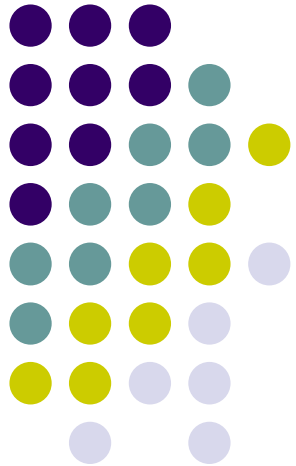
ARTIFICIAL INTELLIGENCE

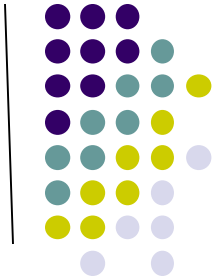
Informed search algorithms

Chapter 4



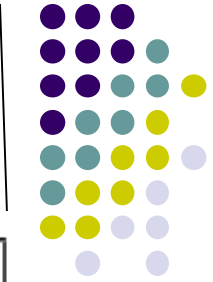
Assoc. Prof. Zeynep ORMAN

Outline



- Best-first search
  - Greedy best-first search
  - A* search
- Heuristics
- Local search algorithms
  - Hill-climbing search
  - Simulated annealing search
  - Local beam search
  - Genetic algorithms

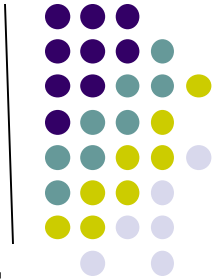
Review: Tree search



```
function TREE-SEARCH( problem, strategy) returns a solution, or failure
    initialize the search tree using the initial state of problem
    loop do
        if there are no candidates for expansion then return failure
        choose a leaf node for expansion according to strategy
        if the node contains a goal state then return the corresponding solution
        else expand the node and add the resulting nodes to the search tree
```

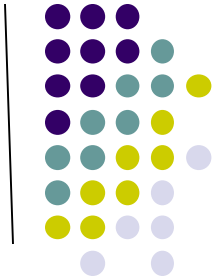
- A search strategy is defined by picking the order of node expansion.
- Uninformed search strategies can find solutions to problems by generating new states and testing them against the goal.
- But, these algorithms are inefficient in most cases.

Informed search algorithms



- Informed search strategy uses problem-specific knowledge beyond the definition of the problem itself.
- Informed search can find solutions more efficiently than an uninformed search.
- Best-first search is an instance of the general TREE-SEARCH or GRAPH-SEARCH algorithm.
  - a node is selected for expansion based on an evaluation function, $f(n)$.
  - The node with the lowest evaluation is selected for expansion, because the evaluation measures distance to the goal.

Best-first search

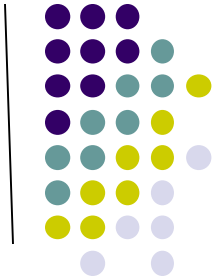


- There are various Best-first search algorithms with different evaluation functions.
- A key component of these algorithms is a heuristic function:

  h(n) = estimated cost of the cheapest path from node *n* to a goal node.

- if *n* is a goal node, then *h(n)=0*.

Best-first search



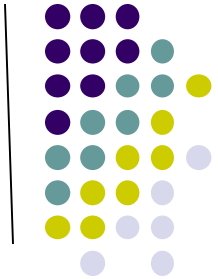
- Idea: use an evaluation function *f(n)* for each node
  - estimate of "desirability"
  - → Expand most desirable unexpanded node

- Implementation:

  Order the nodes in fringe in decreasing order of desirability

- Special cases:
  - greedy best-first search
  - A* search

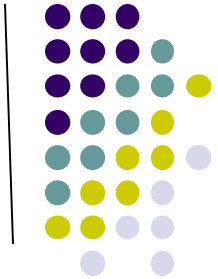
Best-first search



1. Start with OPEN containing just the initial state.
2. Until a goal is found or there are no nodes left on OPEN do:
   (a) Pick the best node on OPEN.
   (b) Generate its successors.
   (c) For each successor do:
      i. If it has not been generated before, evaluate it, add it to OPEN, and record its parent.

      ii. If it has been generated before, change the parent if this new path is better than the previous one. In that case, update the cost of getting to this node and to any successors that this node may already have.

