Université de M'Sila
Faculté des mathématiques et de l'informatique
Département d'informatique

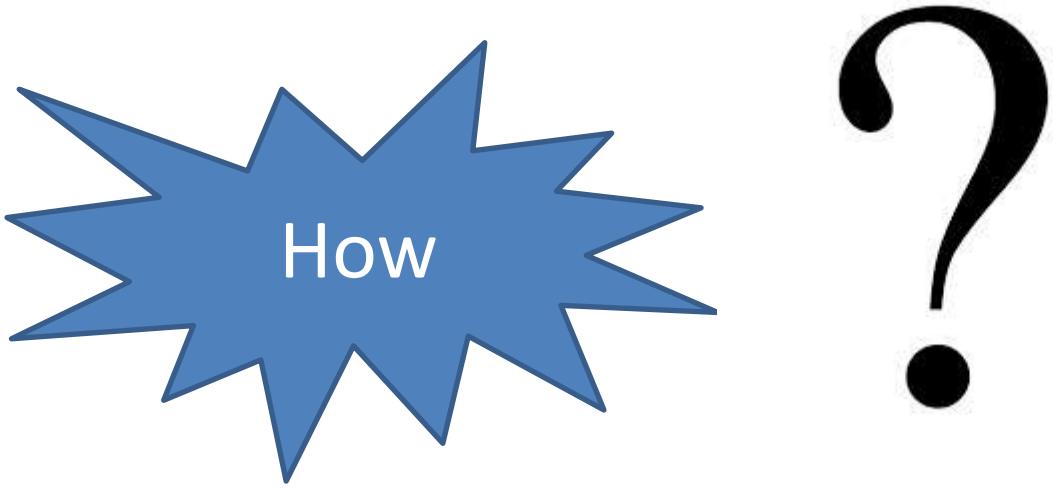
2 Solving Problems by Searching

2 ND YEAR MASTER IDO
2025-2024

Université de M'Sila
Faculté des mathématiques et de l'informatique
Département d'informatique

PROBLEM SOLVING THROUGH RESEARCH

Research plays a key role in AI. These algorithms provide the conceptual basis for almost all approaches to the systematic exploration of state spaces.

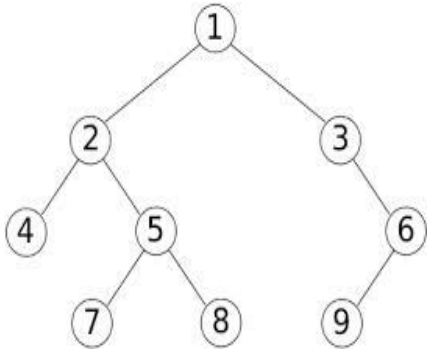


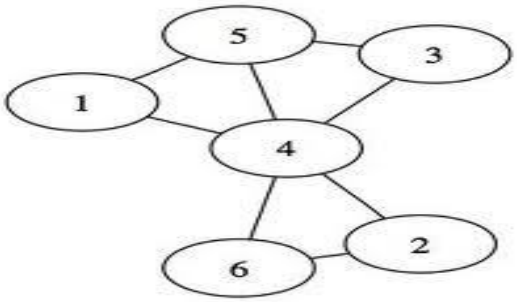
Terminology

- State: description of the world relative to the problem considered
- State space (or search space): set of all possible states
- Action: operation allowing you to change state
- Path: sequence of states linked by actions
- Exploration: calculation process which consists of determining a sequence of actions allowing a solution to be reached
- Strategy: choice of the next action to take in the exploration process

- Explore the search space: determine a sequence of actions that achieves the goal

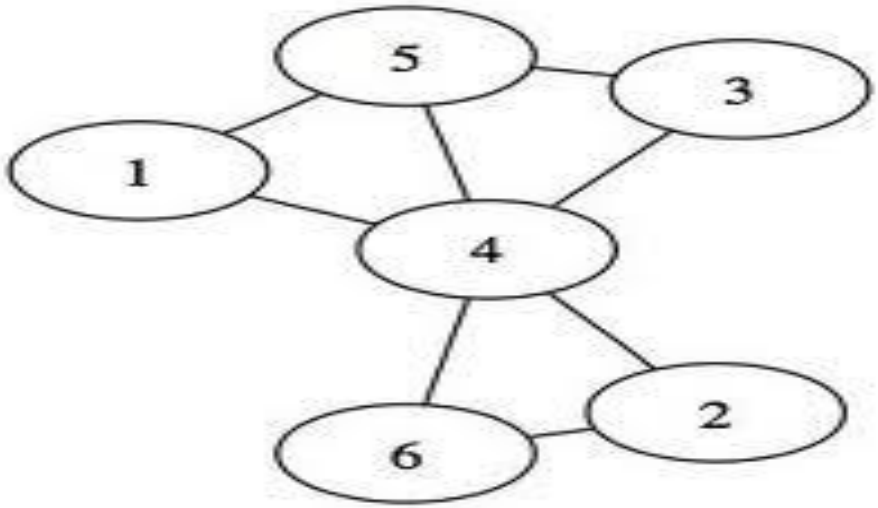
The search





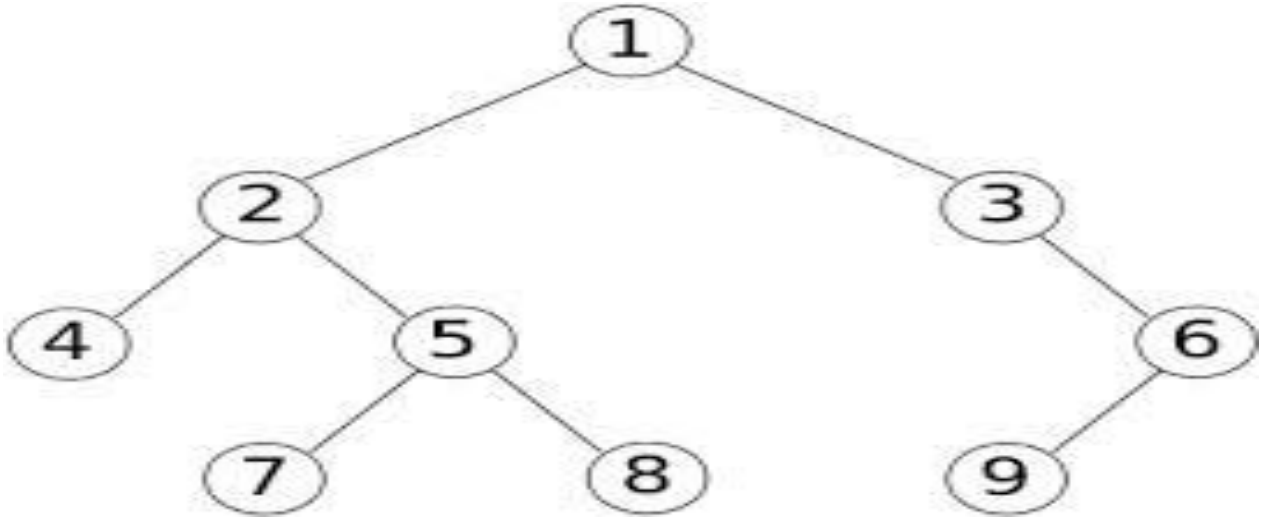
Graphs

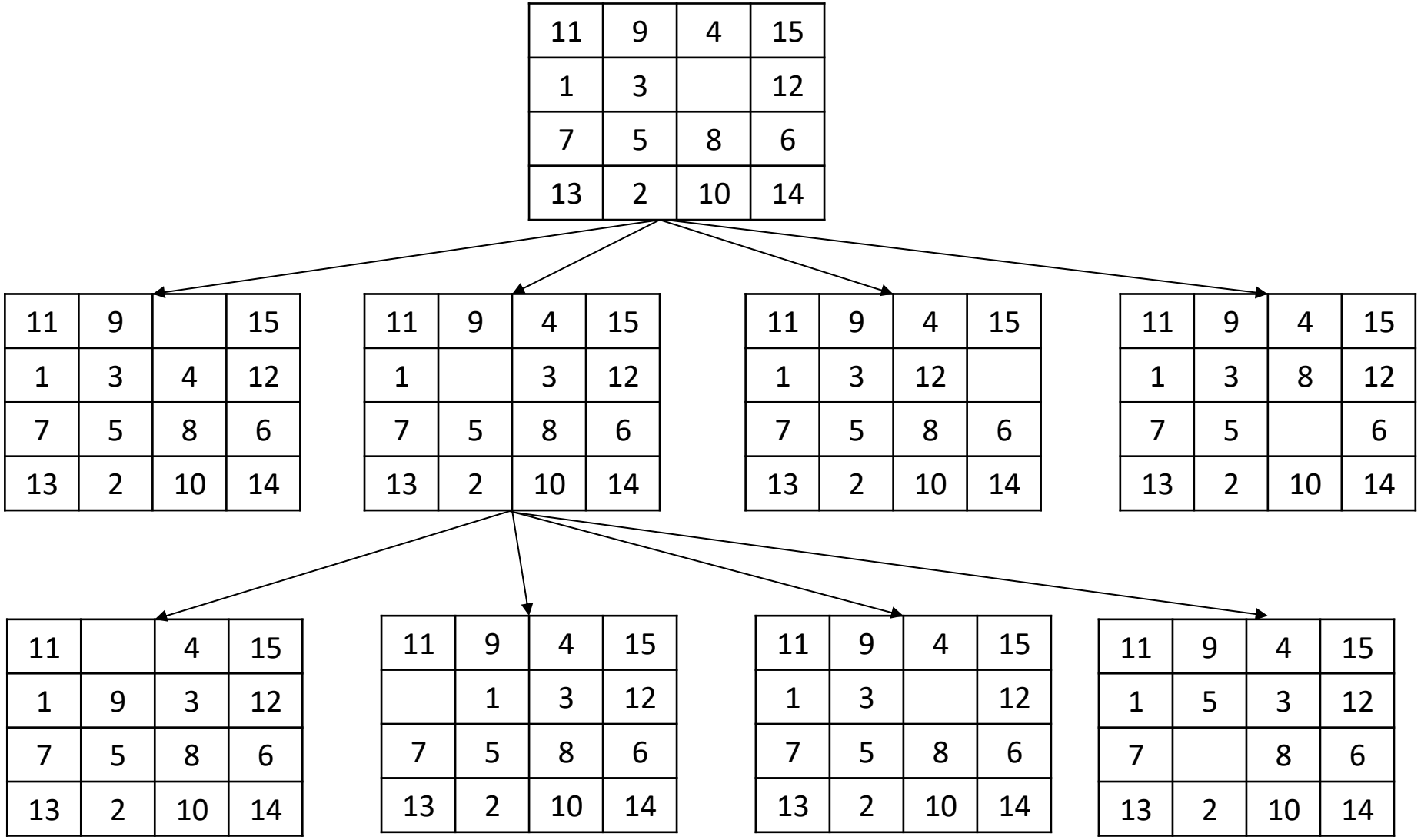
A graph is a set of nodes that are linked together by edges. Mathematically, a graph is represented by a pair of two sets G = (X;U) where X is the set of nodes and U is the set of arcs.



Trees

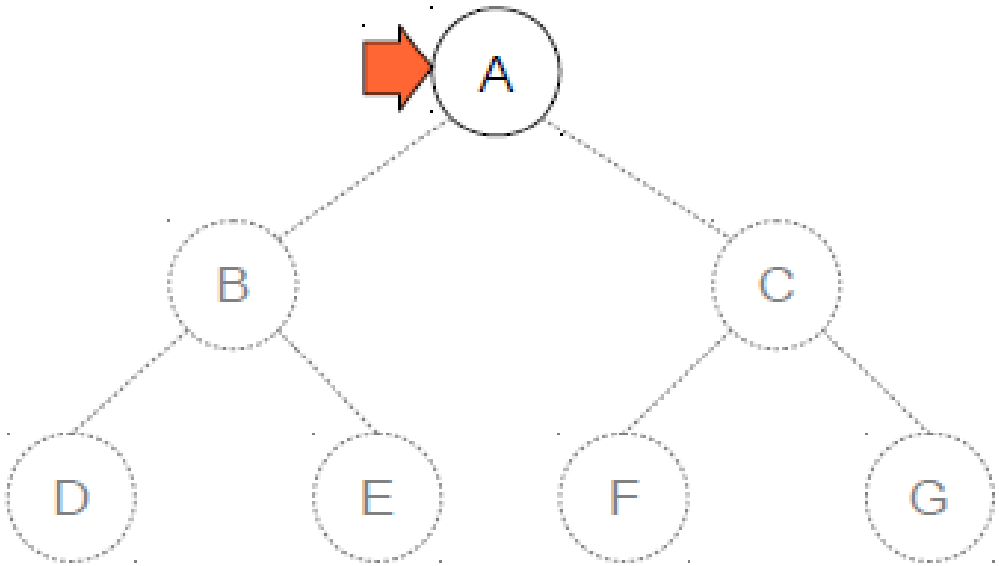
A tree is a connected graph without a cycle. It therefore has n - 1 arcs. We can therefore say that a tree is a graph that connects all the nodes together with a minimum of edges.




11 9 4 15
1 3 12
7 5 8 6
13 2 10 14
11 9 4 15
1 3 4 12
7 5 8 6
13 2 10 14
11 9 4 15
1 3 12
7 5 8 6
13 2 10 14
11 9 4 15
1 3 12
7 5 8 6
13 2 10 14
11 9 4 15
1 3 8 12
7 5 6
13 2 10 14
11 4 15
1 9 3 12
7 5 8 6
13 2 10 14
11 9 4 15
1 3 12
7 5 8 6
13 2 10 14
11 9 4 15
1 3 12
7 5 8 6
13 2 10 14
11 9 4 15
1 5 3 12
7 8 6
13 2 10 14

Breadth-First Search (BFS)

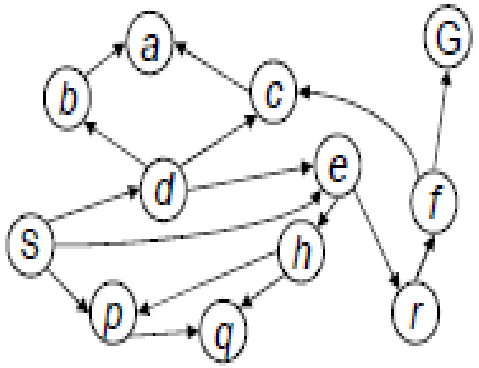
- Development of all nodes at level n before those at level n+1
- The fringe implemented by a FIFO (first-in-first-out) queue

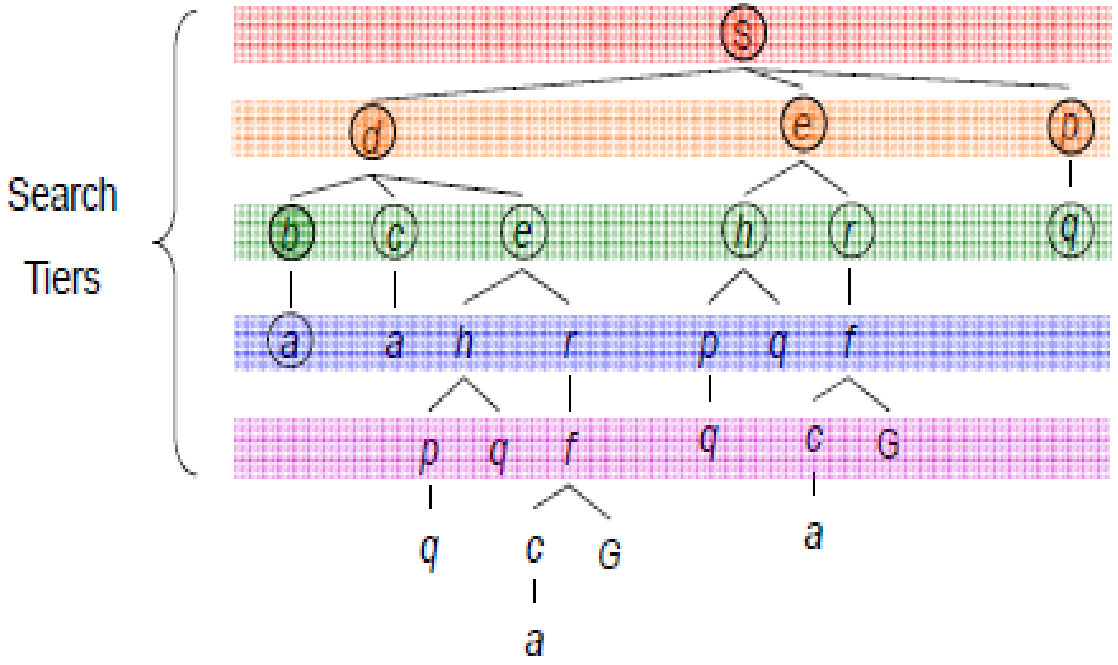


Breadth-First Search

Strategy: expand a shallowest node first

Implementation: Fringe is a FIFO queue

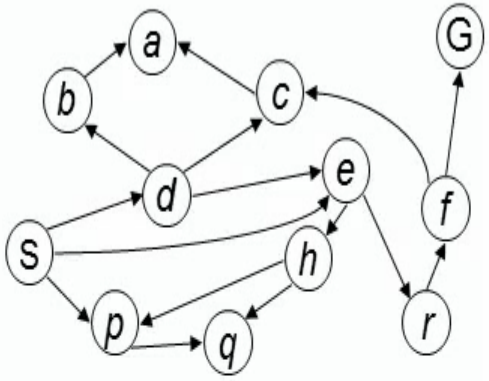


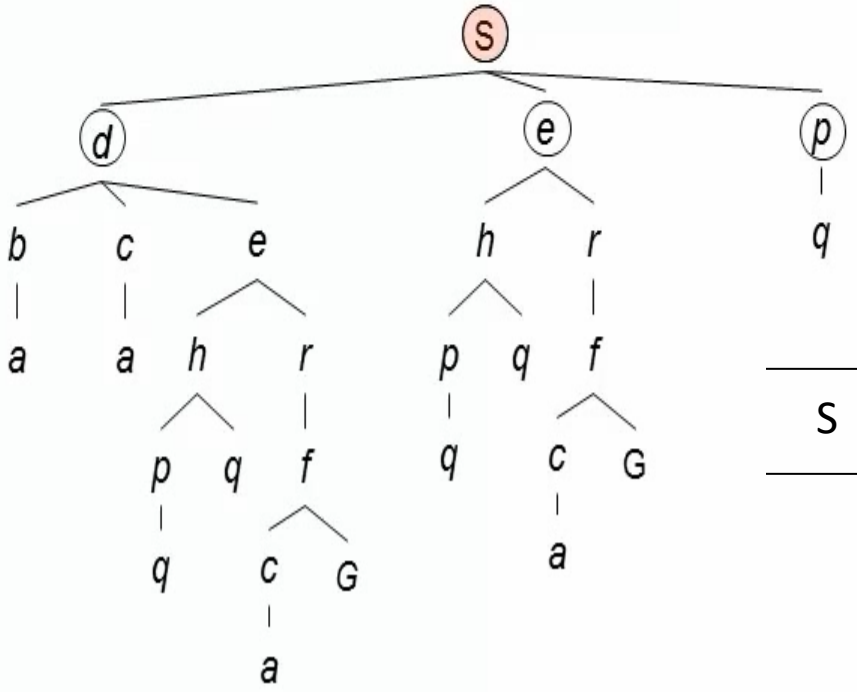


Breadth-First Search

Strategy: expand a shallowest node first

Implementation: Fringe is a FIFO queue





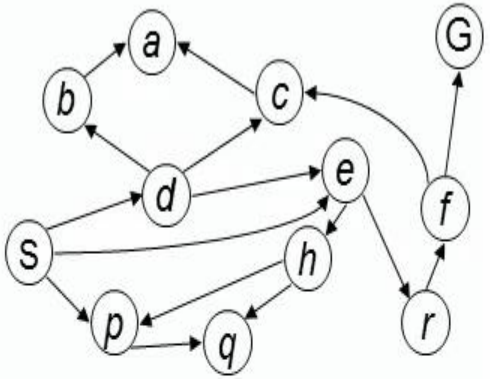
S

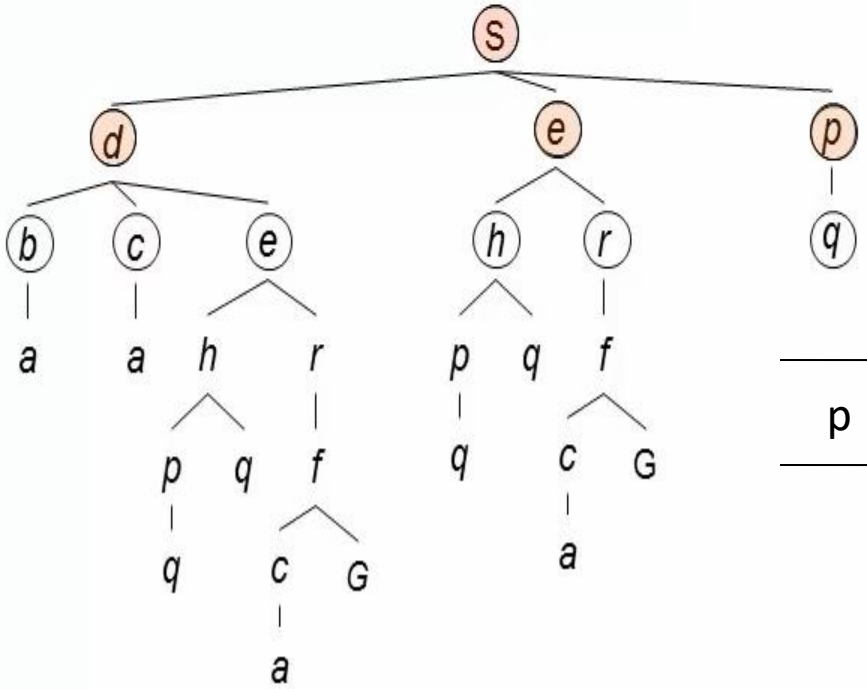
The fringe (FIFO)

