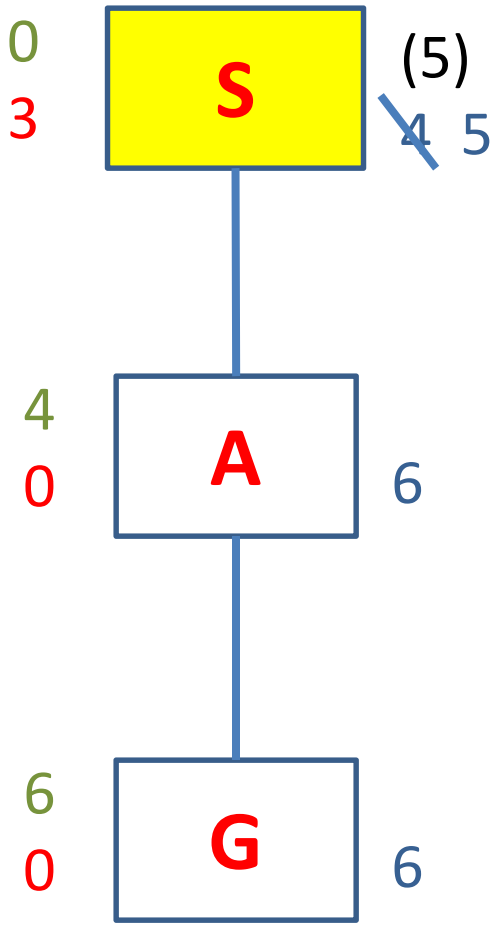
SMA* Example

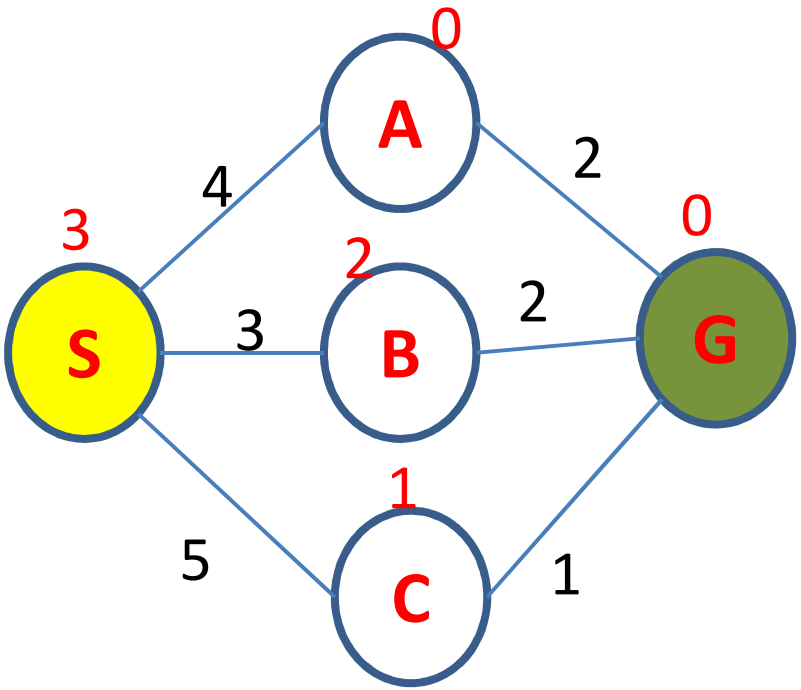




All children explored
Adjust f-values

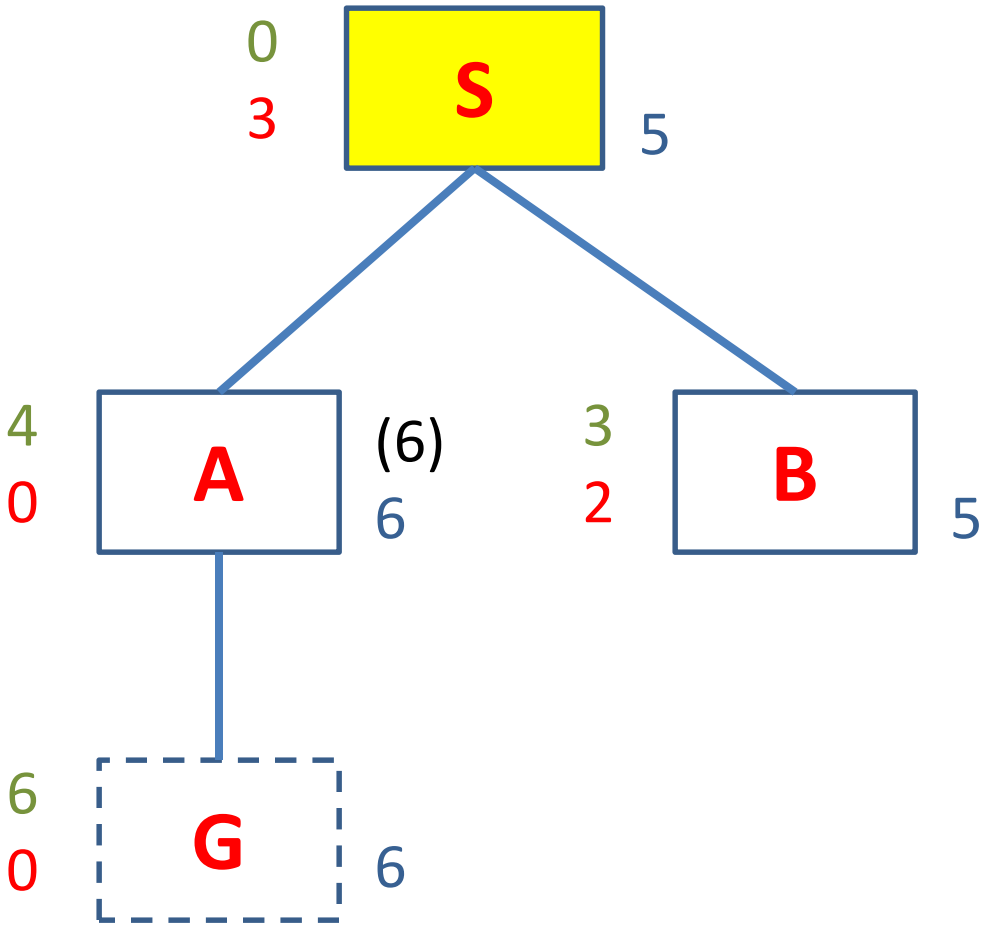
SMA* Example