Greedy best-first search



- Greedy best first search tries to expand the node that is closest to the goal.
- It evaluates nodes by using just the heuristic function: $f(n) = h(n)$ (heuristic)
  - e.g., $h_{SLD}(n)$ = straight-line distance from n to Bucharest
  - one might estimate the cost of the cheapest path from n to Bucharest via the SLD.

Greedy best-first search

Arad	366	**Mehadia**	241
Bucharest	0	**Neamt**	234
Craiova	160	**Oradea**	380
Dobreta	242	**Pitesti**	100
Eforie	161	**Rimnicu Vilcea**	193
Fagaras	176	**Sibiu**	253
Giurgiu	77	**Timisoara**	329
Hirsova	151	**Urziceni**	80
Iasi	226	**Vaslui**	199
Lugoj	244	**Zerind**	374

Values of h_{SLD}-straight line distances to Bucharest

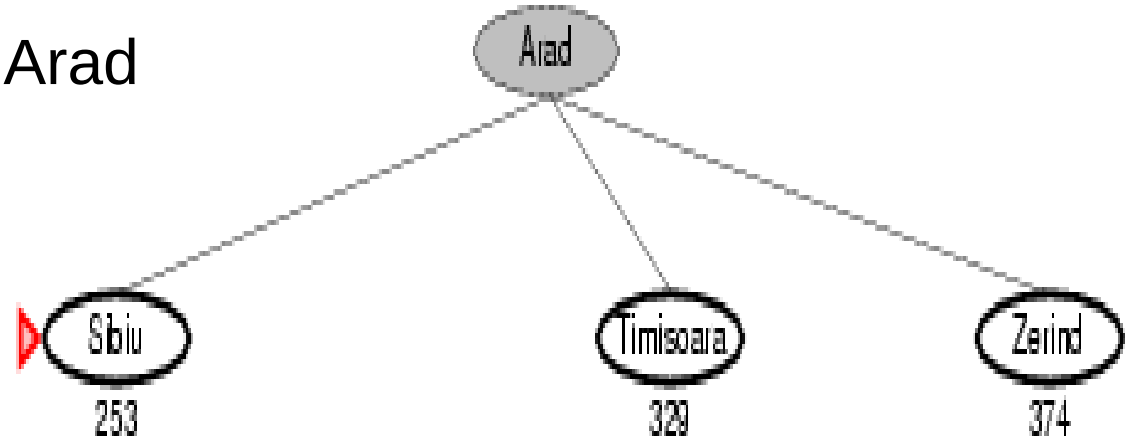
NOTE: the values of h_{SLD} cannot be computed from the problem description itself.