Breadth-First Search

Strategy: expand a shallowest node first

Implementation: Fringe is a FIFO queue





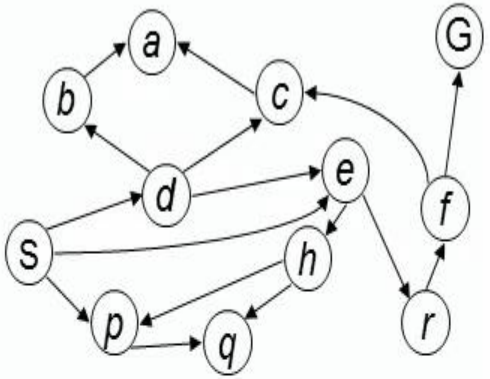
p e d

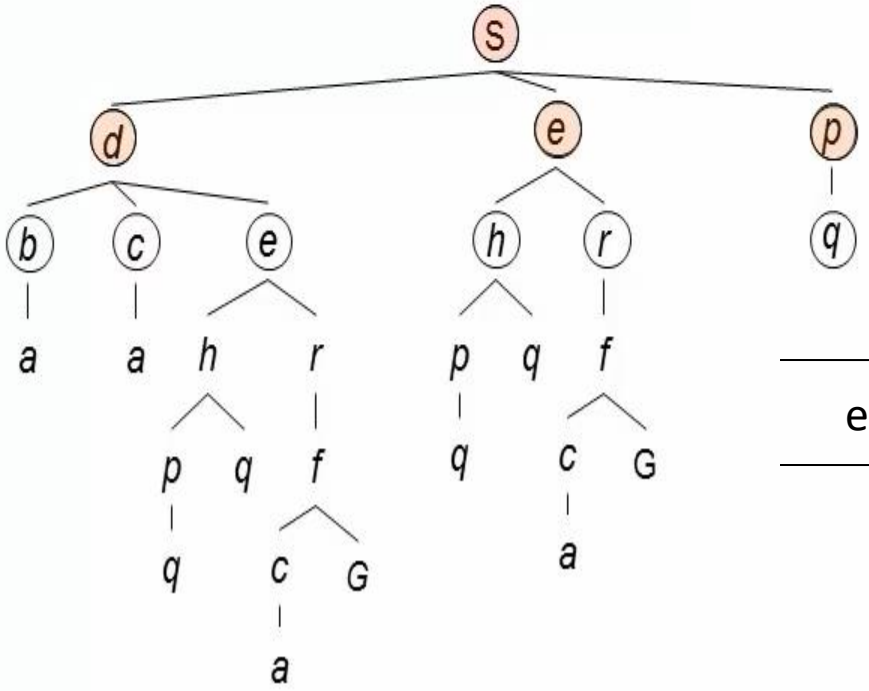
The fringe (FIFO)

Breadth-First Search

Strategy: expand a shallowest node first

Implementation: Fringe is a FIFO queue





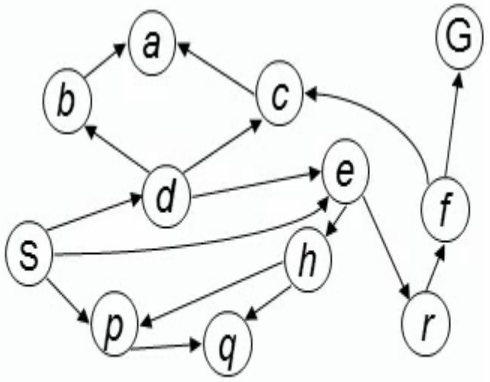
e c b p e

The fringe (FIFO)

Breadth-First Search

Strategy: expand a shallowest node first

Implementation: Fringe is a FIFO queue

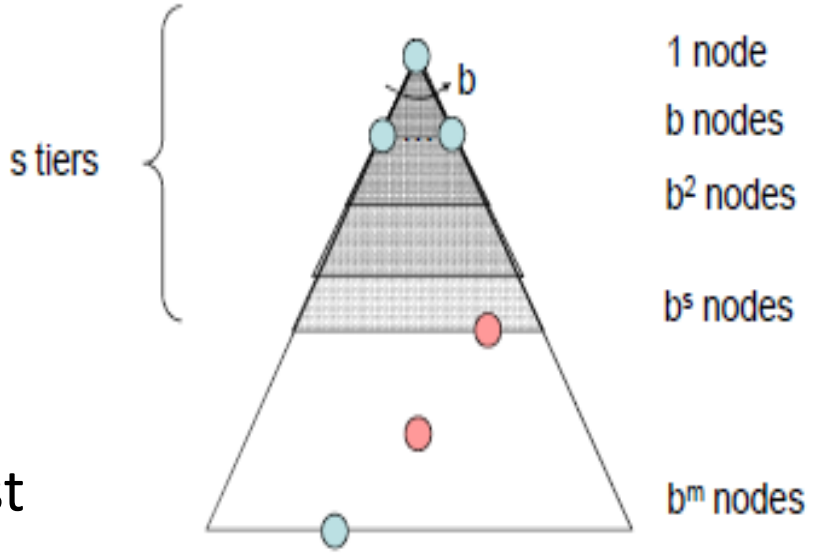


La recherche en Largeur d'abord

S-> D -> E -> P -> B -> C -> E ….

Breadth-First Search (BFS) Properties

If the solution is at level SSo
the search takes
$O(b^s)$ unit of time
$O(b^s)$ is the max fringe size



Complete?

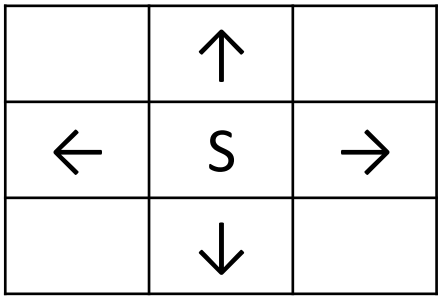
If there is a solution then S must be finite, the BSF algorithm will surely find it.

Optimal?

Only if all arcs have the same cost.

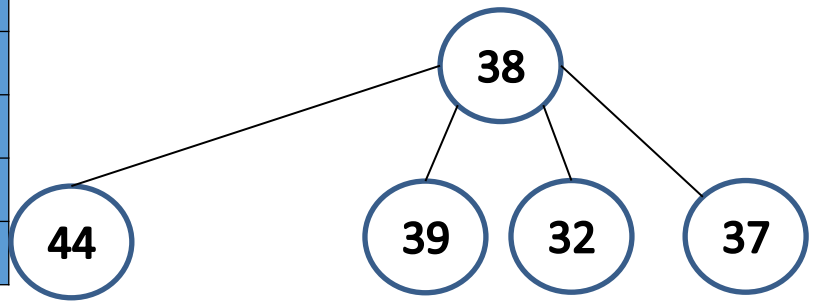
Breadth-First Search

1	2	3	4	5	6
6	7	8	9	10	11
12	13	14	15	16	17
18	19	20	21	22	23
24	25	26	27	28	29
30	31	32	33	34	35
36	37	38	39	40	41
42	43	44	45	46	47
48	49	50	51	52	53
54	55	56	57	58	59

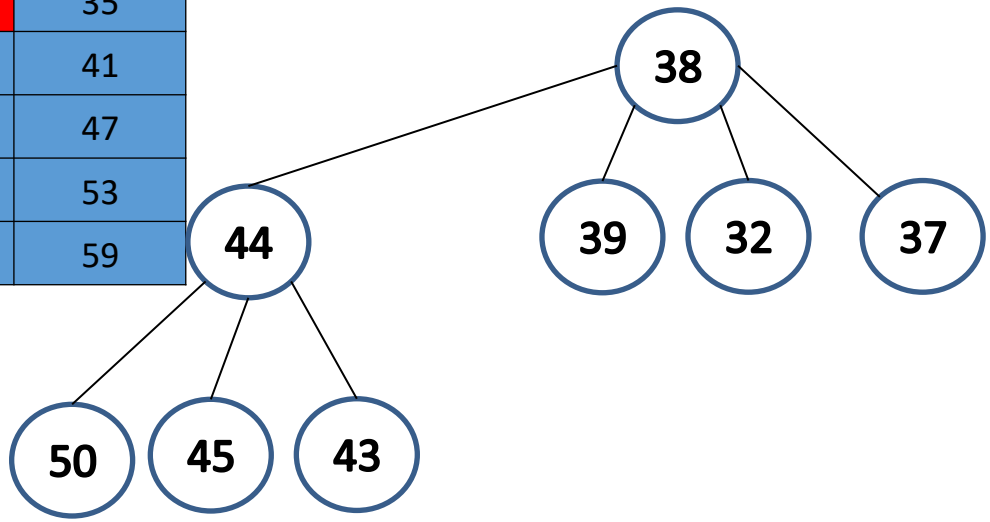


38

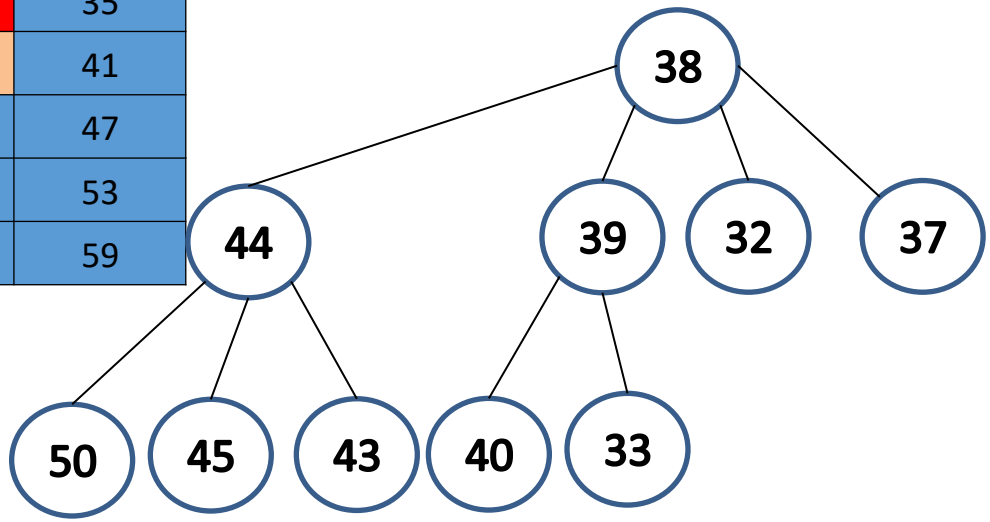
1	2	3	4	5	6
6	7	8	9	10	11
12	13	14	15	16	17
18	19	20	21	22	23
24	25	26	27	28	29
30	31	32	33	34	35
36	37	38	39	40	41
42	43	44	45	46	47
48	49	50	51	52	53
54	55	56	57	58	59



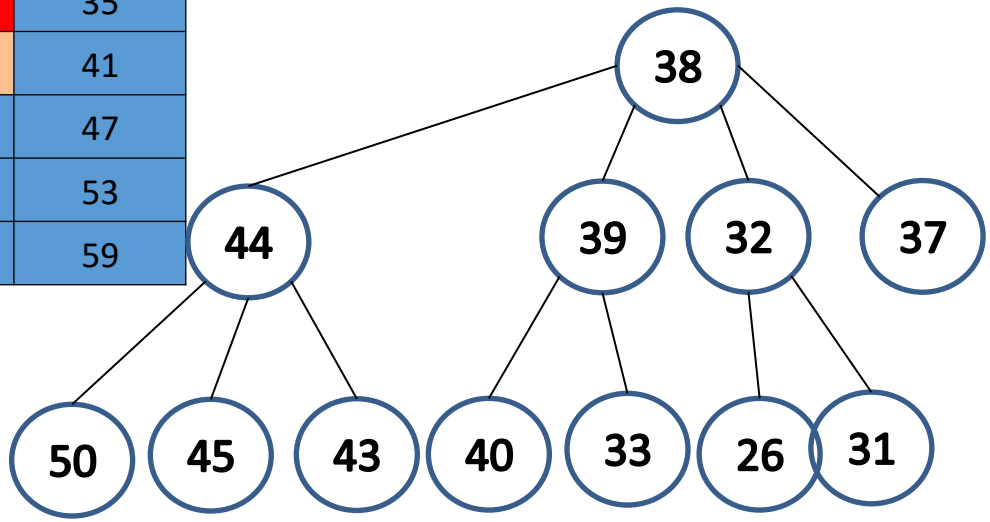
1	2	3	4	5	6
6	7	8	9	10	11
12	13	14	15	16	17
18	19	20	21	22	23
24	25	26	27	28	29
30	31	32	33	34	35
36	37	38	39	40	41
42	43	44	45	46	47
48	49	50	51	52	53
54	55	56	57	58	59



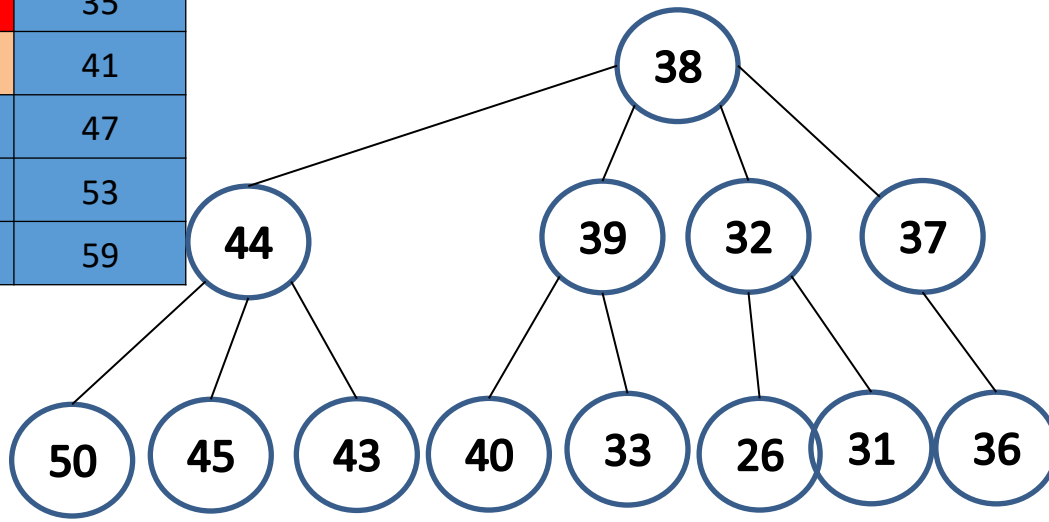
1	2	3	4	5	6
6	7	8	9	10	11
12	13	14	15	16	17
18	19	20	21	22	23
24	25	26	27	28	29
30	31	32	33	34	35
36	37	38	39	40	41
42	43	44	45	46	47
48	49	50	51	52	53
54	55	56	57	58	59



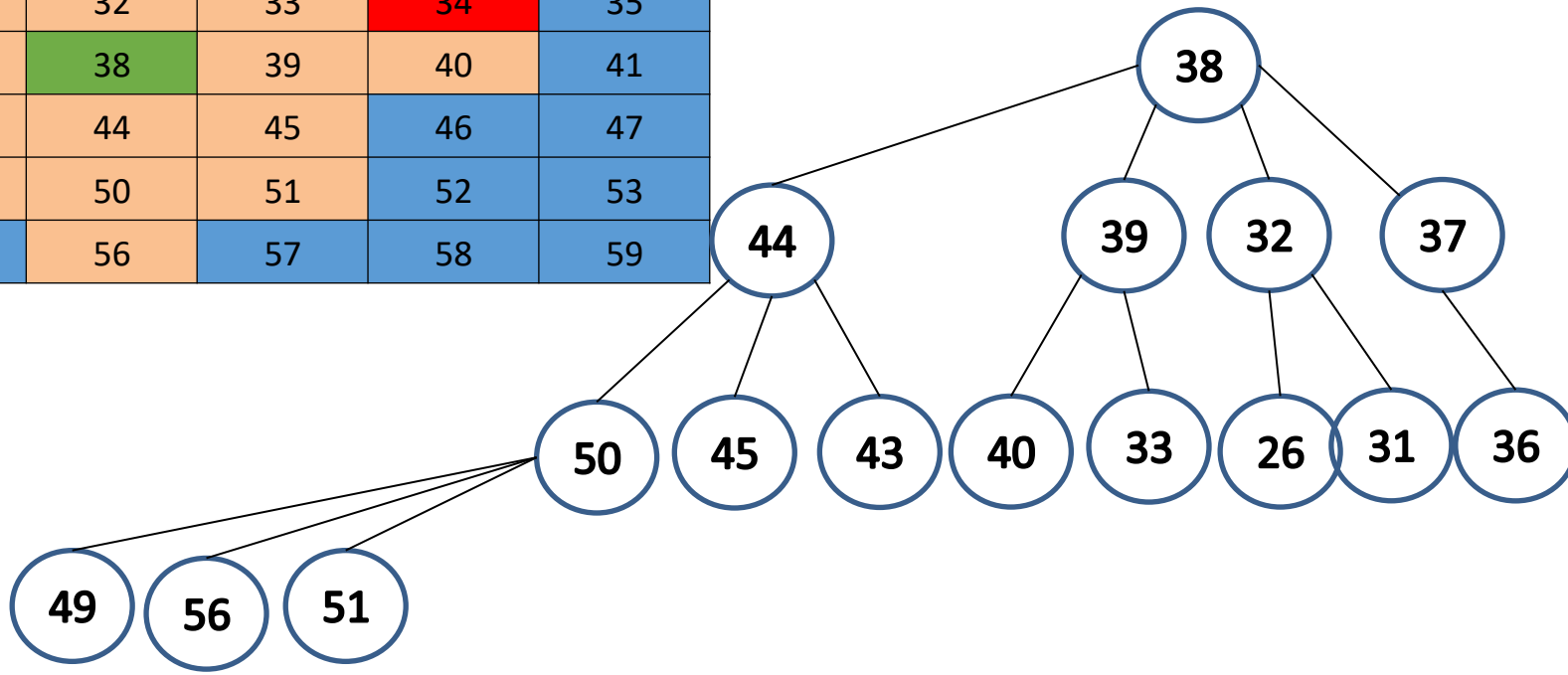
1	2	3	4	5	6
6	7	8	9	10	11
12	13	14	15	16	17
18	19	20	21	22	23
24	25	26	27	28	29
30	31	32	33	34	35
36	37	38	39	40	41
42	43	44	45	46	47
48	49	50	51	52	53
54	55	56	57	58	59