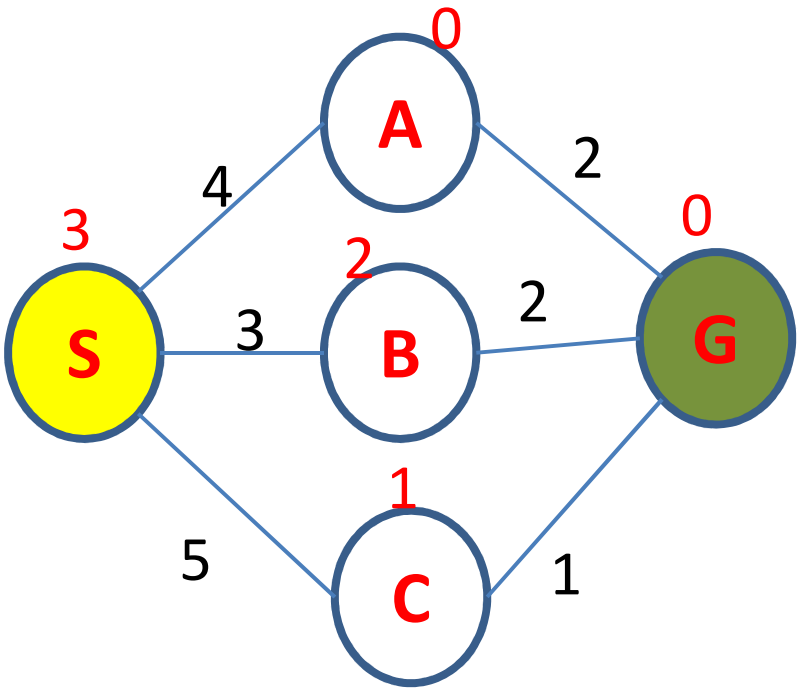




All children explored
Adjust f-values

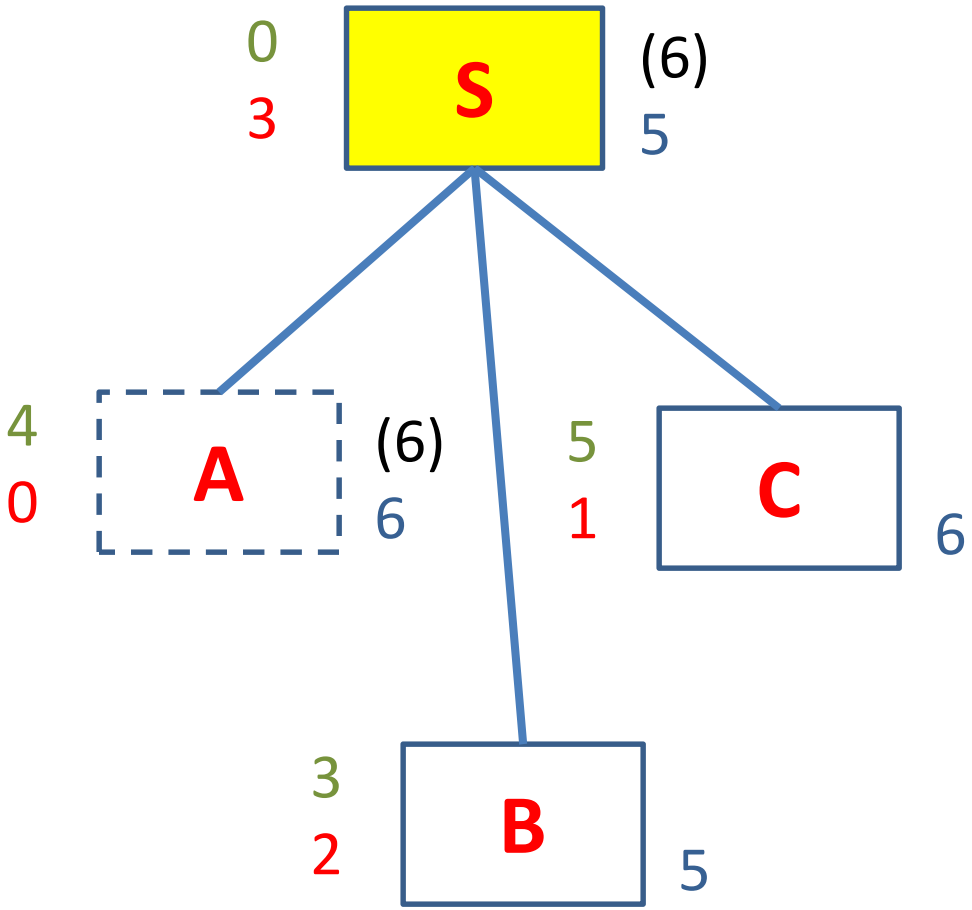
SMA* Example

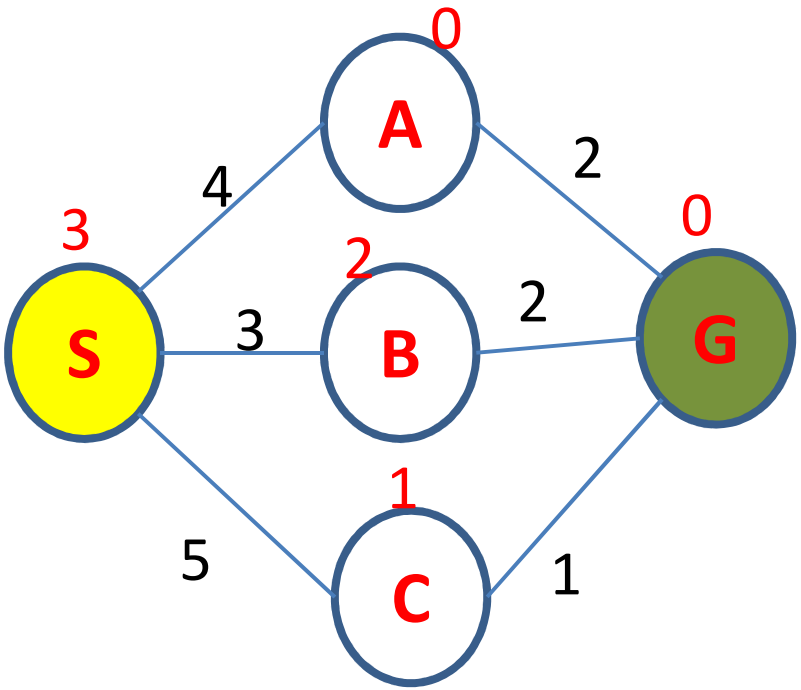




Memory Full
Remove G