Greedy best-first search example

(a) The initial state

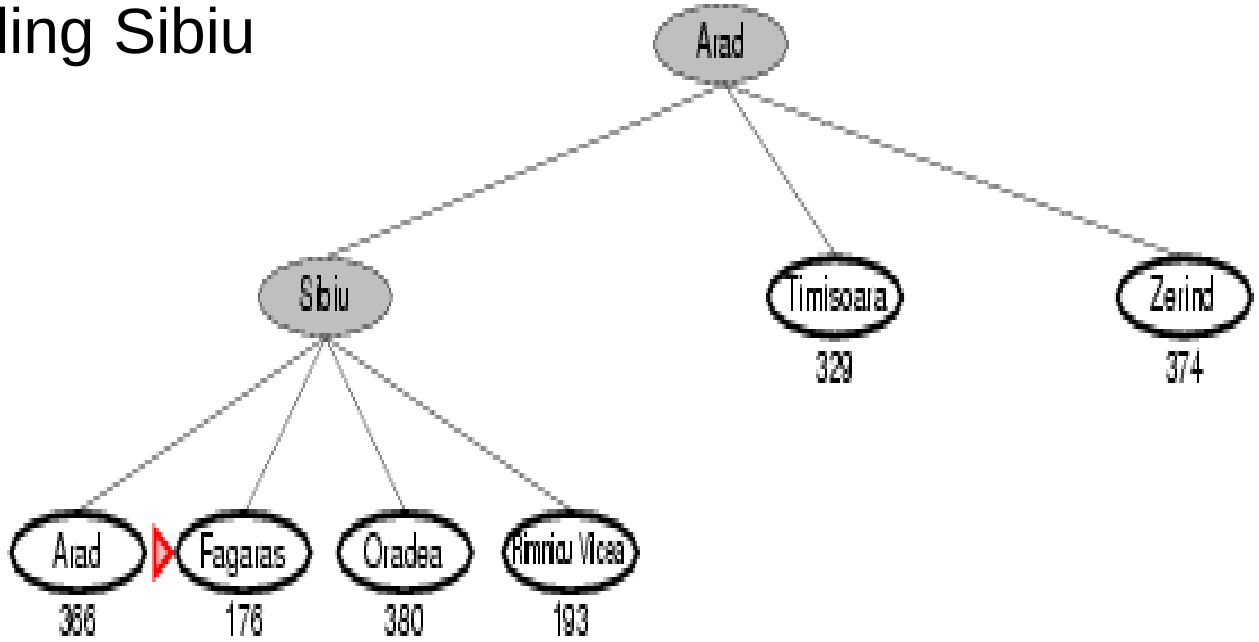
Greedy best-first search example

(b) After expanding Arad



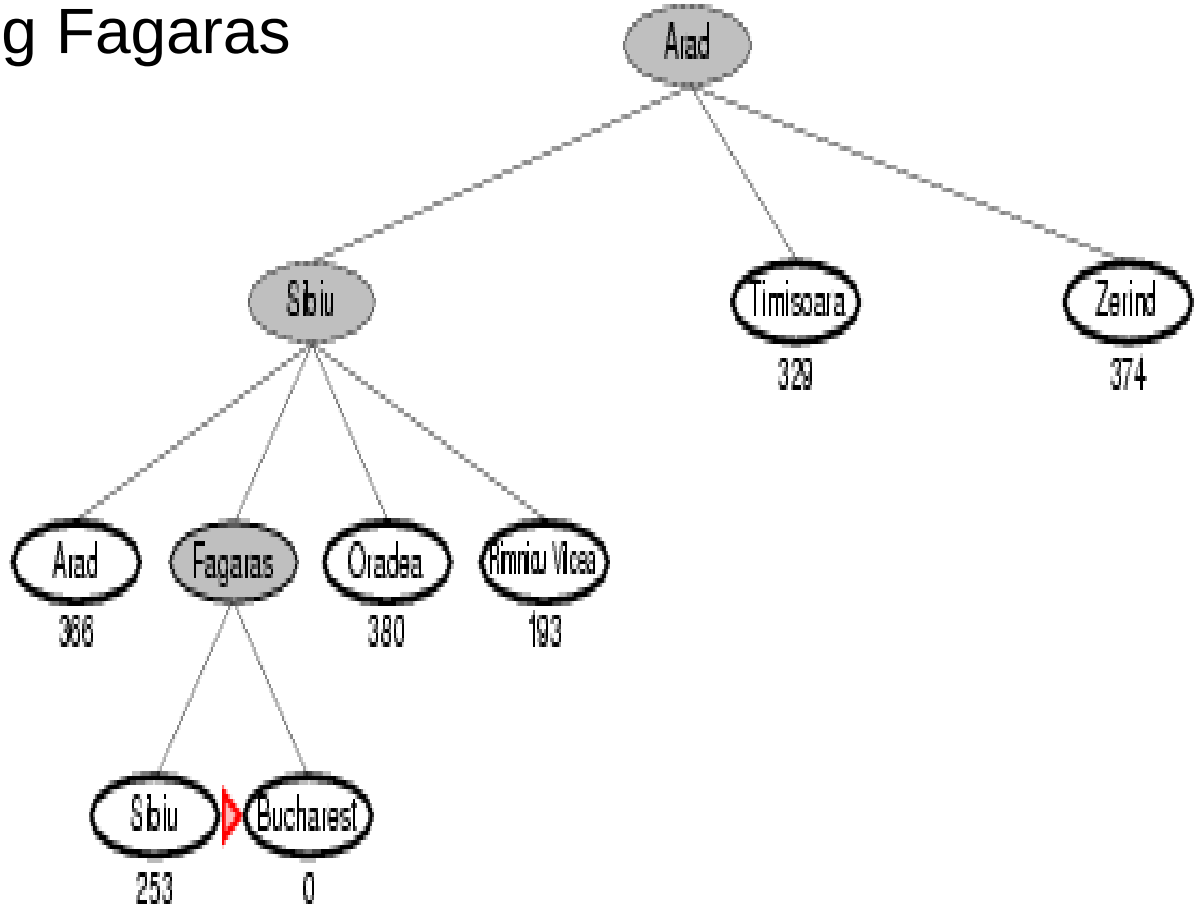
Greedy best-first search example

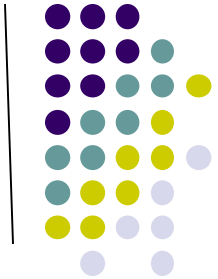
(c) After expanding Sibiu



Greedy best-first search example

(d) After expanding Fagaras

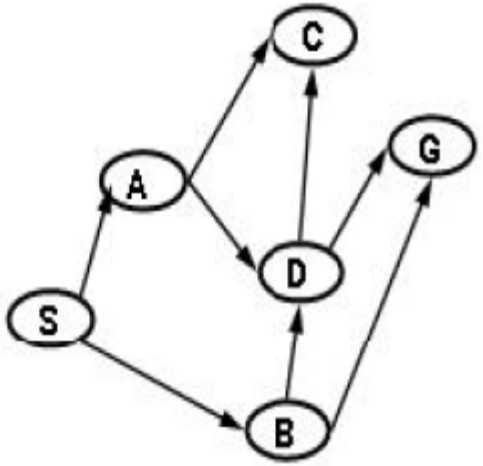