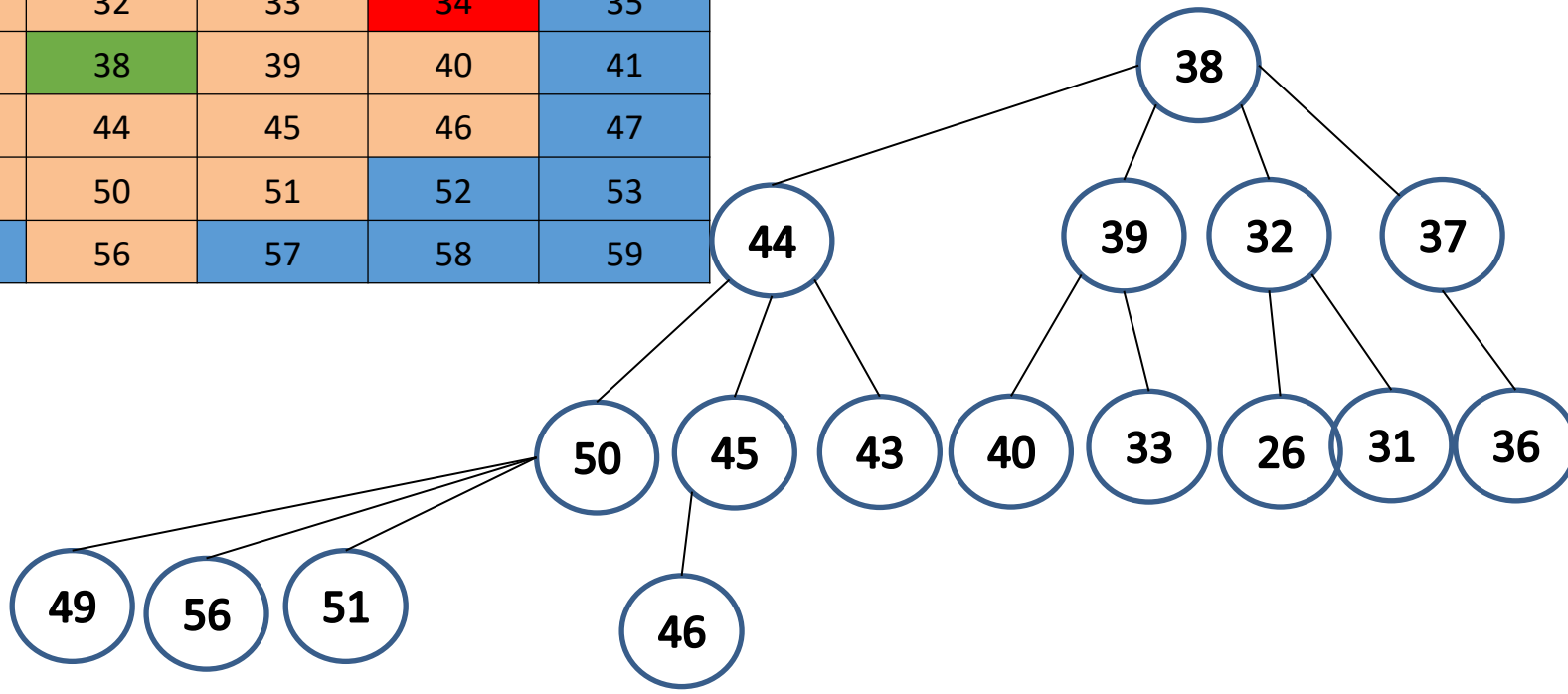
1	2	3	4	5	6
6	7	8	9	10	11
12	13	14	15	16	17
18	19	20	21	22	23
24	25	26	27	28	29
30	31	32	33	34	35
36	37	38	39	40	41
42	43	44	45	46	47
48	49	50	51	52	53
54	55	56	57	58	59



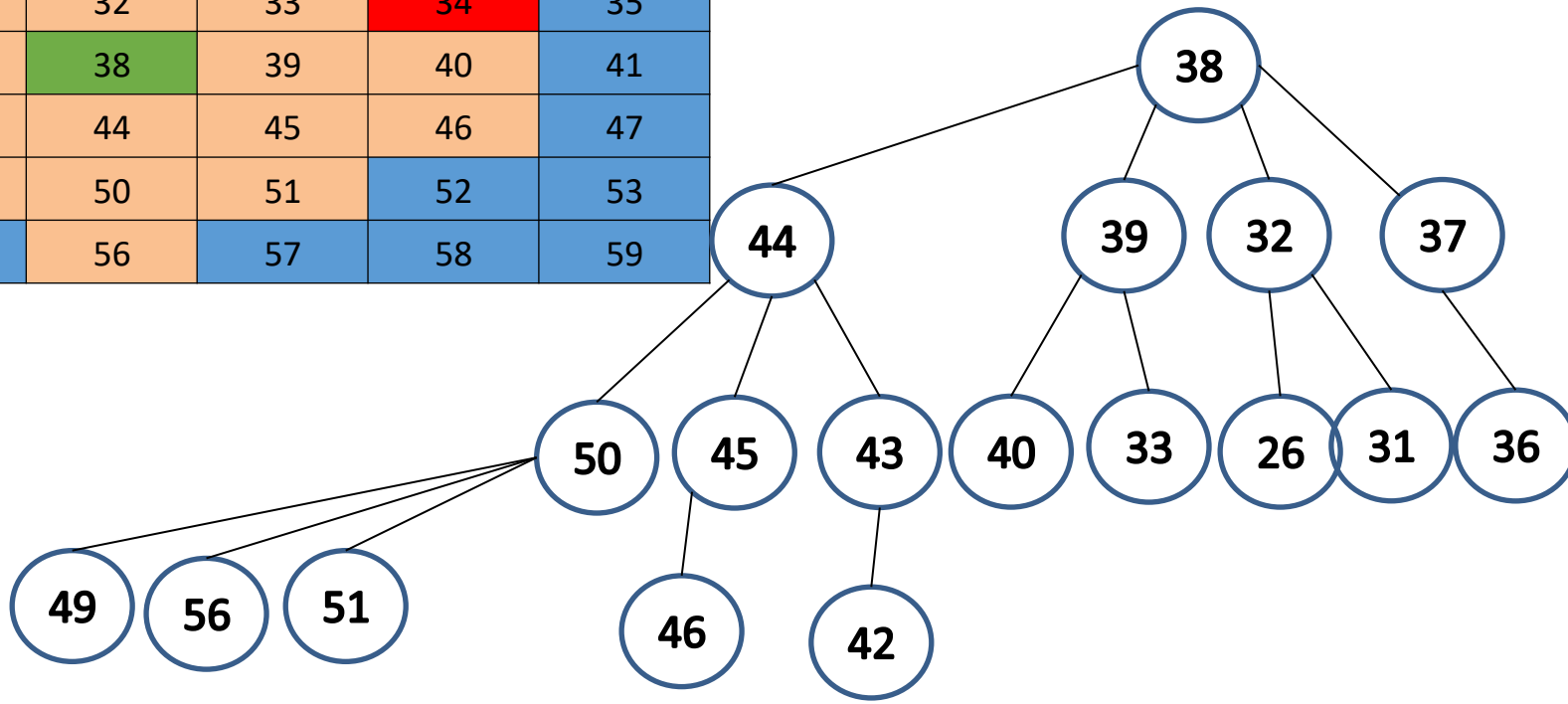
1	2	3	4	5	6
6	7	8	9	10	11
12	13	14	15	16	17
18	19	20	21	22	23
24	25	26	27	28	29
30	31	32	33	34	35
36	37	38	39	40	41
42	43	44	45	46	47
48	49	50	51	52	53
54	55	56	57	58	59



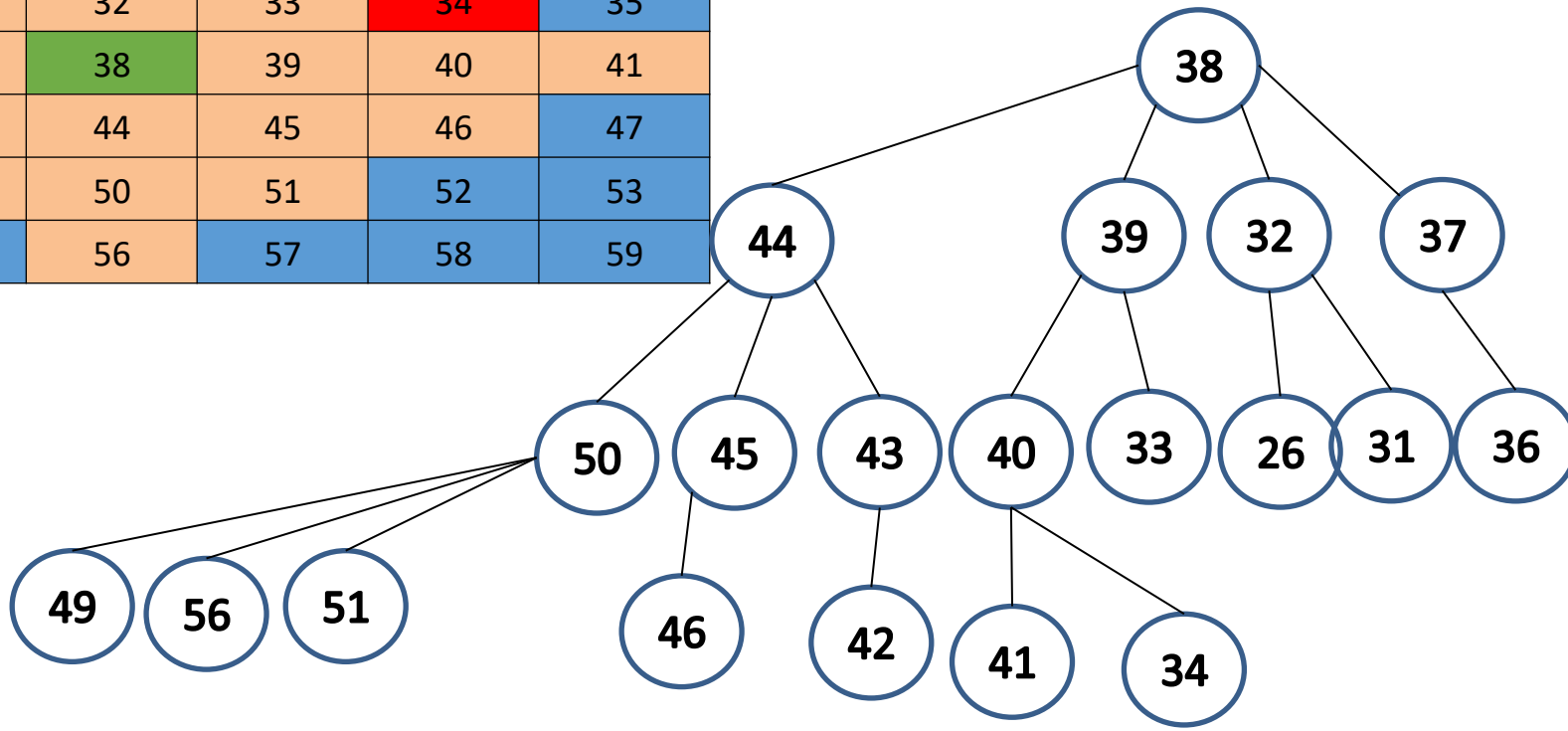
1	2	3	4	5	6
6	7	8	9	10	11
12	13	14	15	16	17
18	19	20	21	22	23
24	25	26	27	28	29
30	31	32	33	34	35
36	37	38	39	40	41
42	43	44	45	46	47
48	49	50	51	52	53
54	55	56	57	58	59



1	2	3	4	5	6
6	7	8	9	10	11
12	13	14	15	16	17
18	19	20	21	22	23
24	25	26	27	28	29
30	31	32	33	34	35
36	37	38	39	40	41
42	43	44	45	46	47
48	49	50	51	52	53
54	55	56	57	58	59

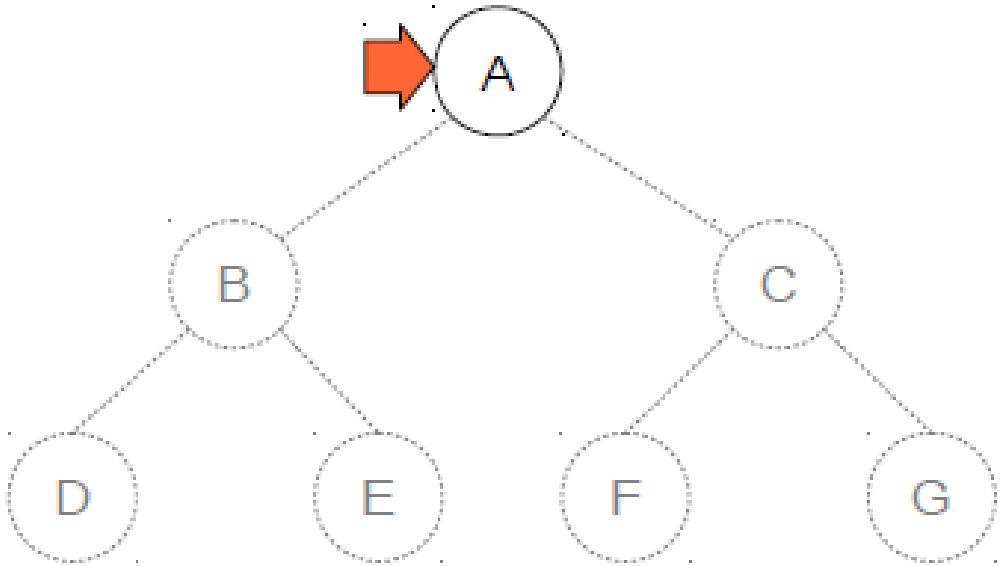


1	2	3	4	5	6
6	7	8	9	10	11
12	13	14	15	16	17
18	19	20	21	22	23
24	25	26	27	28	29
30	31	32	33	34	35
36	37	38	39	40	41
42	43	44	45	46	47
48	49	50	51	52	53
54	55	56	57	58	59



Depth-First Search

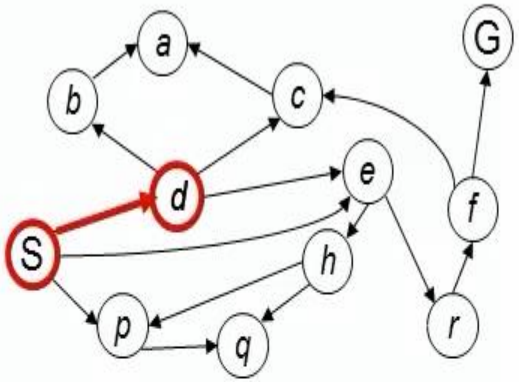
- Expansion of the deepest node of the search tree boundary
- The fringe implemented by a stack (or queue)LIFO (last-in-first-out)
- Alternative: recursive function

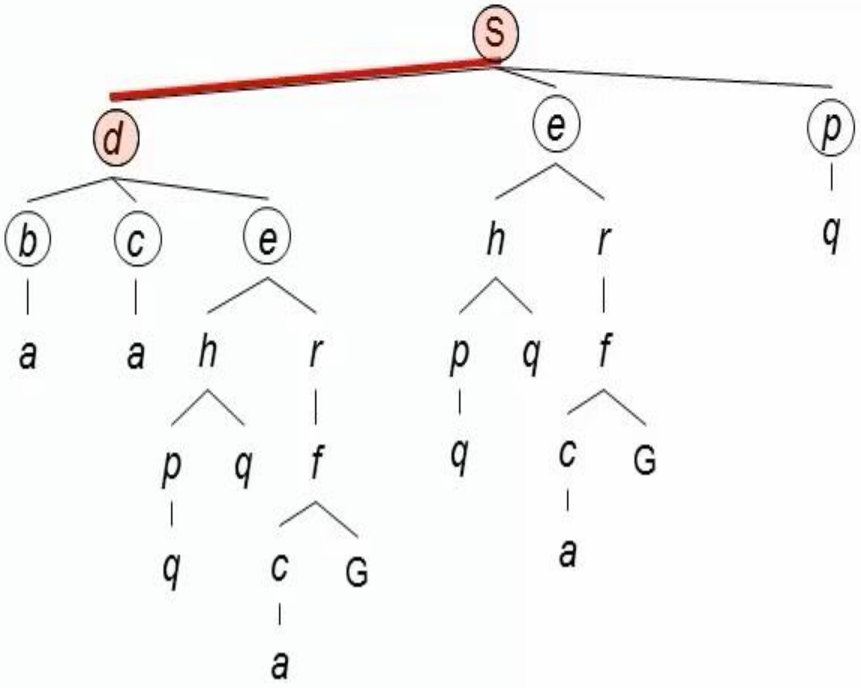


Depth-First Search

Strategy: expand a deepest node first

Implementation: Fringe is a LIFO stack





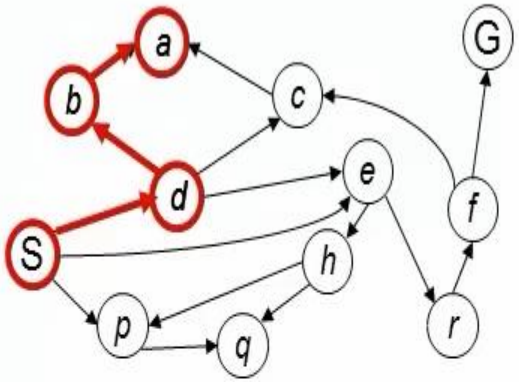
d

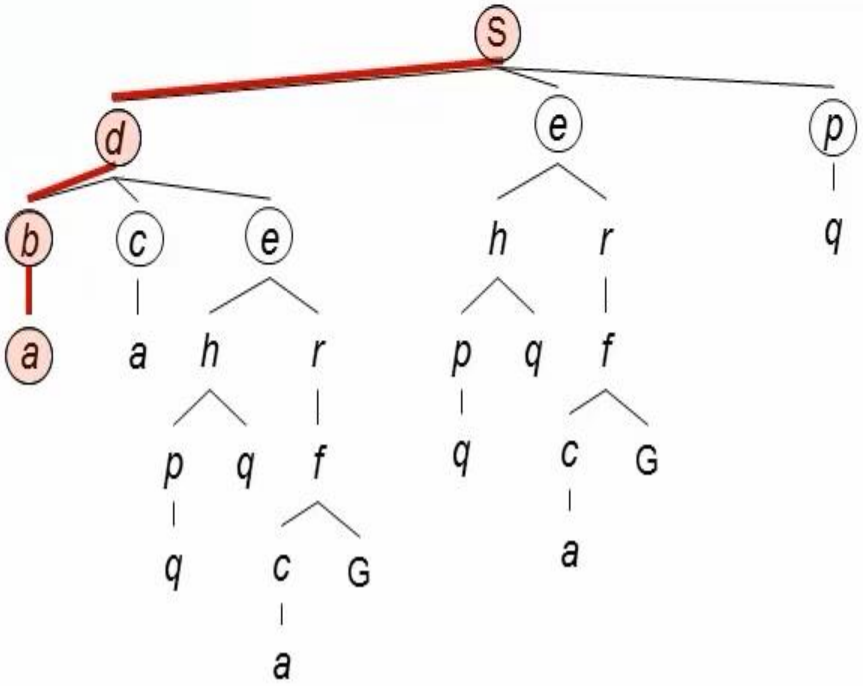
d
S

Depth-First Search

Strategy: expand a deepest node first

Implementation: Fringe is a LIFO stack





c

a
b
d
S

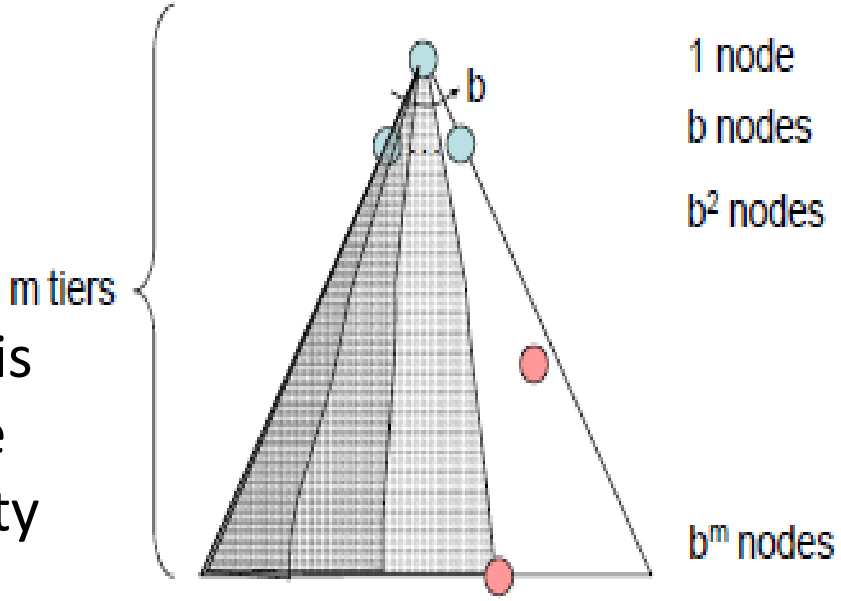
Depth-First Search (DFS) Properties

Complete?