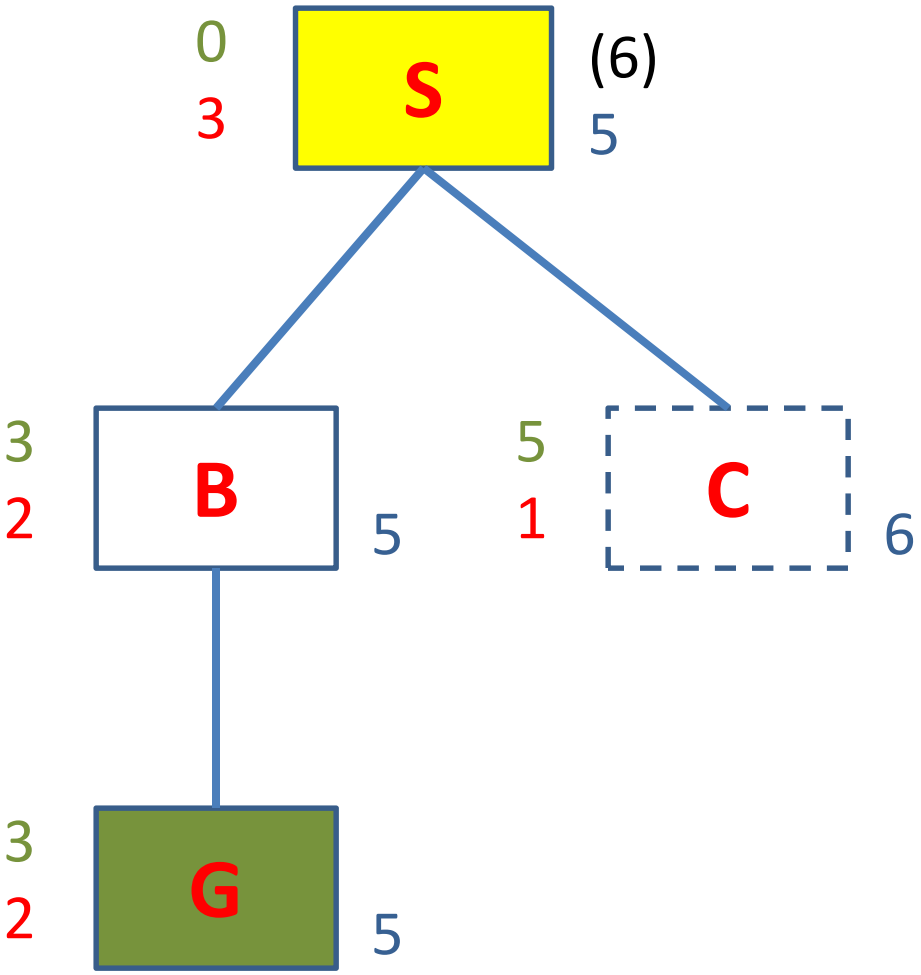
SMA* Example

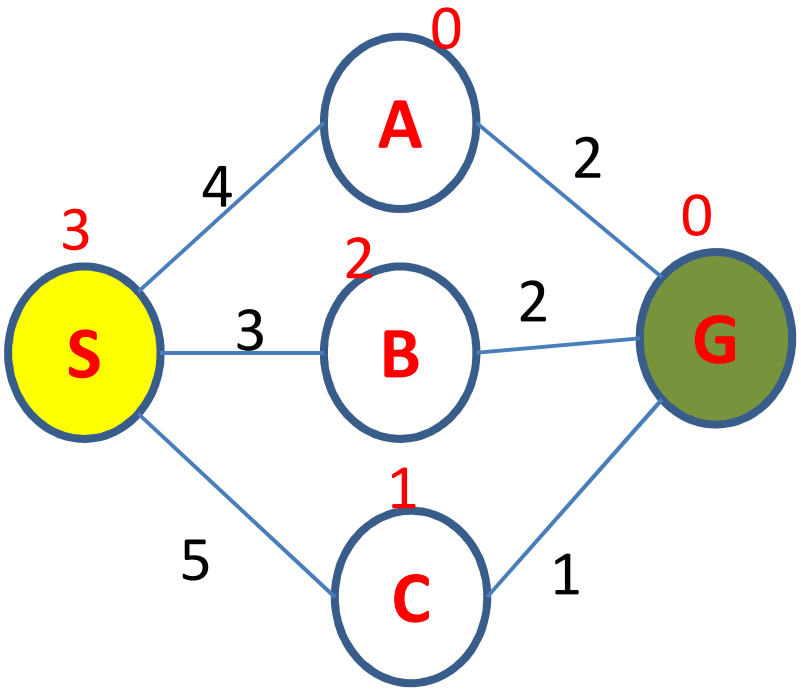




Memory Full
Remove A

SMA* Example





Memory Full
Remove C

Question

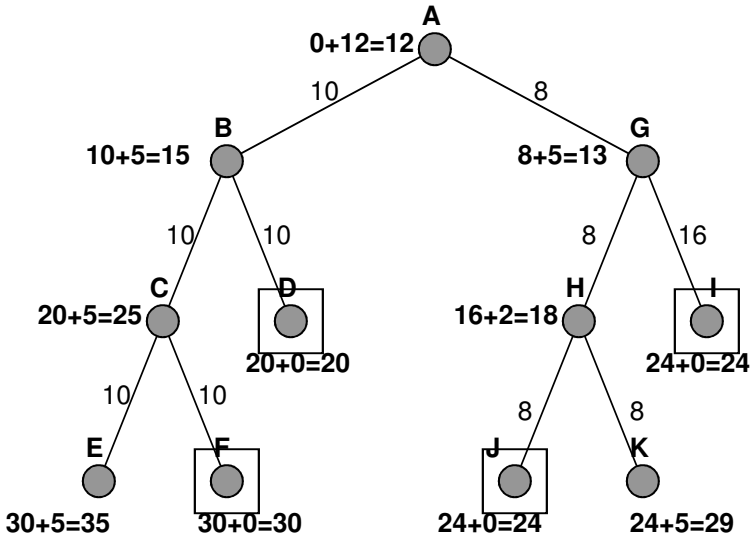
- **Current Rule:** Replace new node with the leaf with worst f-value ?
- **New Rule:** Why not replace only if f-value of new leaf is better than the leaf that it was replacing ?
- **Example in next slide is only for illustration of this counter example.**
- **Please follow the notation used in the SMA* example worked in earlier slides for exam purposes**

Simple Memory-bounded A* (SMA*)