GREEDY BEST-FIRST SEARCH – ANOTHER EXAMPLE

Pick "best " (by heuristic value) element of Q;
Add path extensions anywhere in Q

	Q	Visited
1	(10 S)	S
2		
3		
4		
5		



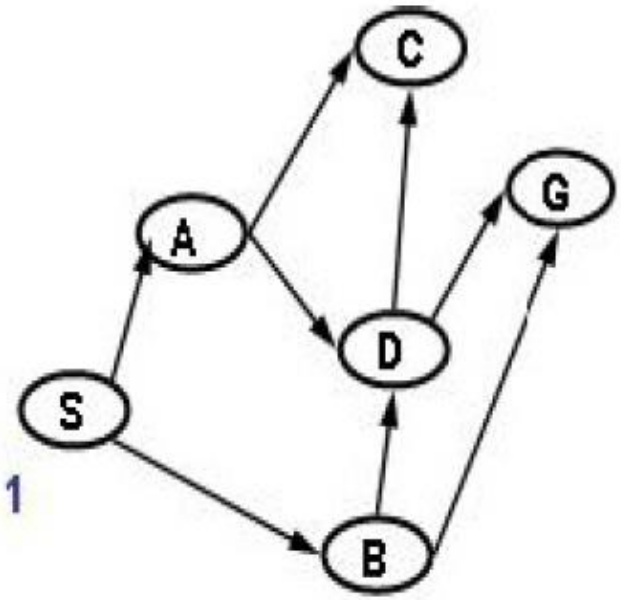
Heuristic Values

A=2 C=1 S=10

B=3 D=4 G=0

Added paths in blue; heuristic value of node's state is in front. We show the paths in reversed order; the node's state is the first entry.