The existence of cycles poses a problem.

time complexity for DFS

depends on the criterion of completeness, because if the DFS is not complete, the calculation time will be infinite, SINO the complexity equal to $O(b^m)$



Space complexity for DFS *O(b.m).*

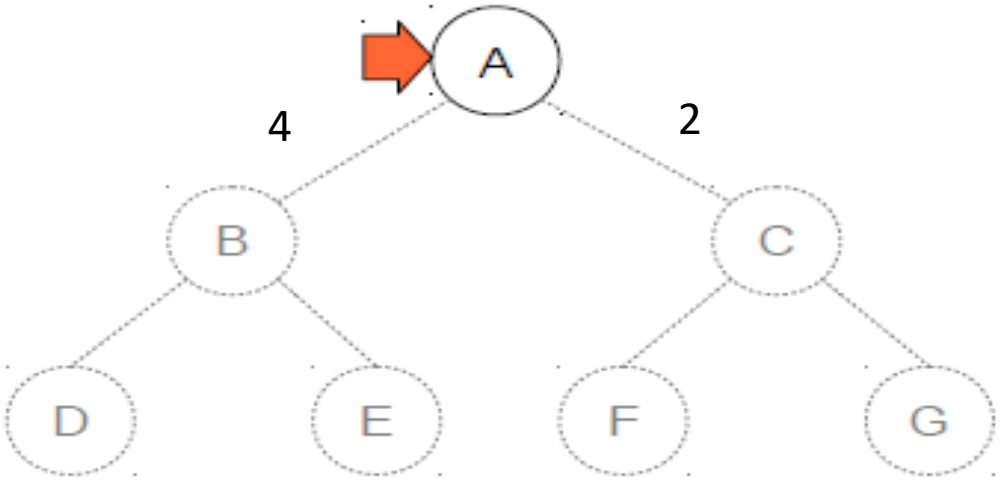
Optimal?

No, we will find the most awkward solution.

Depth-First Search

Uniform Cost Search

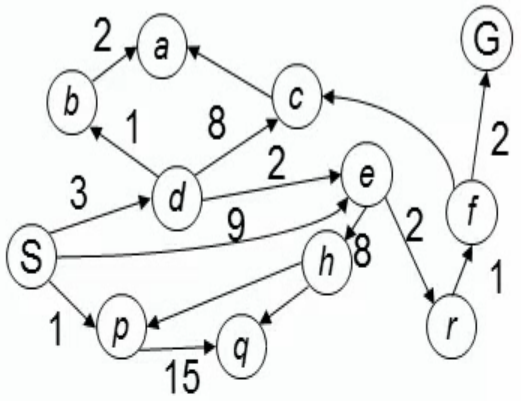
- Expand the node with the lowest cost g(n).
- g(n) = is the sum of costs from the root to node n
- The fringe implemented by File is sorted by cost.
- Equivalent to breadth-first if the cost of actions is alwavs the same.

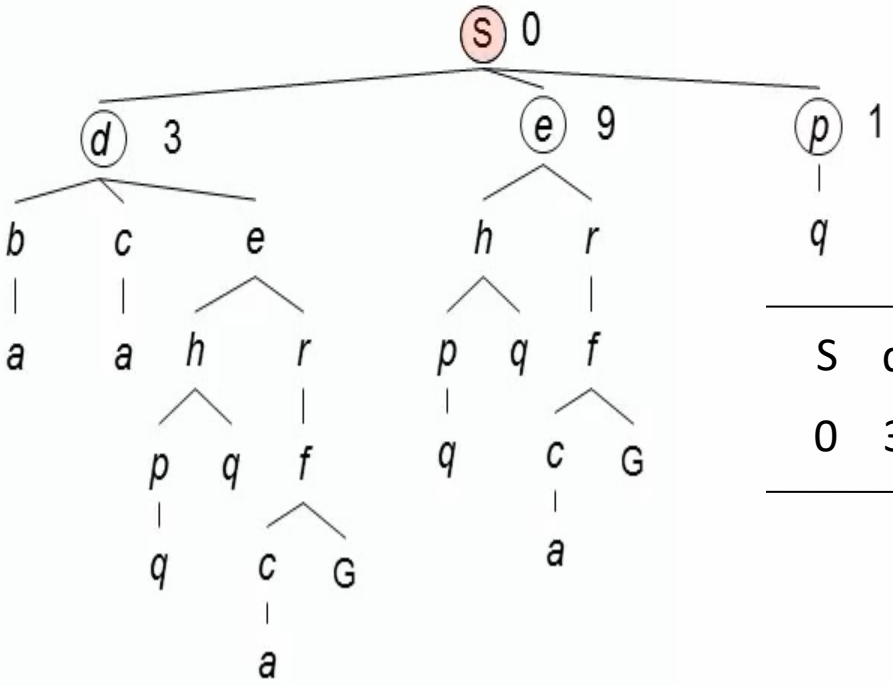


Uniform Cost Search

Strategy: expand a cheapest node first:

Fringe is a priority queue (priority: cumulative cost)





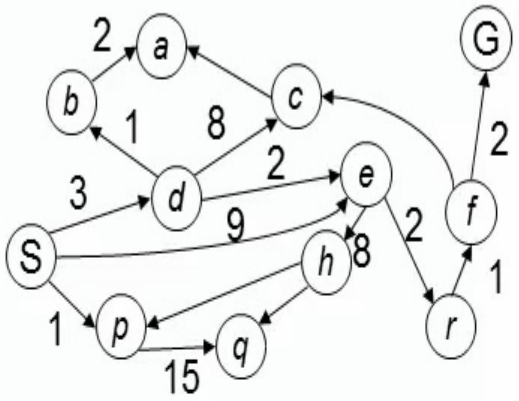
S	d	e	p
0	3	9	1

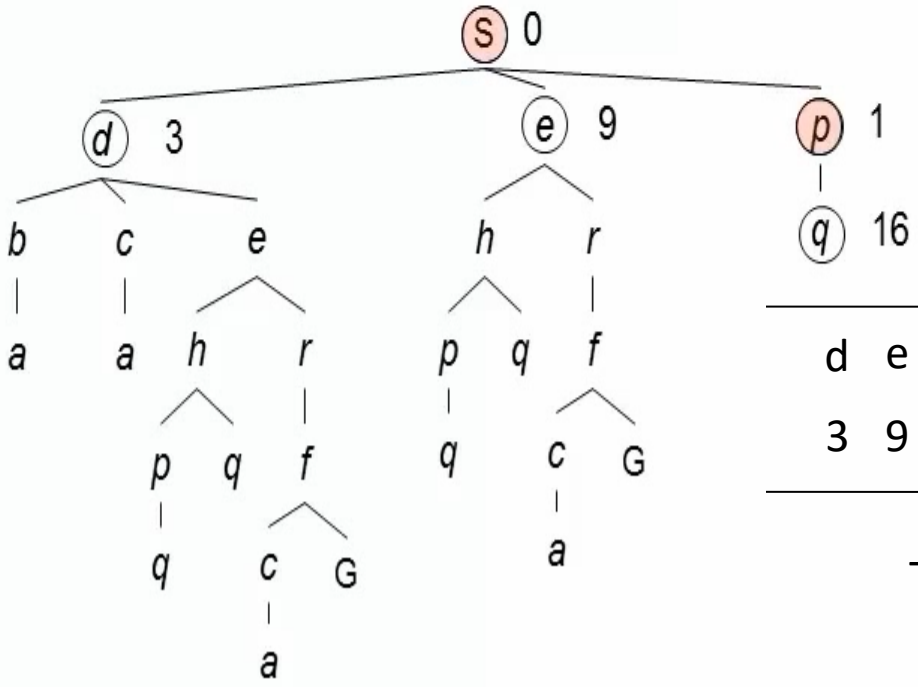
The fringe (FIFO)

Uniform Cost Search

Strategy: expand a cheapest node first:

Fringe is a priority queue (priority: cumulative cost)





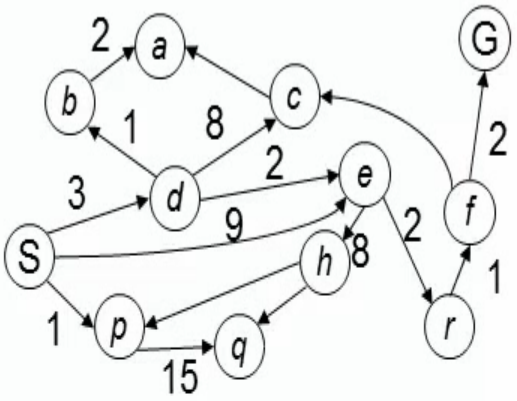
d	e	q
3	9	16

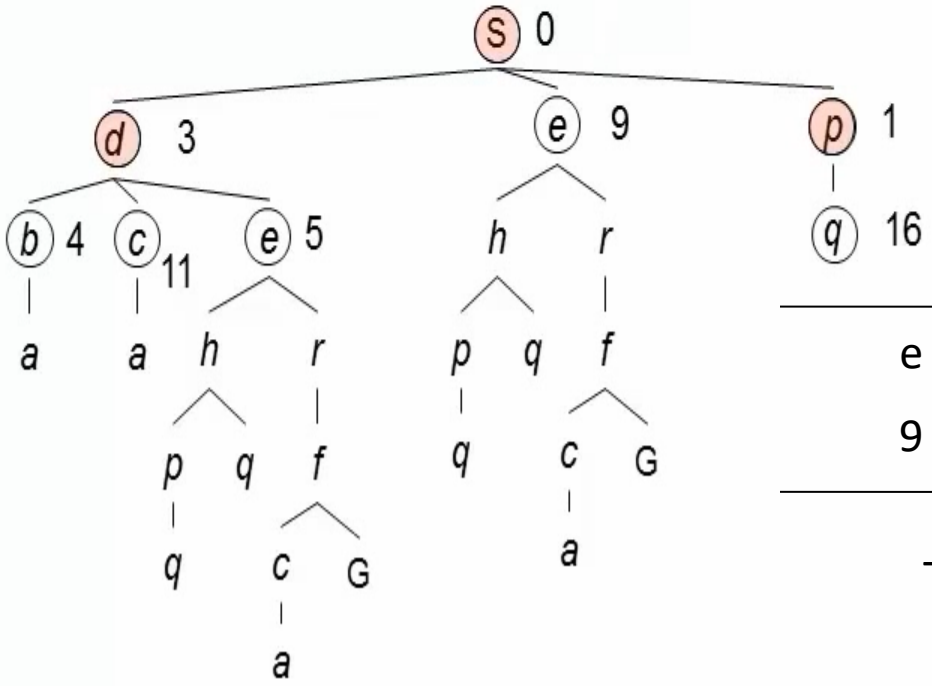
The fringe (FIFO)

Uniform Cost Search

Strategy: expand a cheapest node first:

Fringe is a priority queue (priority: cumulative cost)





e q b c e

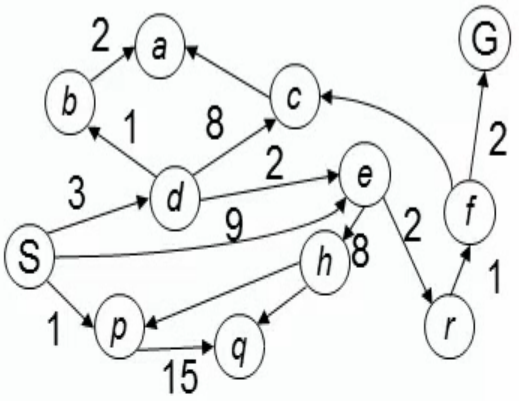
9 16 4 11 5

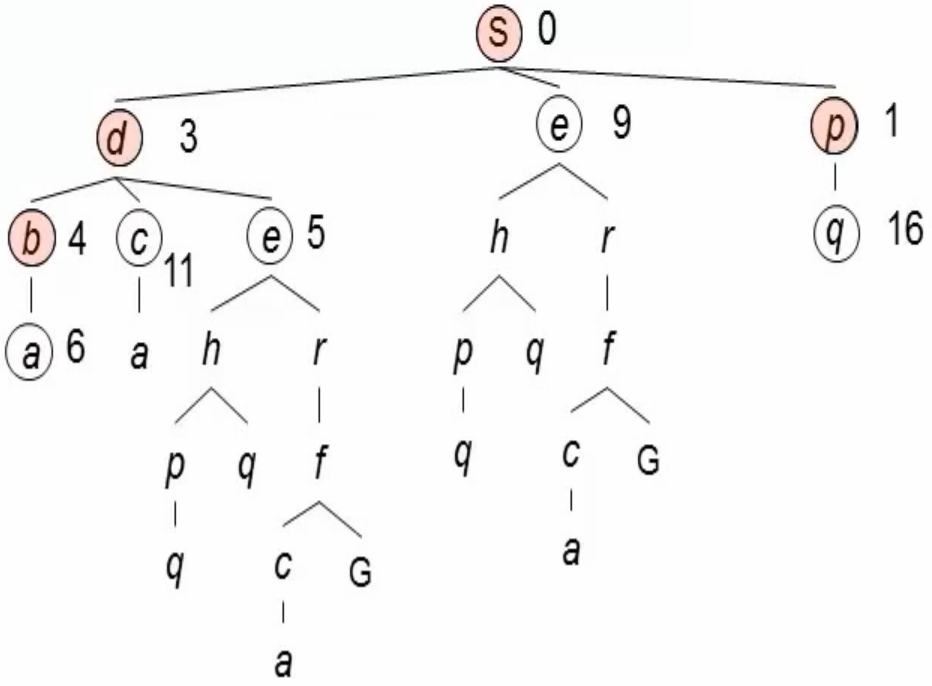
The fringe (FIFO)

Uniform Cost Search

Strategy: expand a cheapest node first:

Fringe is a priority queue (priority: cumulative cost)

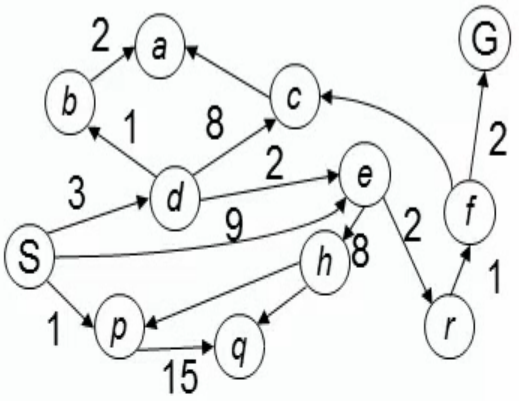


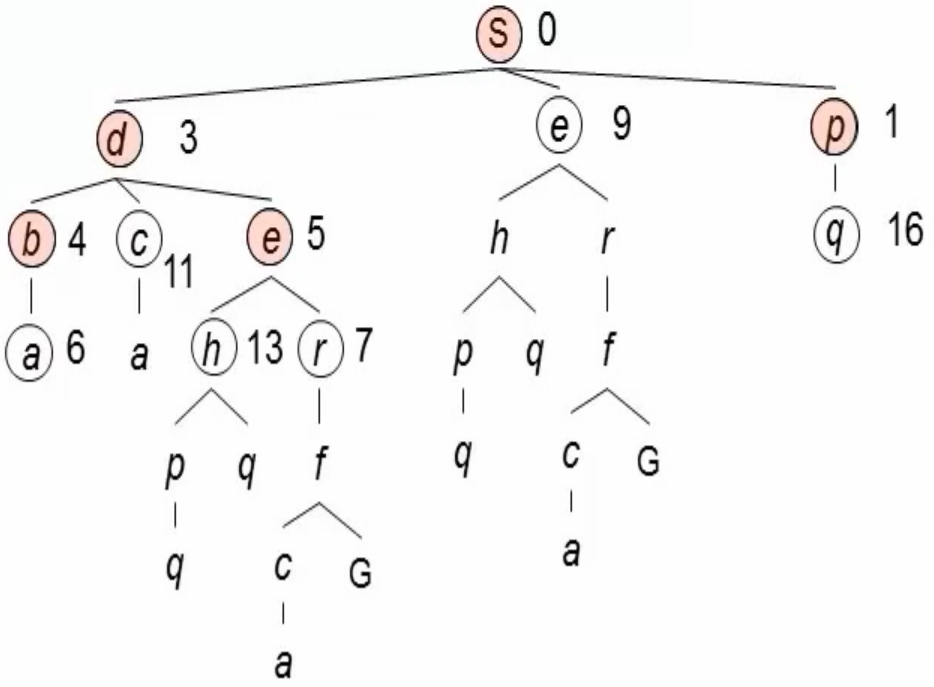


Uniform Cost Search

Strategy: expand a cheapest node first:

Fringe is a priority queue (priority: cumulative cost)





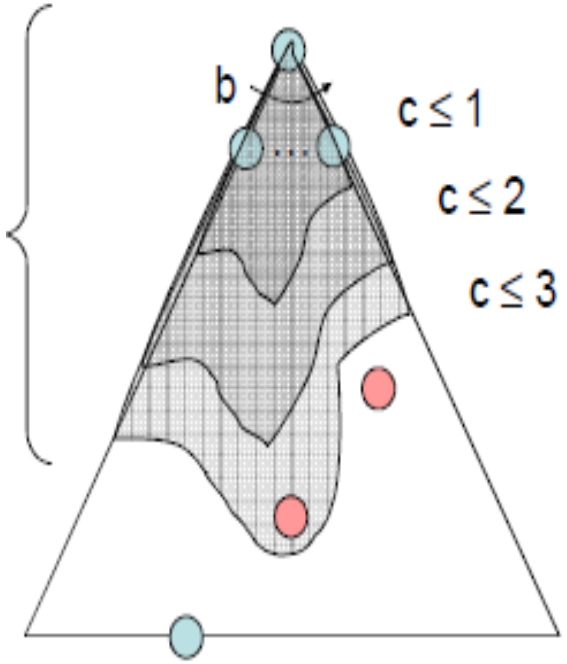
Uniform Cost Search (UCS) Properties

Complete?

If there exists a solution with finite cost and all edge costs are > 0, the UCS algorithm is complete.

Optimal?

Yes, you can consult the proof in the literatures.



Problème : Atteindre le rectangle rouge à partir du vert?

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40	41	42	43	44	45
46	47	48	49	50	51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70	71	72	73	74	75
76	77	78	79	80	81	82	83	84	85	86	87	88	89	90
91	92	93	94	95	96	97	98	99	100	101	102	103	104	105
106	107	108	109	110	111	112	113	114	115	116	117	118	119	120
121	122	123	124	125	126	127	128	129	130	131	132	133	134	135
136	137	138	139	140	141	142	143	144	145	146	147	148	149	150
151	152	153	154	155	156	157	158	159	160	161	162	163	164	165
166	167	168	169	170	171	172	173	174	175	176	177	178	179	180
181	182	183	184	185	186	187	188	189	190	191	192	193	194	195
196	197	198	199	200	201	202	203	204	205	206	207	208	209	210
211	212	213	214	215	216	217	218	219	220	221	222	223	224	225
226	227	228	229	230	231	232	233	234	235	236	237	238	239	240
241	242	243	244	245	246	247	248	249	250	251	252	253	254	255
256	257	258	259	260	261	262	263	264	265	266	267	268	269	270
261	262	263	264	265	266	267	268	269	270	271	272	273	274	275
276	277	278	279	280	281	282	283	284	285	286	287	288	289	290

Simple search algorithms do not use any information about the structure of the search tree.
Most real problems are likely to cause a combinatorial explosion in the number of possible states.

A heuristic search algorithm uses available information to make the search process more efficient.

Informed Search

Heuristic search

The heuristic function

- Heuristic information is a rule or method that improves the decision-making process!

Improve, but How??!!

Function which estimates the position of the current state relative to the objective state (Goal).
In other words, tells us how close or far we are to the goal.