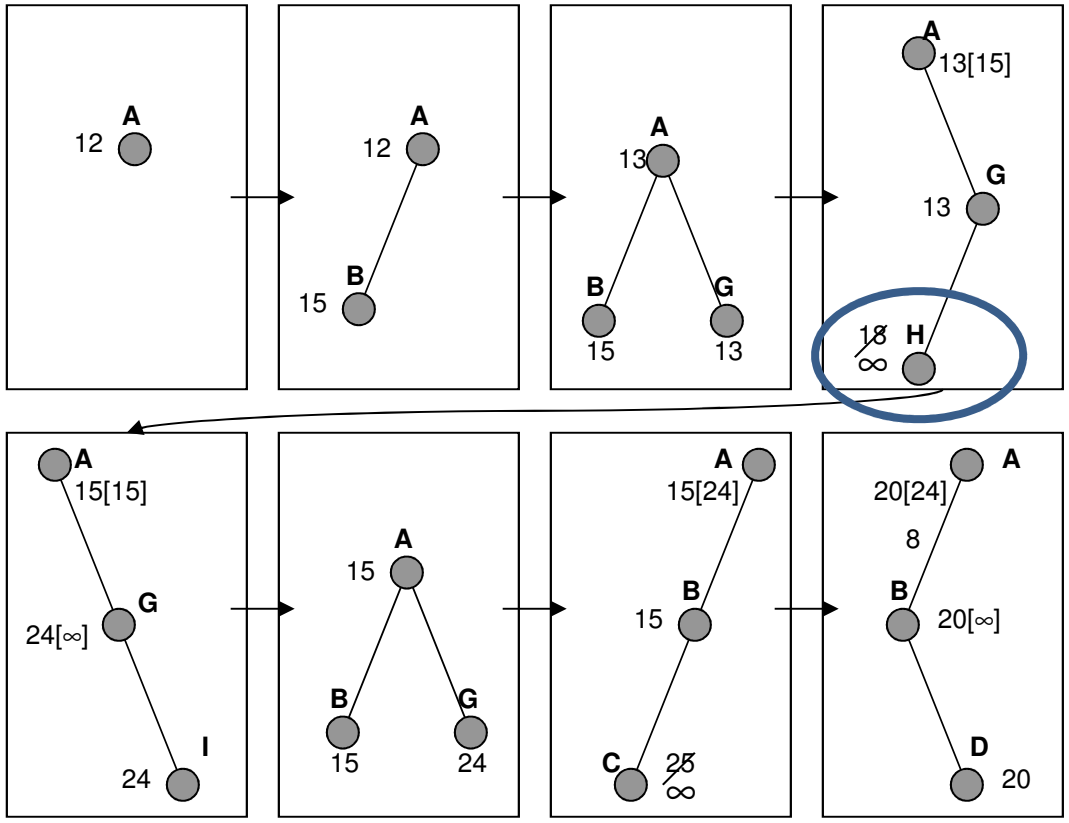
(Example with 3-node memory)

Search space

f = g+h □ = goal



Progress of SMA*. Each node is labeled with its *current f*-cost.
Values in parentheses show the value of the best forgotten descendant.



Algorithm can tell you when best solution found within memory constraint is optimal or not.

One way to view this

- Goal: On an **average** we want to minimize re-expansion of same paths
- We deal with average statistics
- Finer details of the algorithm do vary at implementation level
- Possible to develop variants with some improvements but may not be significant or may add other overheads
- For purposes of this class, please stick with the rule mentioned earlier
  - New rules are obviously encouraged if you can demonstrate (ex: proof or large scale experimentation) that it is (significantly) better

SMA* Details

- In case of tie, expands newest best leaf and deletes oldest worst leaf
- If leaf is not goal node and no more memory, solution is **not reachable** even if on optimal solution path
- Node can discarded as if it has no successors i.e. f-value is infinity
- SMA* is **complete** if there is reachable solution i.e. depth of shallowest goal node is less than memory size
- Pretty robust choice for finding optimal solutions
- On very hard problems, possible to delete and regenerate a path many times due to limited memory (thrashing)
- Problem can become intractable due to overhead of repeated regeneration