	Q	Visited
1	(10 S)	S
2	(2 A S) (3 B S)	A,B,S
3		
4		
5		



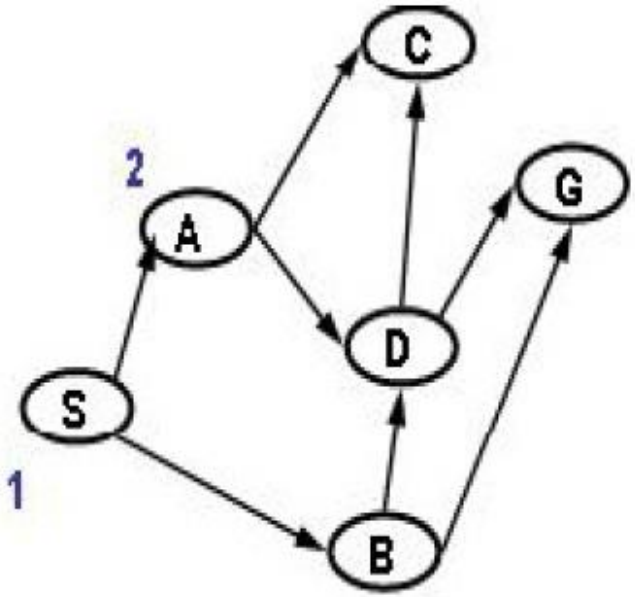
1

Heuristic Values

A=2 C=1 S=10

B=3 D=4 G=0

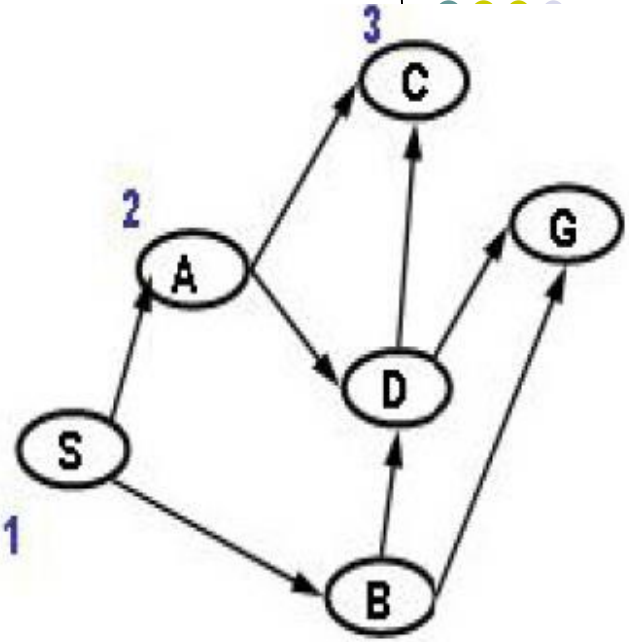
	Q	Visited
1	(10 S)	S
2	(2 A S) (3 B S)	A,B,S
3	(1 C A S) (3 B S) (4 D A S)	C,D,B,A,S
4		
5		



Heuristic Values

A=2	C=1	S=10
B=3	D=4	G=0

	Q	Visited
1	(10 S)	S
2	(2 A S) (3 B S)	A,B,S
3	(1 C A S) (3 B S) (4 D A S)	C,D,B,A,S
4	(3 B S) (4 D A S)	C,D,B,A,S
5		

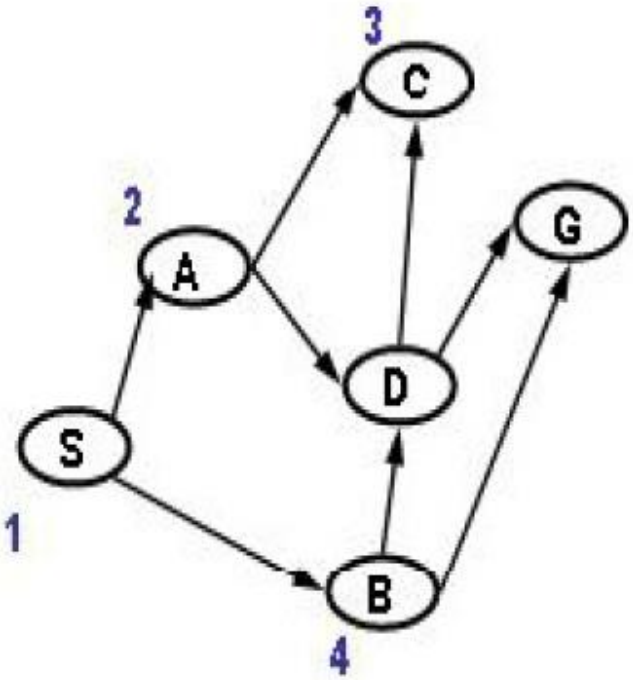


Heuristic Values

A=2 C=1 S=10

B=3 D=4 G=0

	Q	Visited
1	(10 S)	S
2	(2 A S) (3 B S)	A,B,S
3	(1 C A S) (3 B S) (4 D A S)	C,D,B,A,S
4	(3 B S) (4 D A S)	C,D,B,A,S
5	(0 G B S) (4 D A S) (4 D B S)	G,C,D,B,A,S