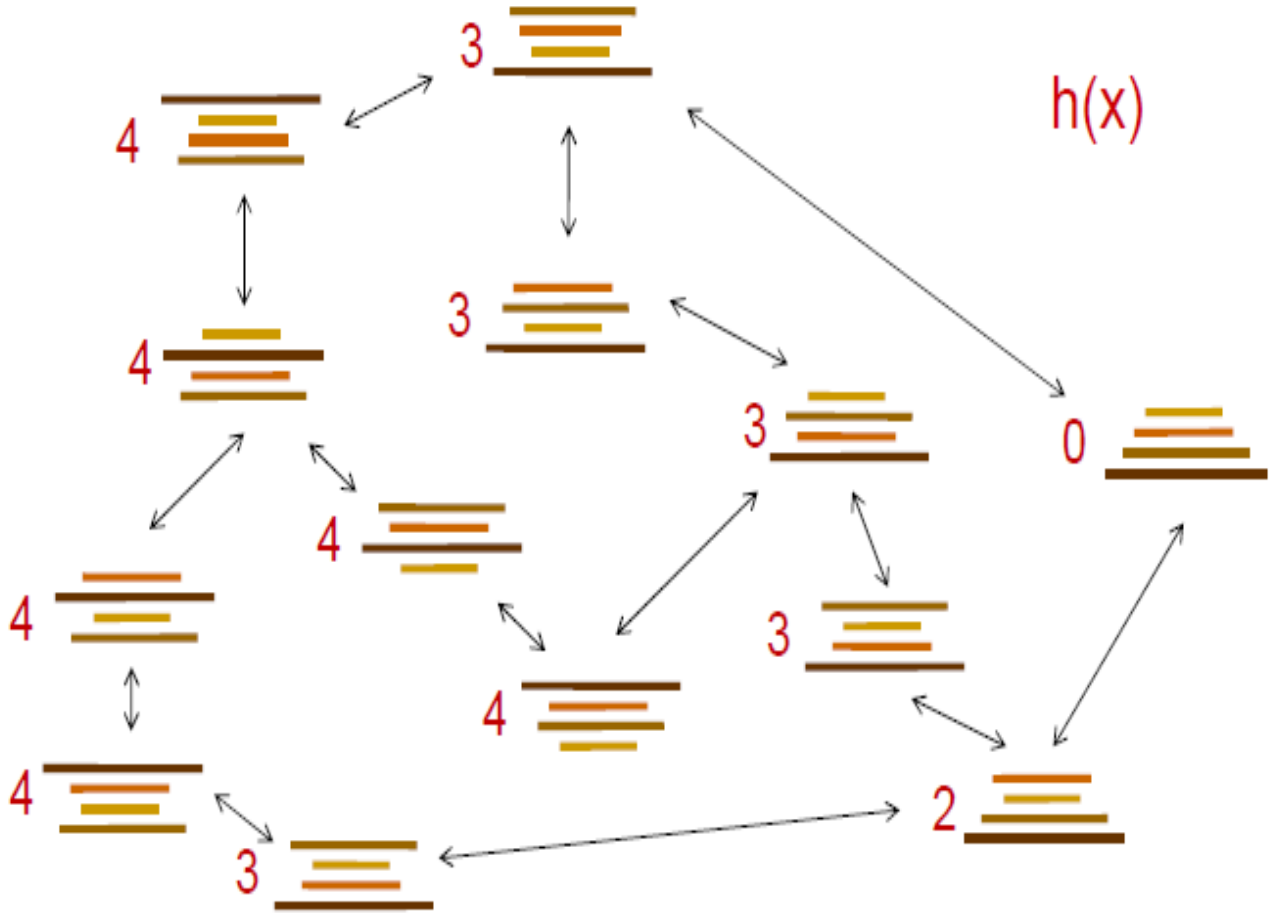
Mathematically:
A heuristic function h: $E \rightarrow \Re$ (E, state space, the set of real numbers)
corresponds to a state $S \in E$ a number $h(s) \in \Re$ which is (generally) an estimate of the cost/benefit ratio of extending the current path passing through s.

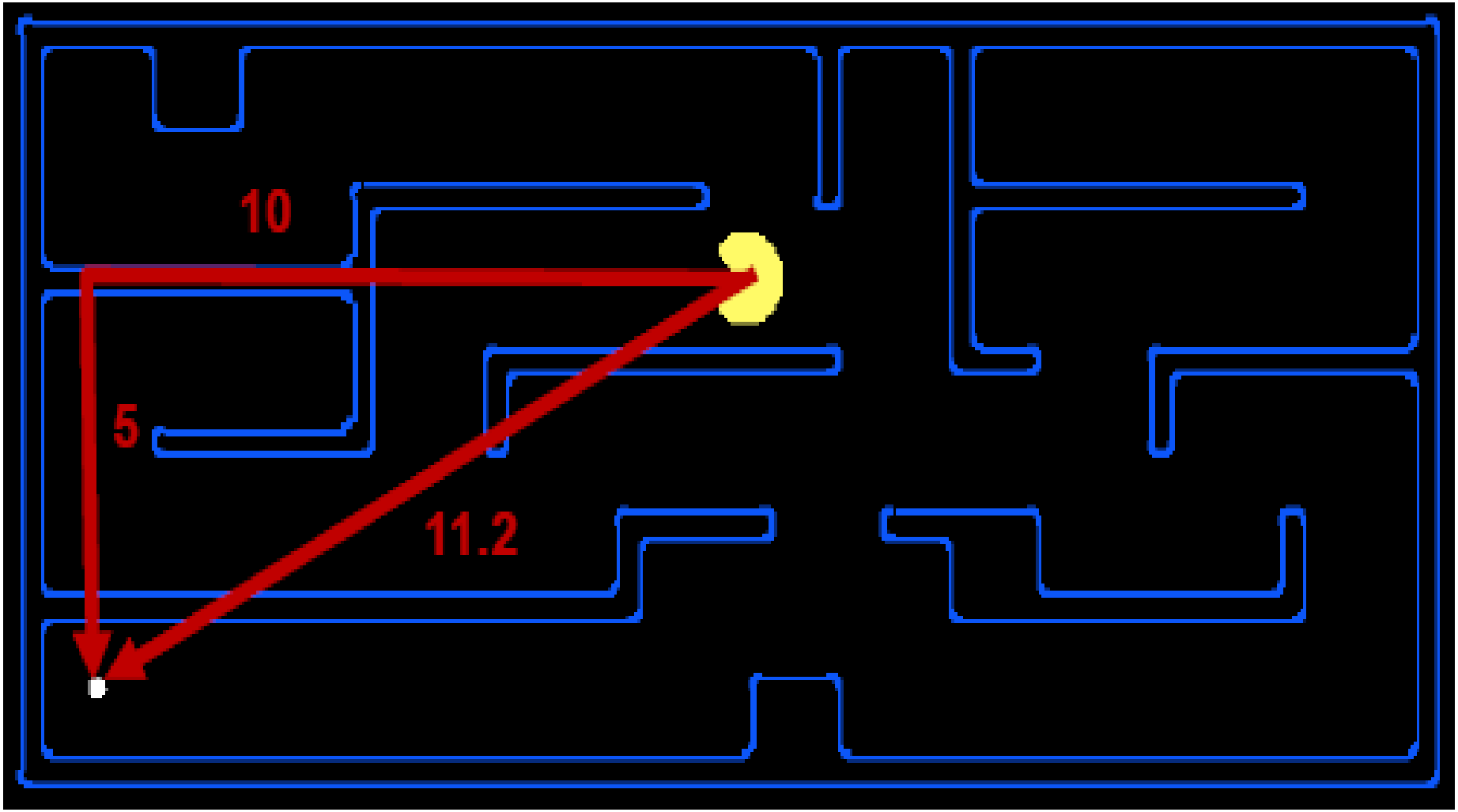
Notation

$h(s) \geq 0$, for any state s
$h(s) = 0$, implies that s is a goal state
$h(s) = \infty$, implies that s is a state not allowing a goal to be achieved

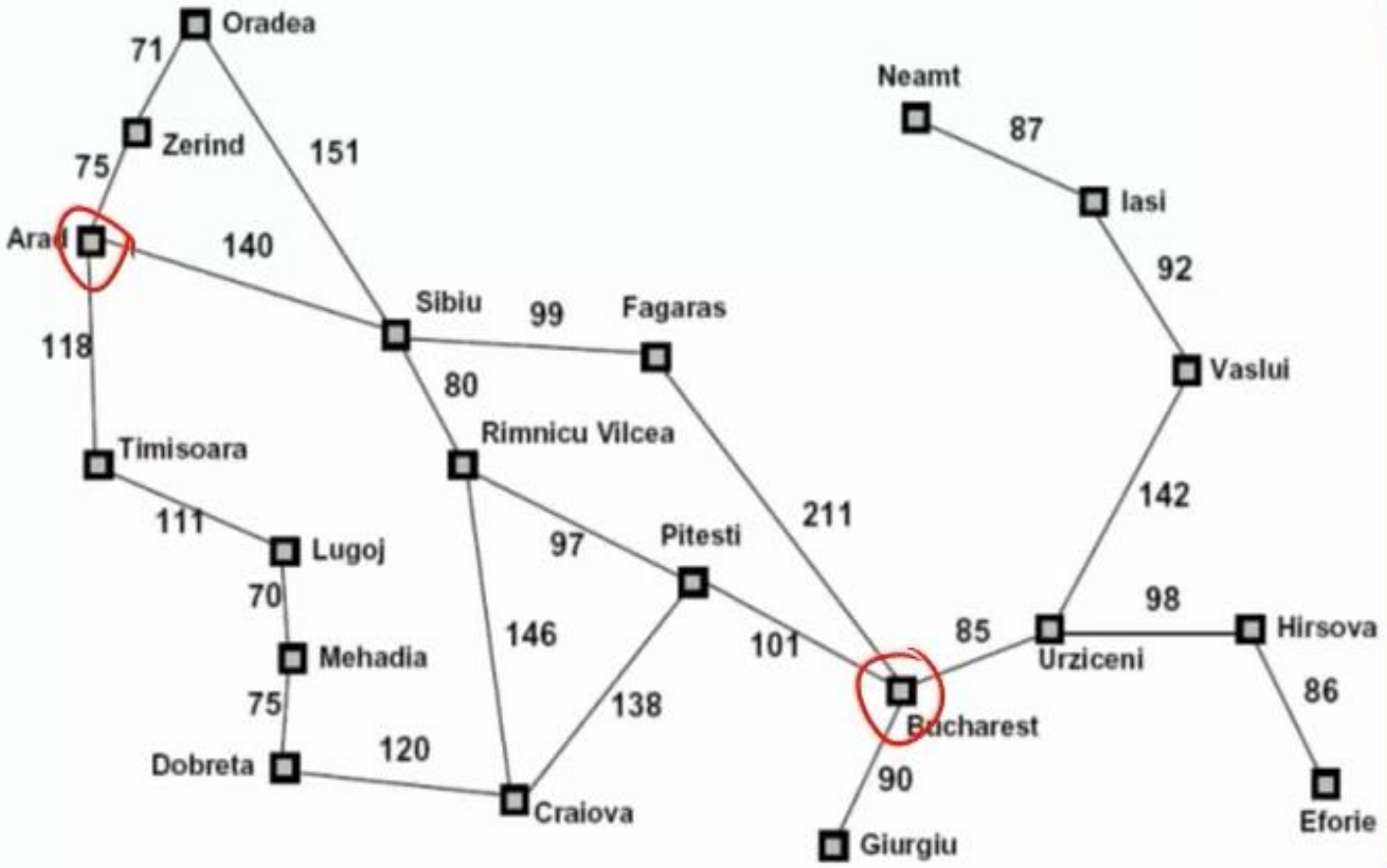
Example: Heuristic Function

Heuristic: the number of the largest pancake that is still out of place



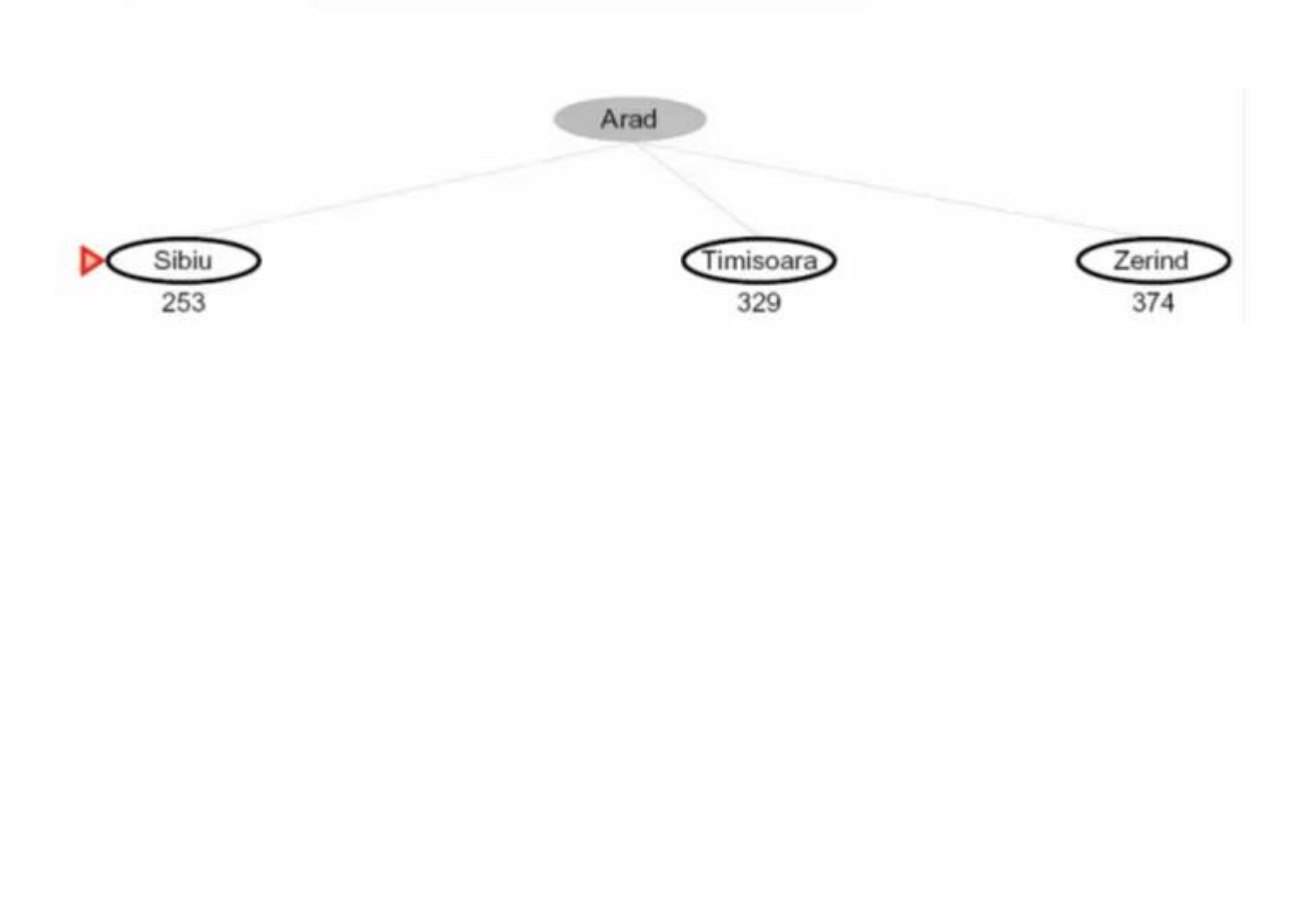

10
5
11.2

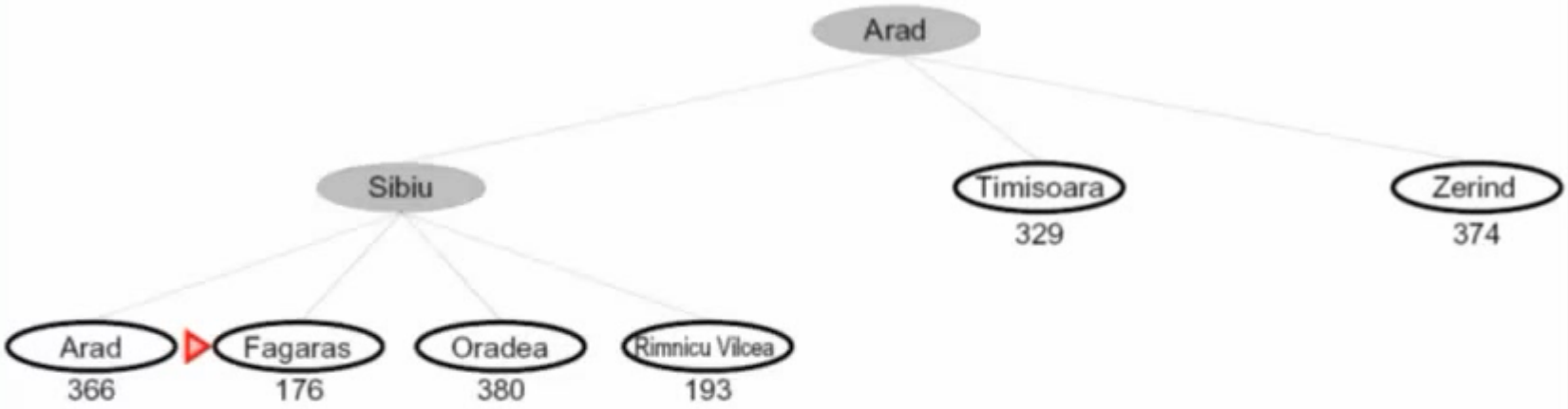
Heuristic Search(Gready Search)

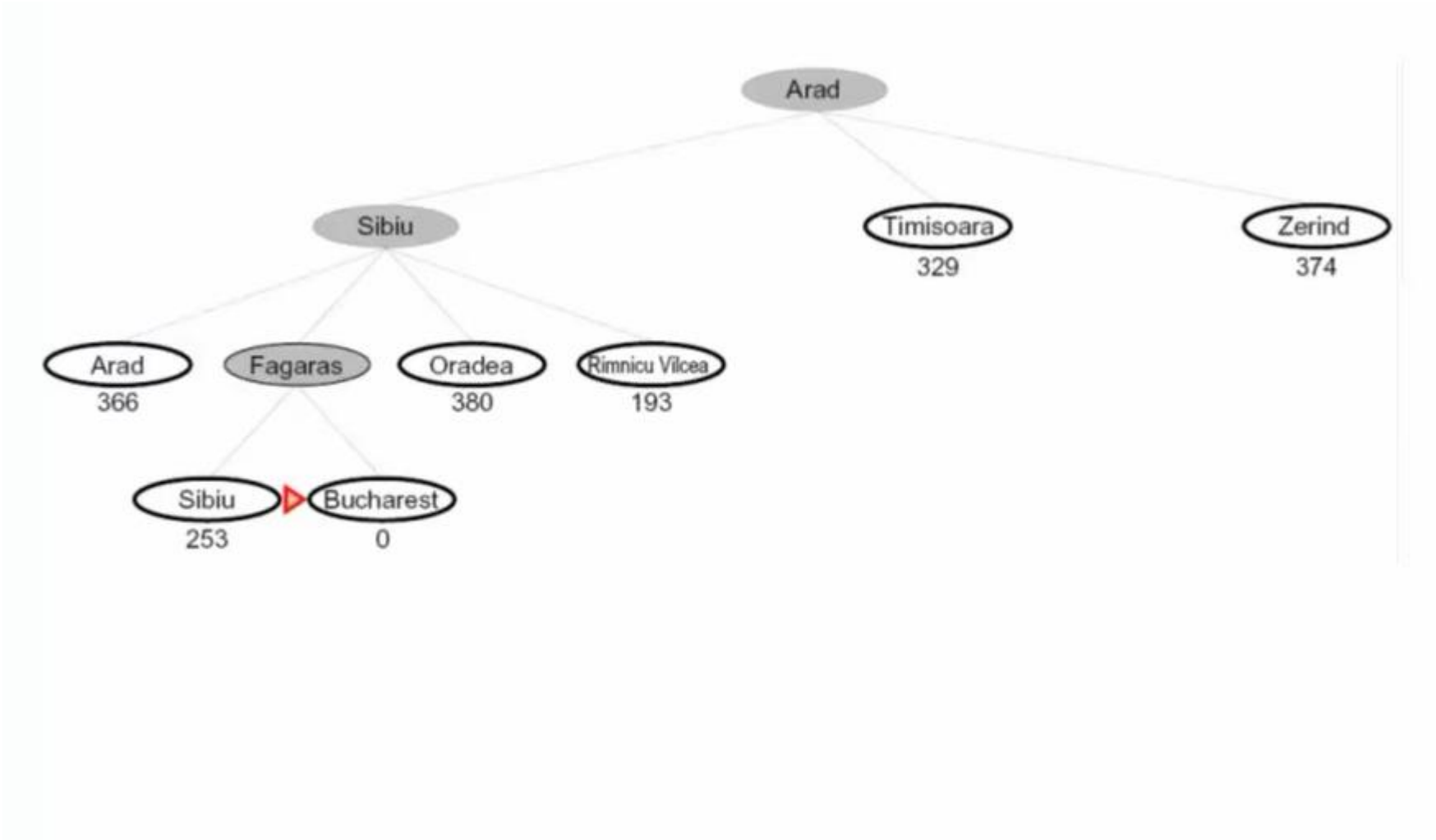


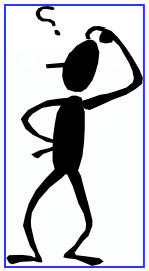
Straight−line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

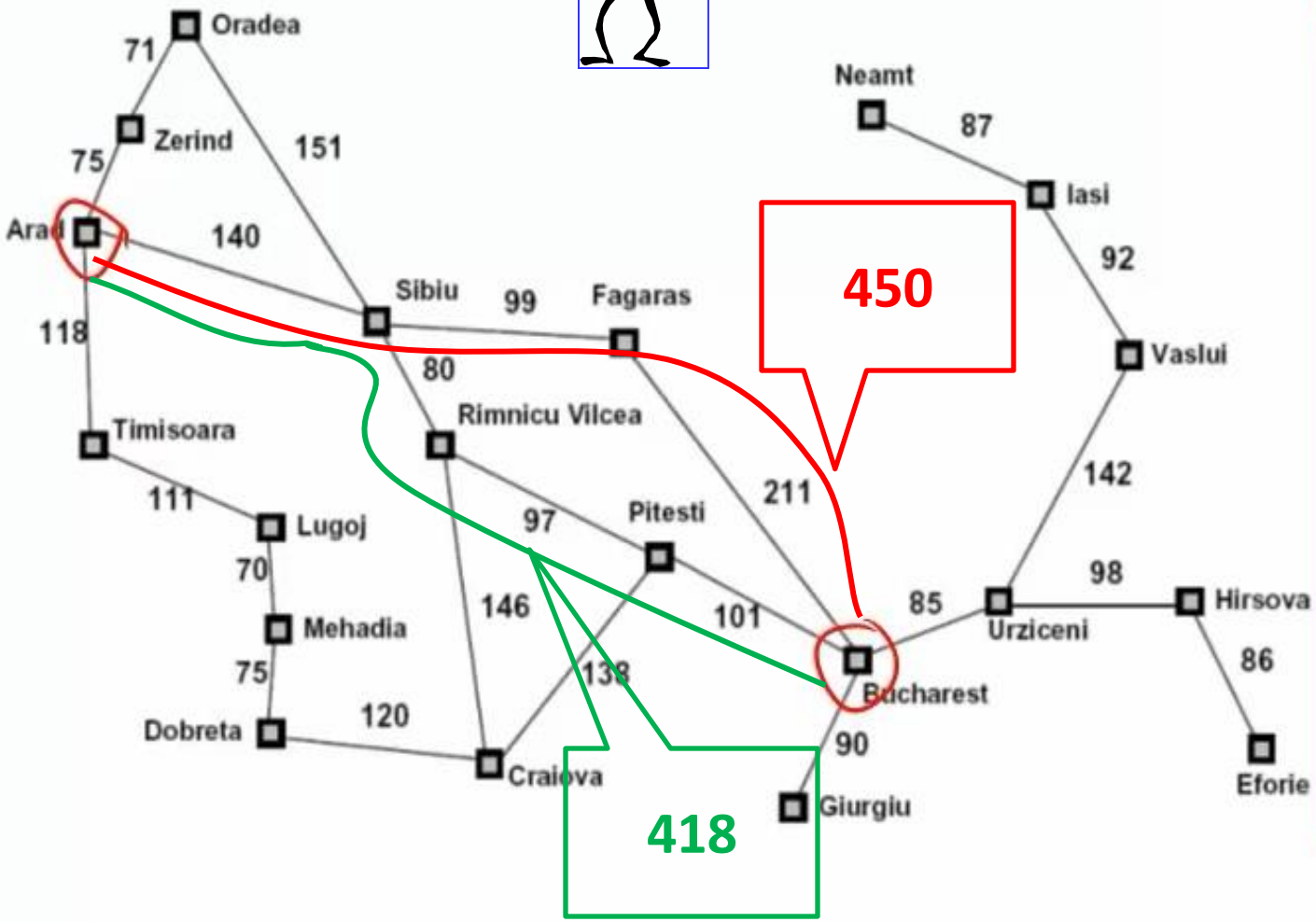
h(x)


Arad
Sibiu
253
Timisoara
329
Zerind
374


Arad
Sibiu
Timisoara
329
Zerind
374
Arad
366
Fagaras
176
Oradea
380
Rimnicu Vilcea
193


Arad
Sibiu
Timisoara
329
Zerind
374
Arad
366
Fagaras
Oradea
380
Rimnicu Vilcea
193
Sibiu
253
Bucharest
0



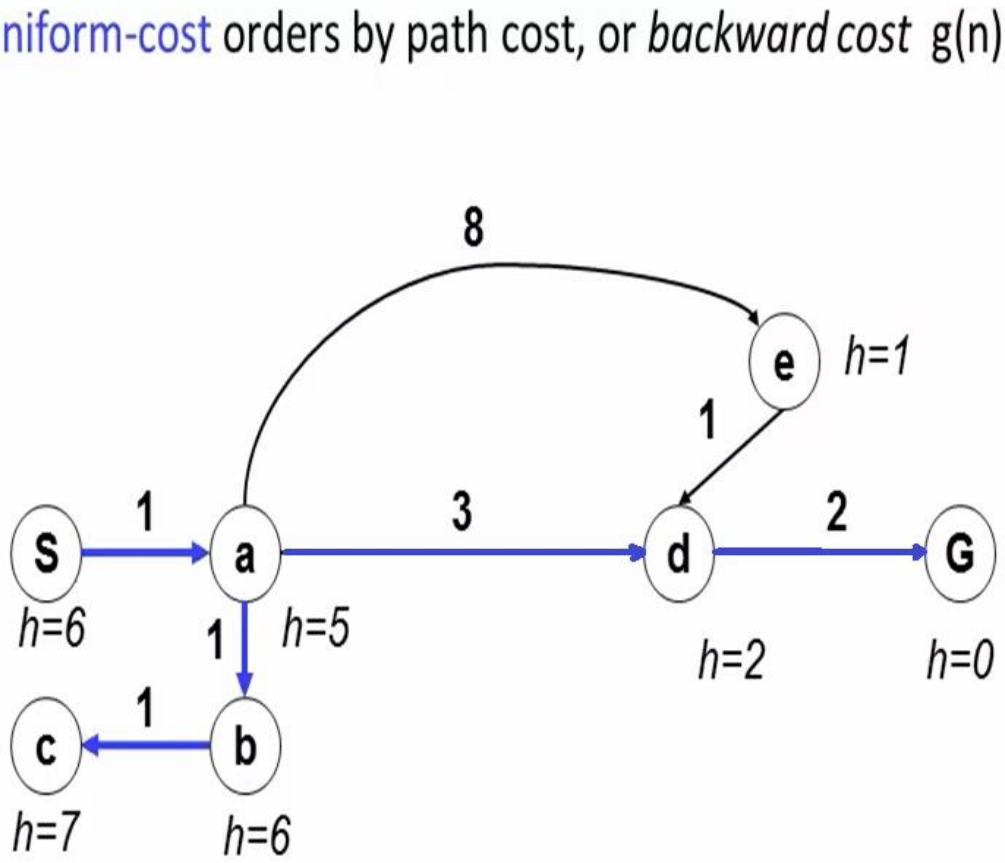


Straight−line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

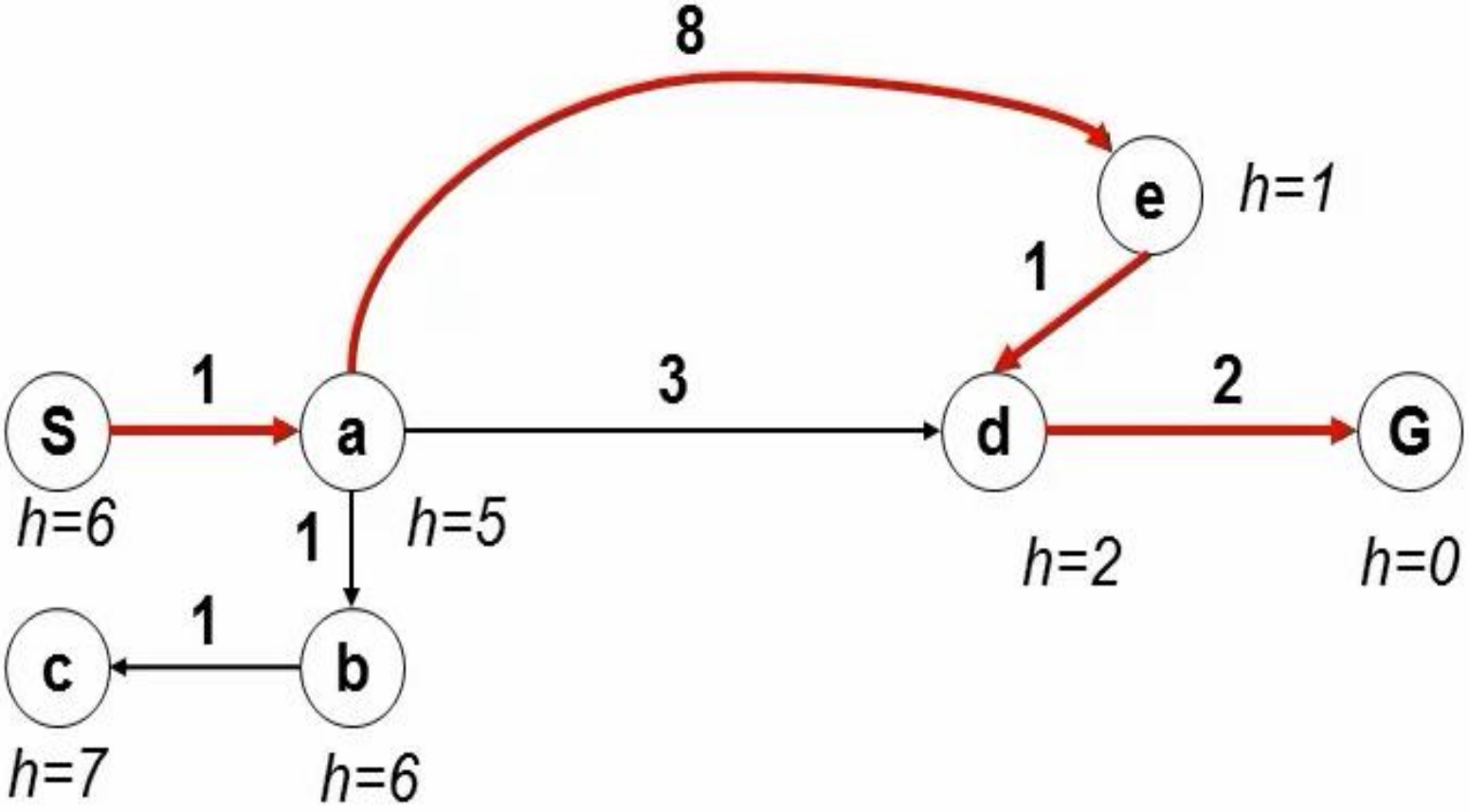
h(x)

L’algorithmes A*

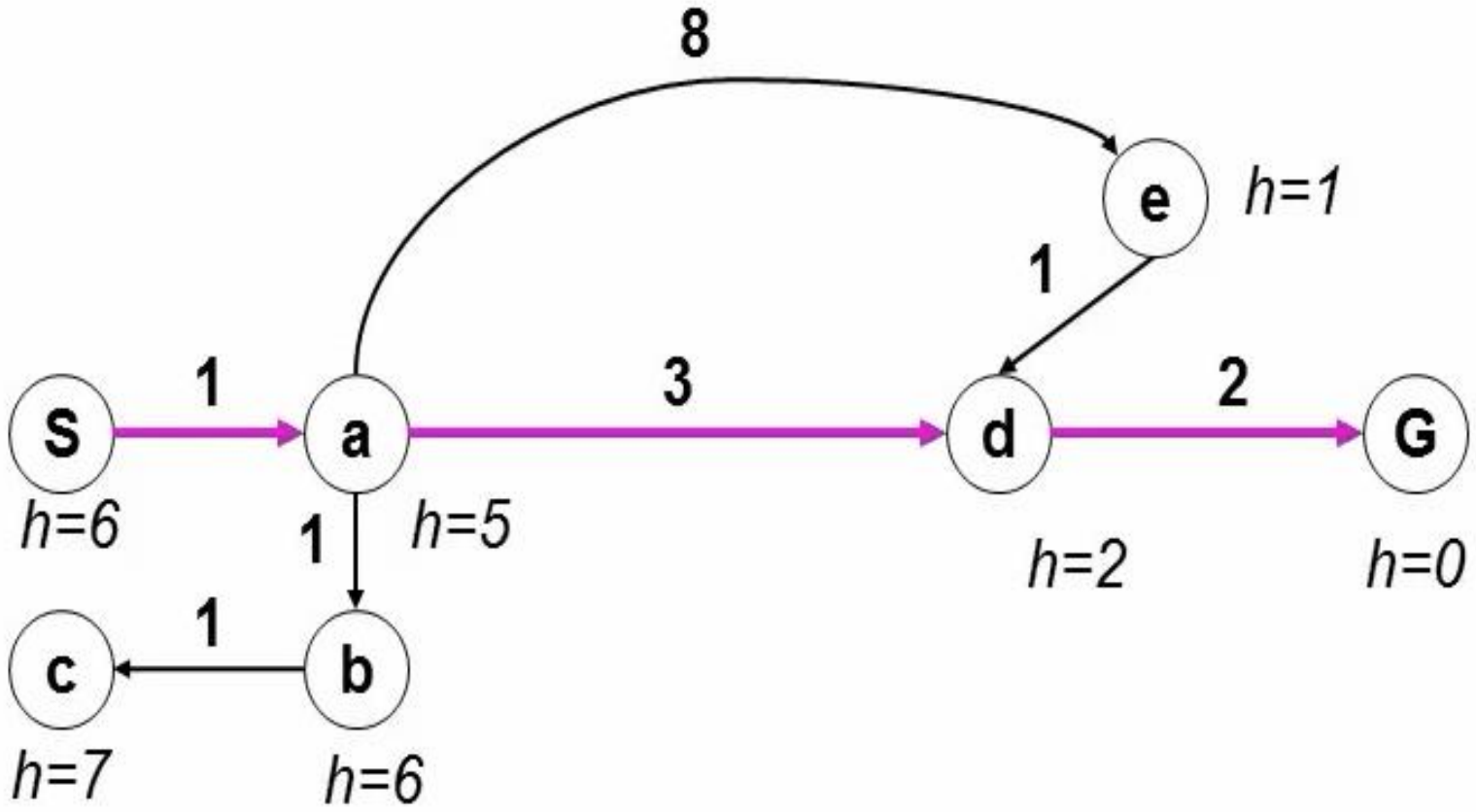
Combination of uniform cost search and heuristics (UCS + Gready search)



- Uniform-cost orders by path cost, or *backward cost* g(n)
- Greedy orders by goal proximity, or *forward cost* h(n)

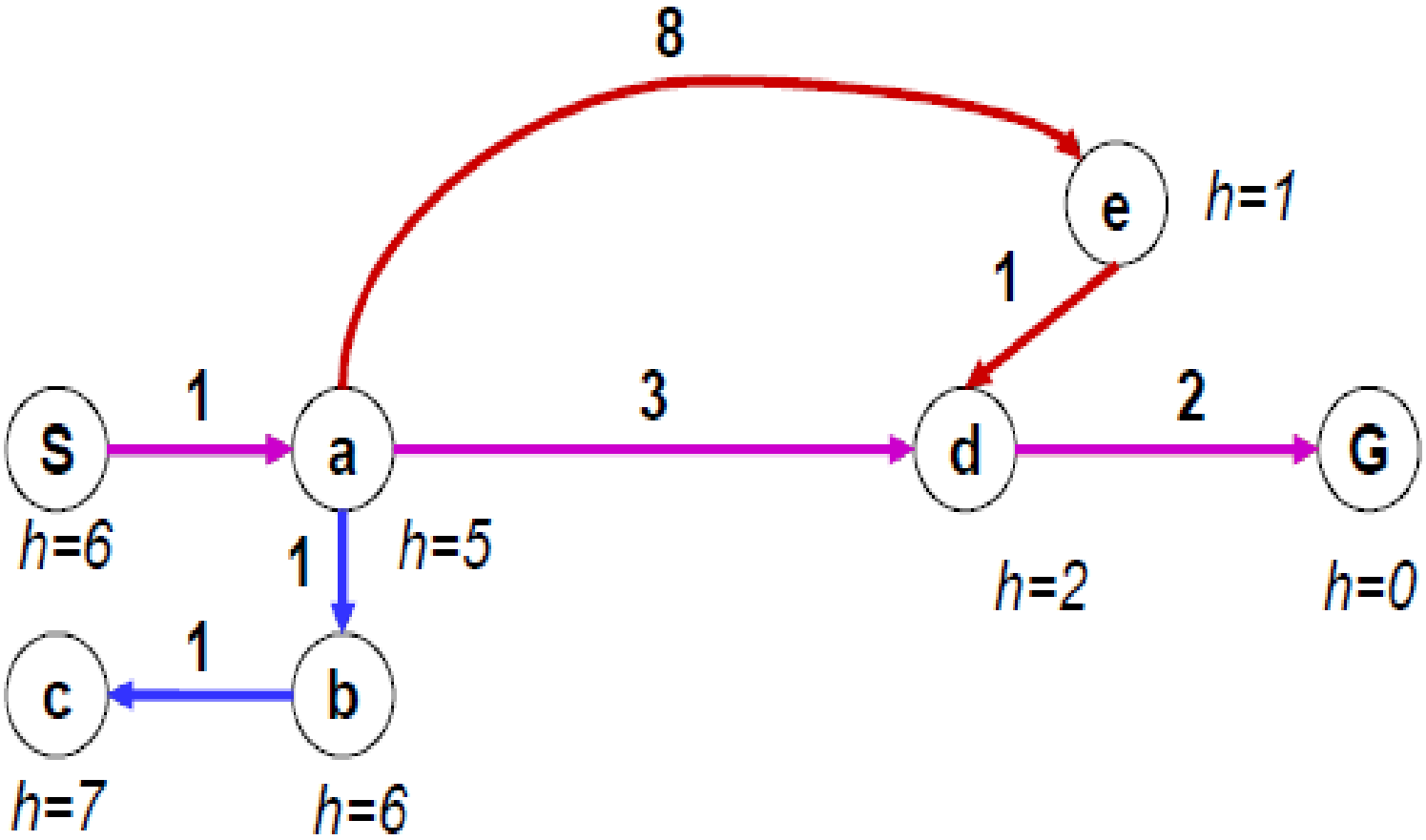


- Uniform-cost orders by path cost, or *backward cost* g(n)
- Greedy orders by goal proximity, or *forward cost* h(n)



- A* Search orders by the sum: f(n) = g(n) + h(n)

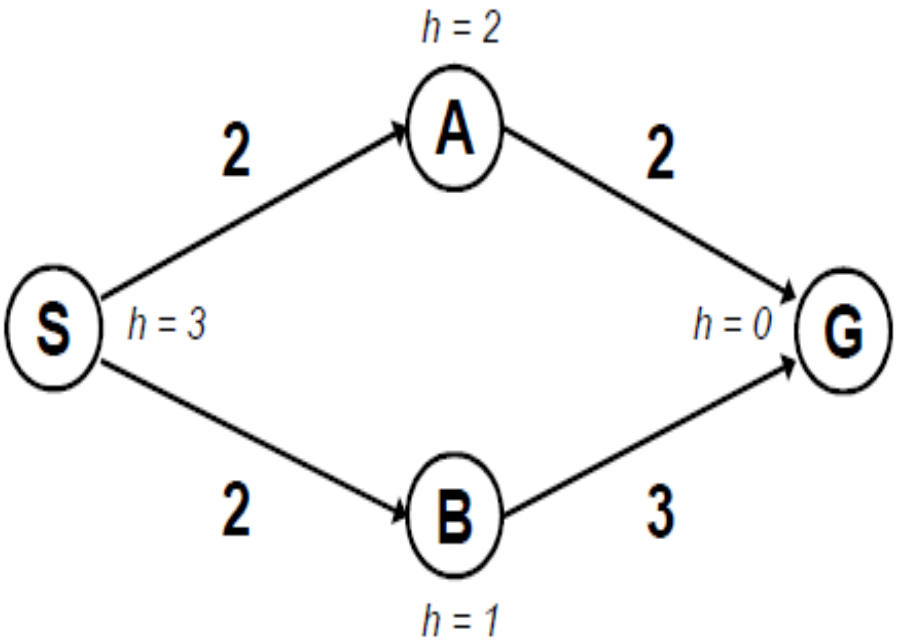
- Uniform-cost orders by path cost, or *backward cost* g(n)
- Greedy orders by goal proximity, or *forward cost* h(n)



- A* Search orders by the sum: f(n) = g(n) + h(n)

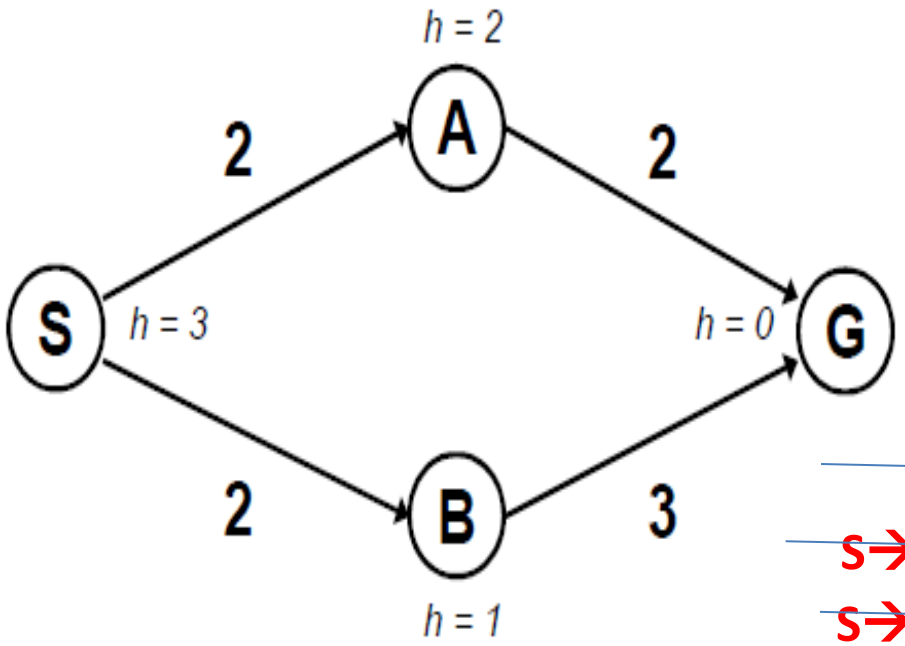
When should A* terminate?

Do we stop searching if we find (stack) a Goal?



When should A* terminate?

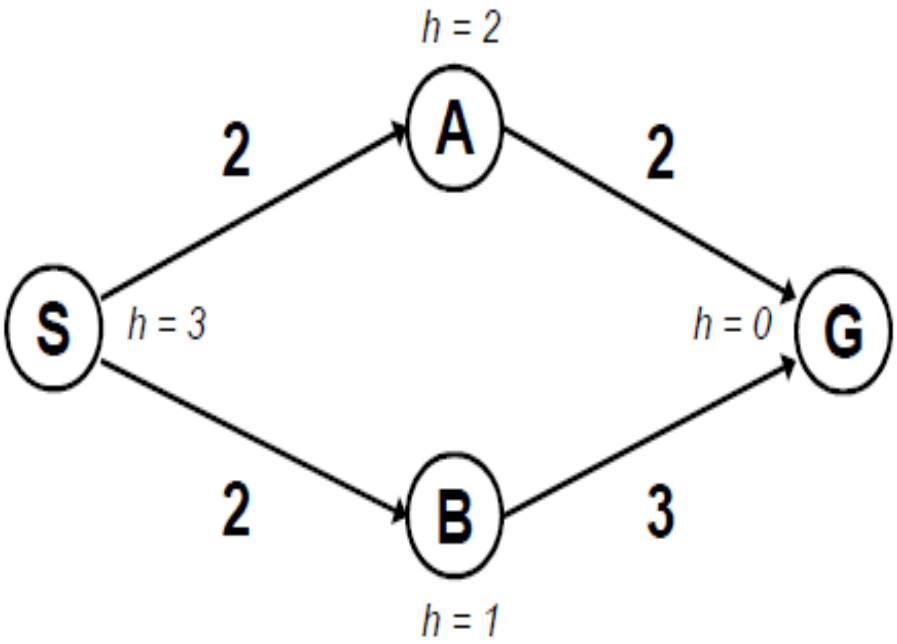
Do we stop searching if we find (stack) a Goal?



	.g	.h	
S	0	3	3
S→A	2	2	4
S→B	2	1	3
S→B→G	5	0	5
S→A→G	4	0	4

When should A* terminate?

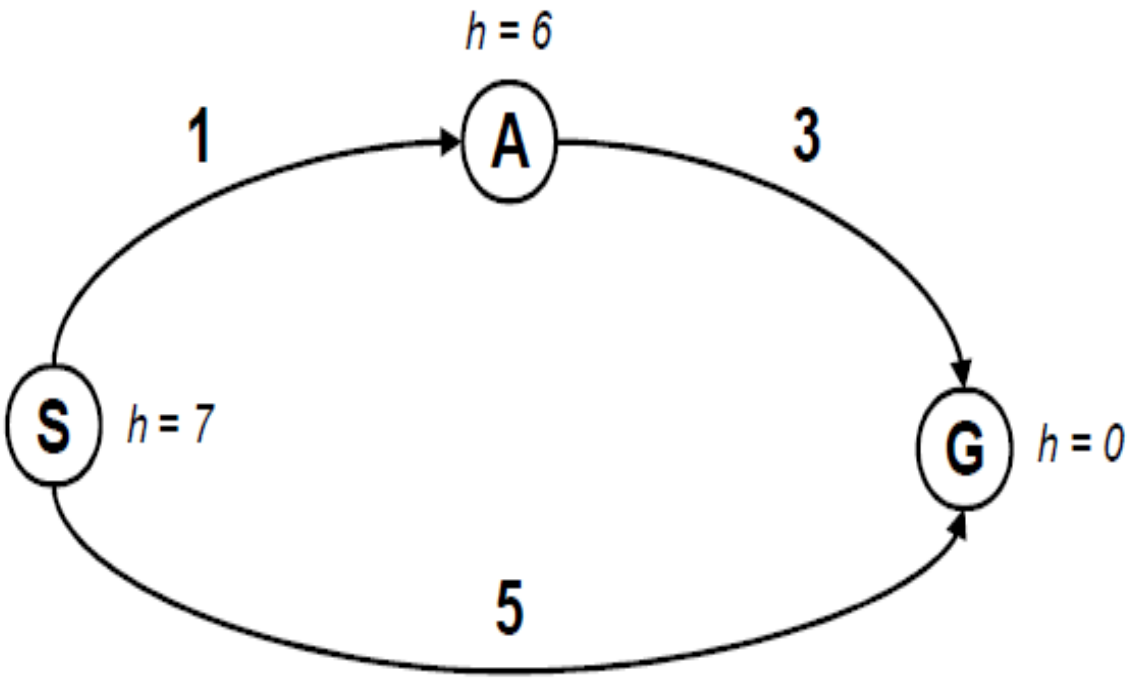
Do we stop searching if we find (stack) a Goal?



Stop: if and only if you unstack a Goal and don't when you stack it.

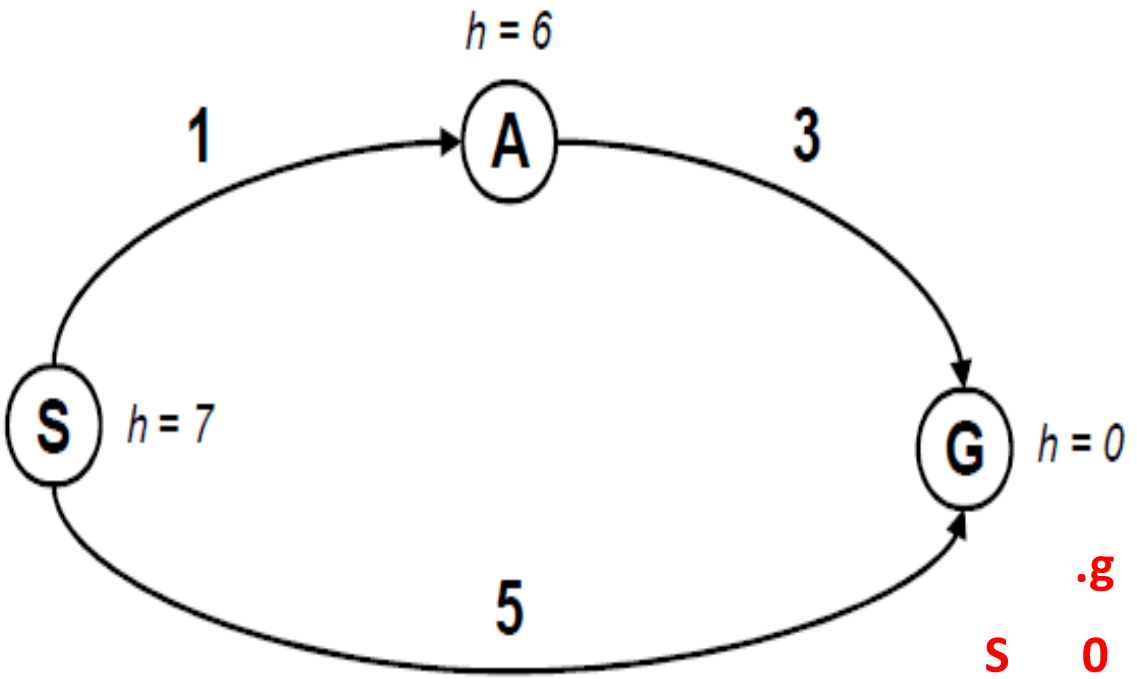
Attention !!!

Is A* Optimal?



Attention !!!

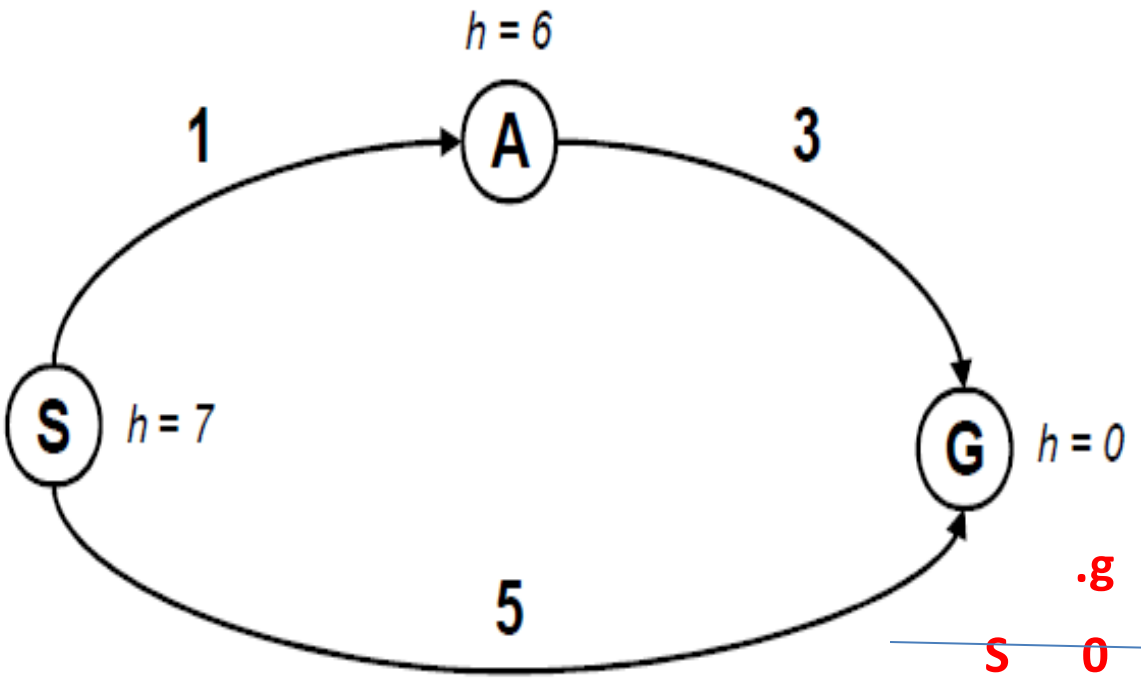
Is A* Optimal?



	.g	.h	
S	0	7	7

Attention !!!

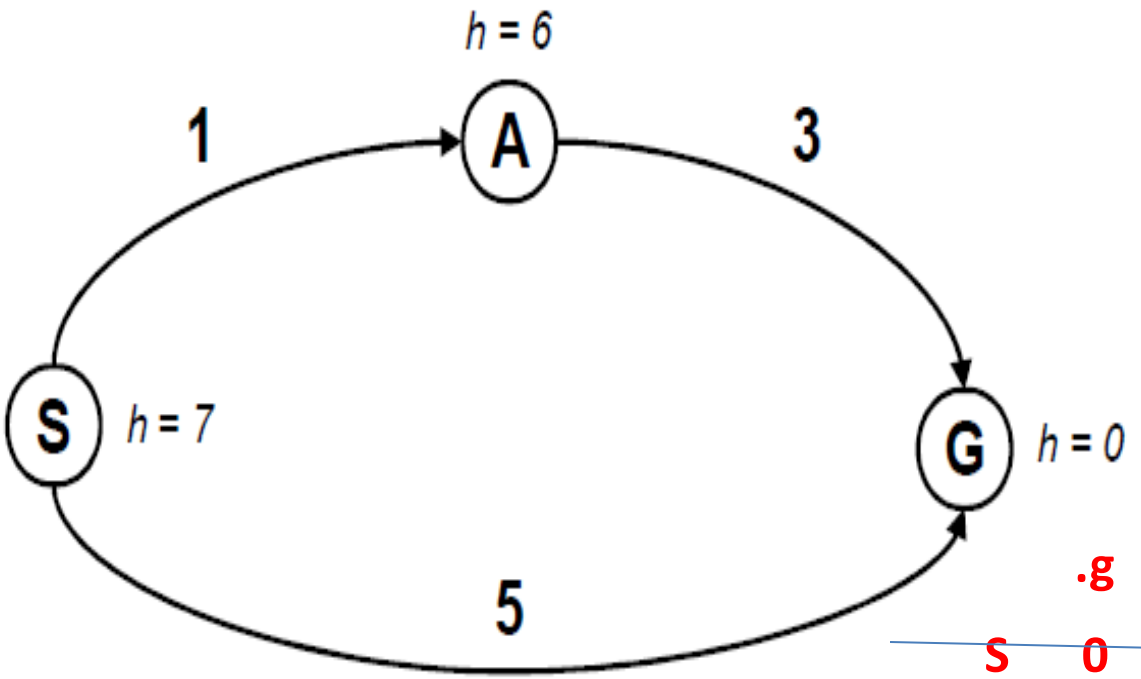
Is A* Optimal?



	.g	.h	
S	0	7	7
S→A	1	6	7
S→G	5	0	5

Attention !!!

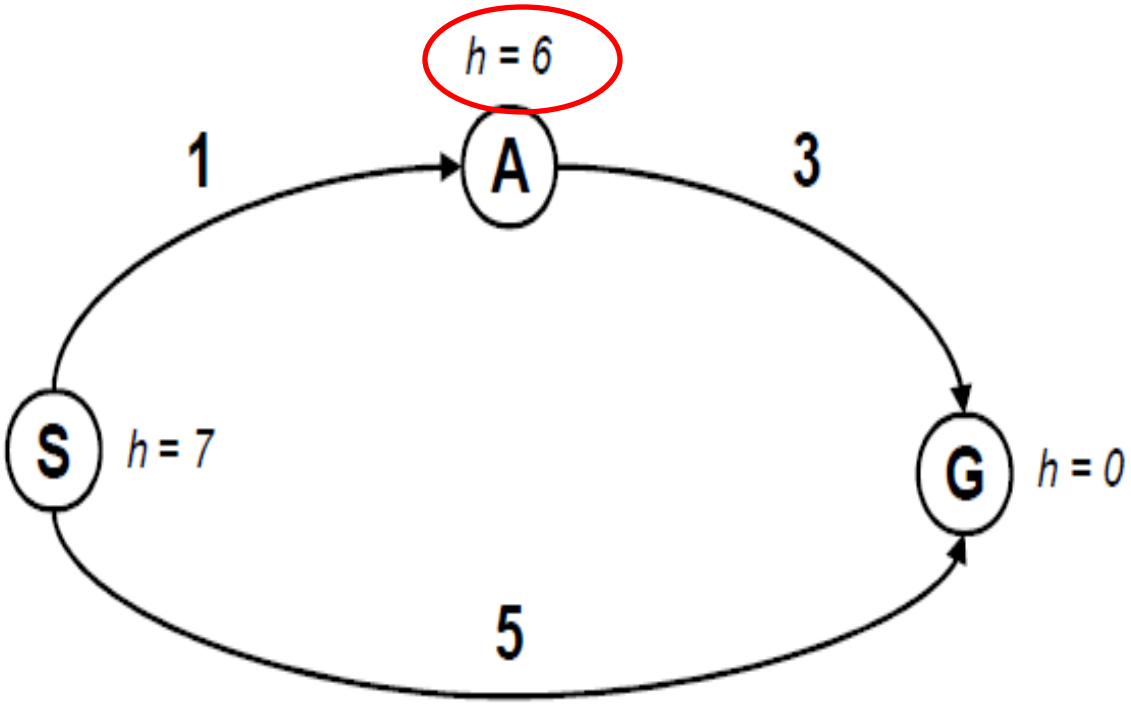
Is A* Optimal?



	.g	.h	
S	0	7	7
S→A	1	6	7
S→G	5	0	5

Attention !!!

Is A* Optimal?



The search ends badly because of the heuristic

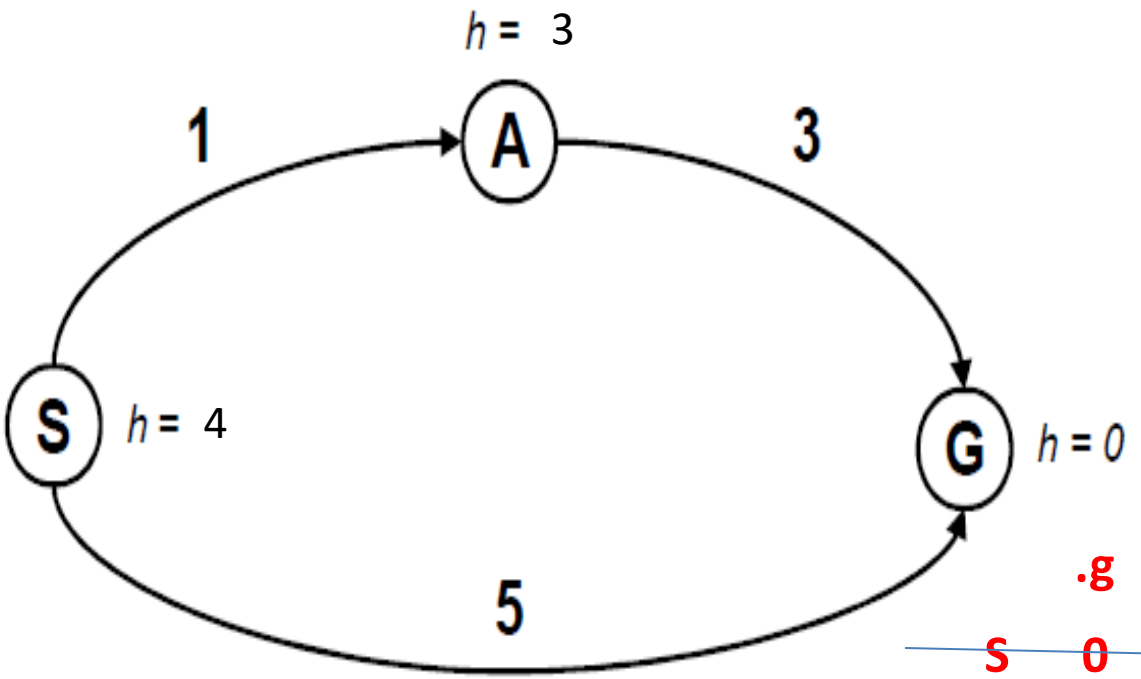
Admissibility of A* Algorithm

The heuristic function h(n) is admissible
if **h(n)** is **never larger** than **h*(n)**
or if **h(n)** is always less or equal to the true value.

h(n) ≤ h*(n) *∴ Underestimation*
h(n) ≥ h*(n) *∴ Overestimation*

Attention !!!

Is A* Optimal?

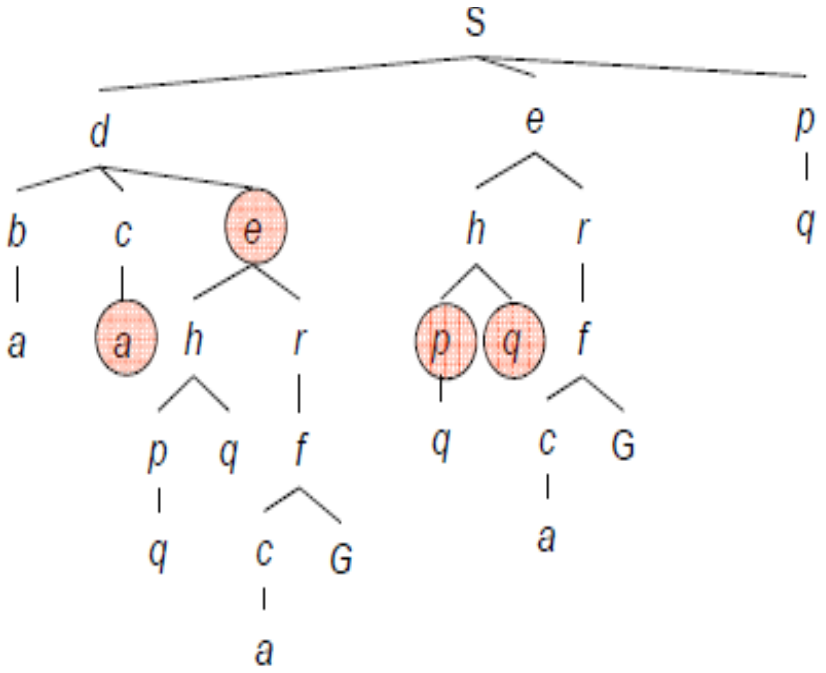


	.g	.h	
S	0	4	4
S→A	1	3	4
S→G	5	0	5
S->A→G	4	0	4

Graph Search

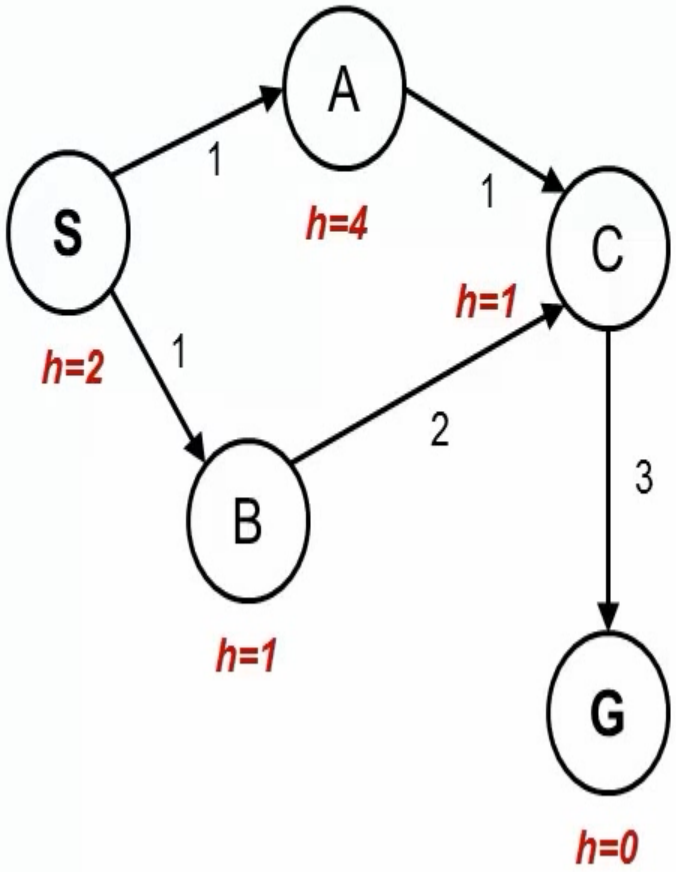
In BFS it is useful to visit the node only once, to improve the search.

Use a set to mark visited nodes.



A* Graph Search Gone Wrong?

State space graph

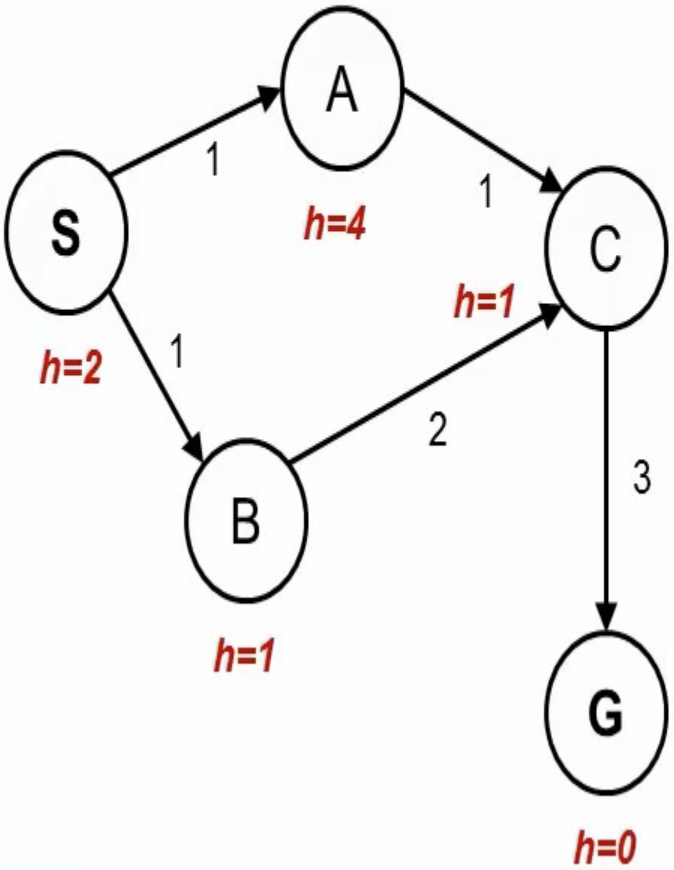


Search tree

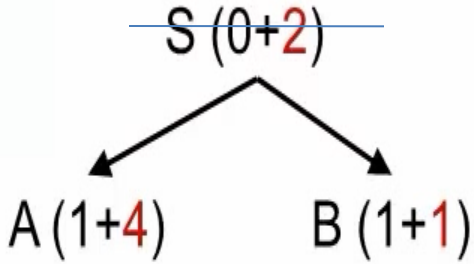
S (0+2)

A* Graph Search Gone Wrong?

State space graph



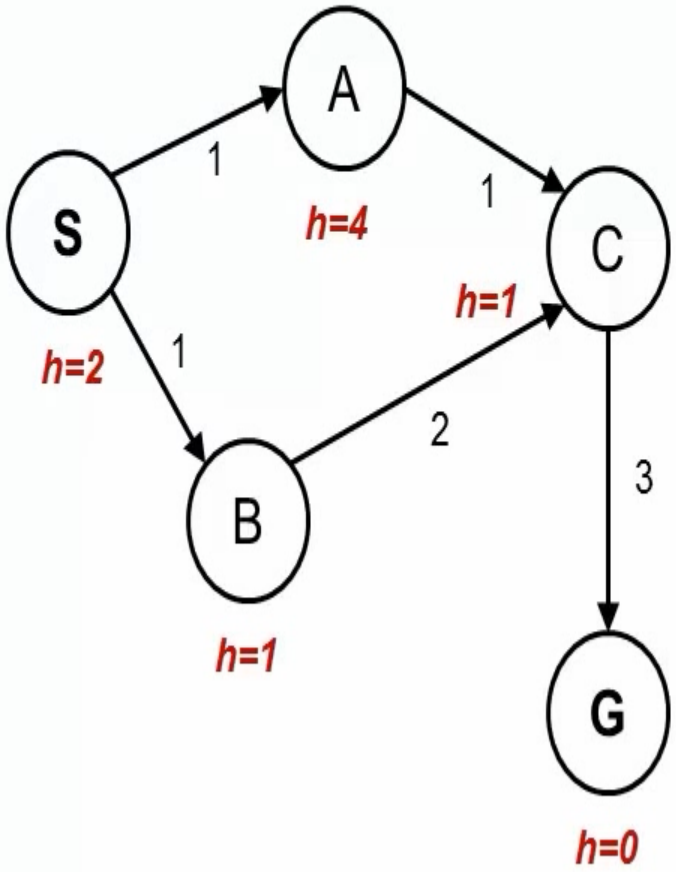
Search tree



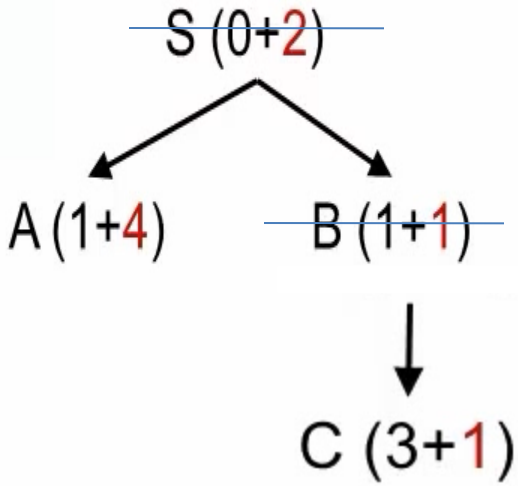
S

A* Graph Search Gone Wrong?

State space graph



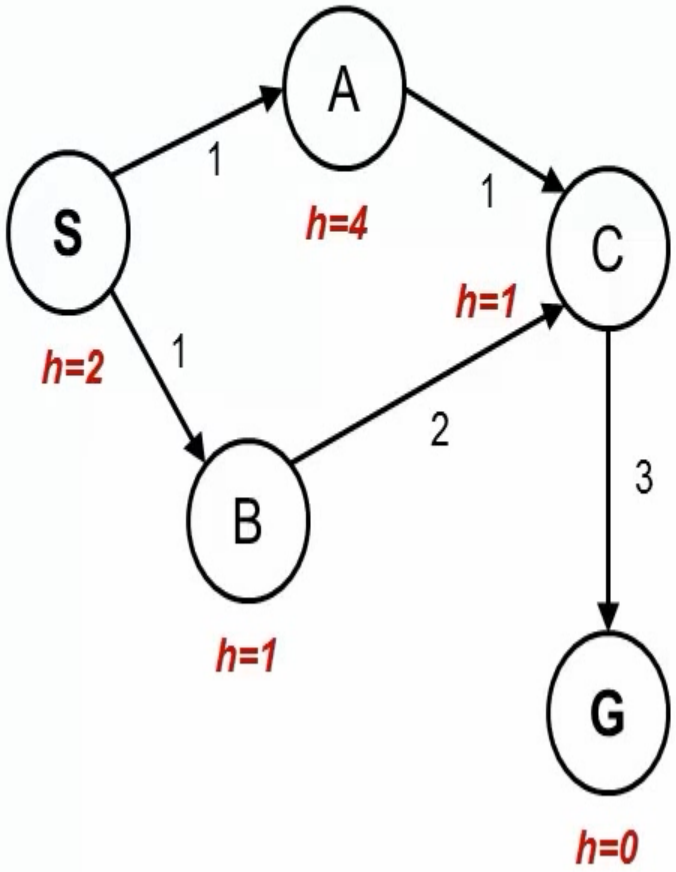
Search tree



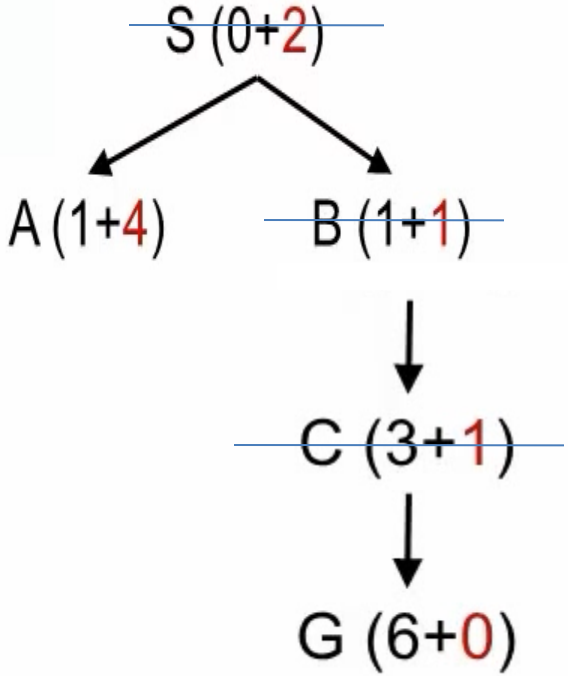
S
B

A* Graph Search Gone Wrong?

State space graph



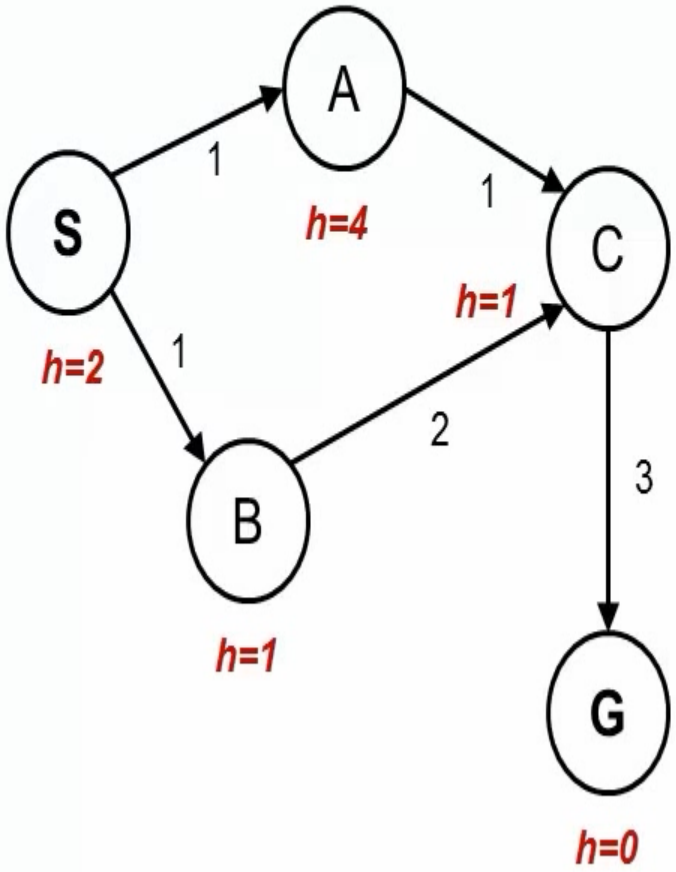
Search tree



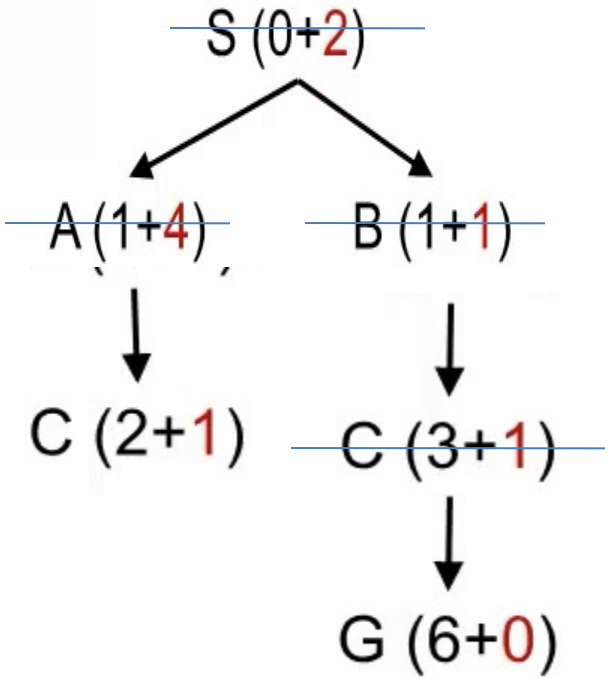
S
B
C

A* Graph Search Gone Wrong?

State space graph



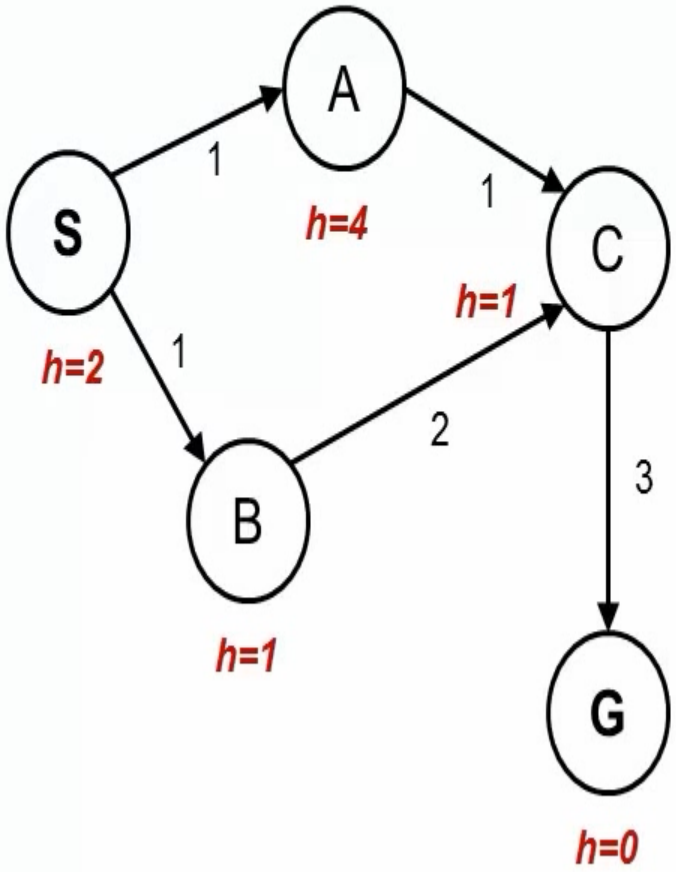
Search tree



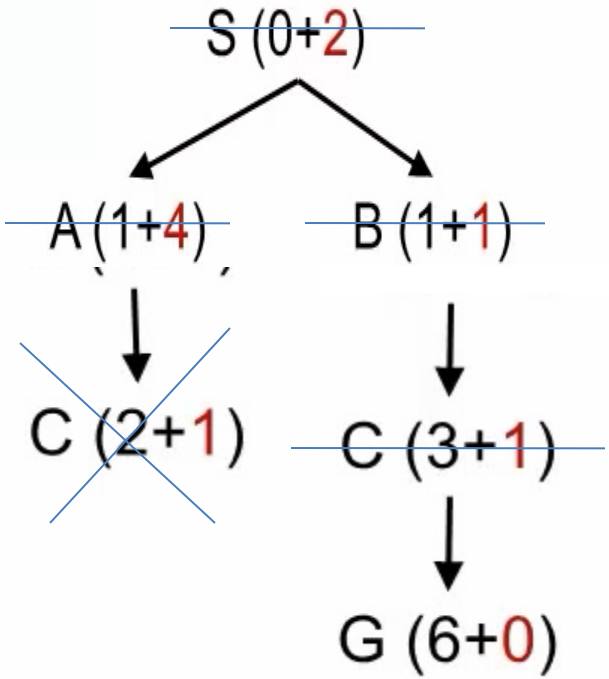
S
B
C
A

A* Graph Search Gone Wrong?

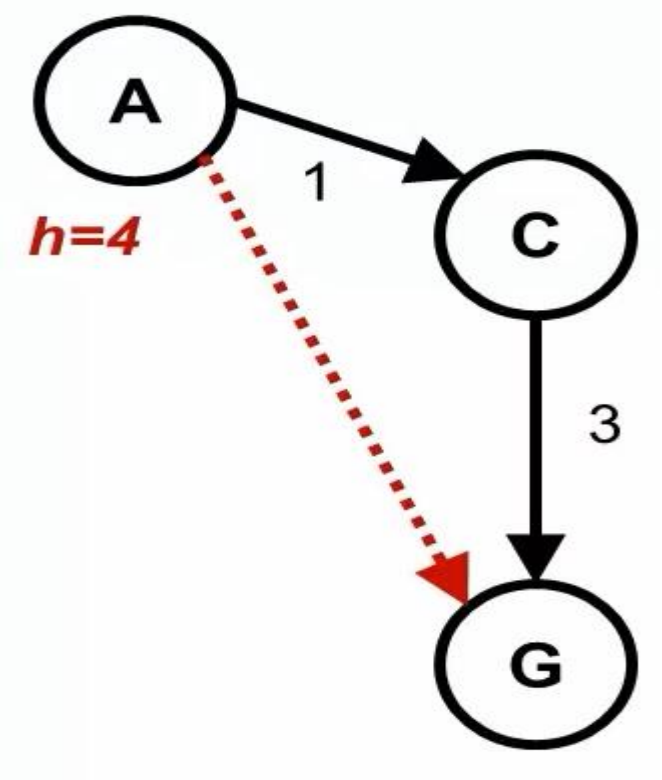
State space graph



Search tree



S
B
C
A



Admissibility of the heuristic search

H(A) <= the real cost of A to G

Consistency

The heuristic cost <= the real cost

For each arc