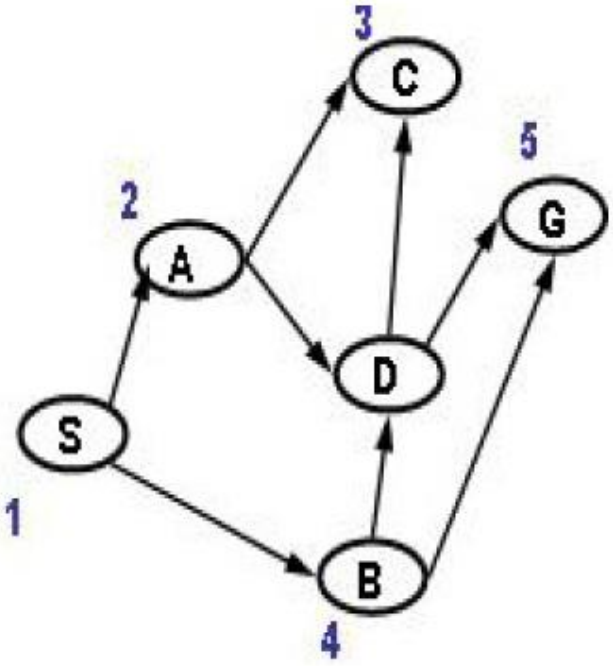


Heuristic Values

A=2 C=1 S=10

B=3 D=4 G=0

	Q	Visited
1	(10 S)	S
2	(2 A S) (3 B S)	A,B,S
3	(1 C A S) (3 B S) (4 D A S)	C,D,B,A,S
4	(3 B S) (4 D A S)	C,D,B,A,S
5	(0 G B S) (4 D A S) (4 D B S)	G,C,D,B,A,S

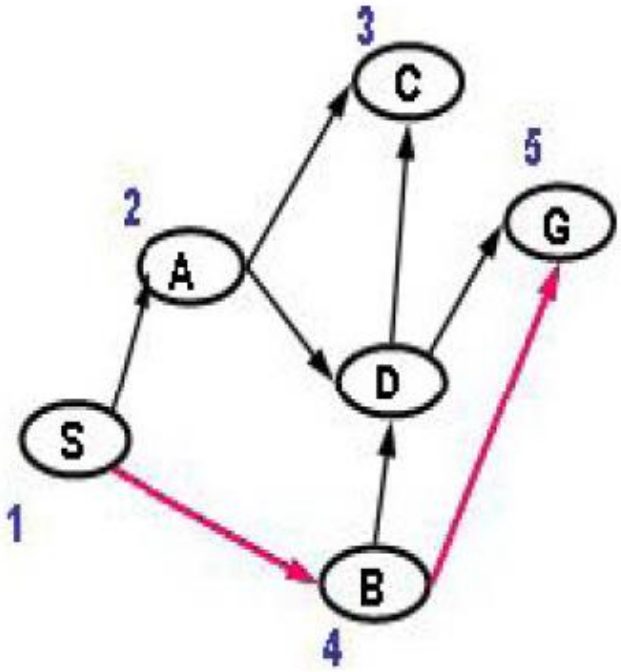


Heuristic Values

A=2 C=1 S=10

B=3 D=4 G=0

	Q	Visited
1	(10 S)	S
2	(2 A S) (3 B S)	A,B,S
3	(1 C A S) (3 B S) (4 D A S)	C,D,B,A,S
4	(3 B S) (4 D A S)	C,D,B,A,S
5	(0 G B S) (4 D A S) (4 D B S)	G,C,D,B,A,S

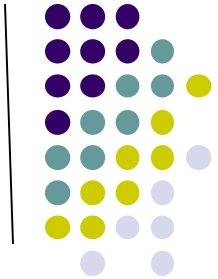


Heuristic Values

A=2 C=1 S=10

B=3 D=4 G=0

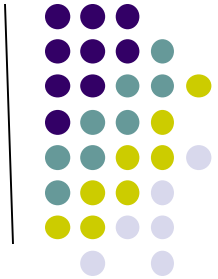
Properties of greedy best-first search



- Complete? No – can get stuck in loops, e.g., Iasi → Neamt → Iasi → Neamt →

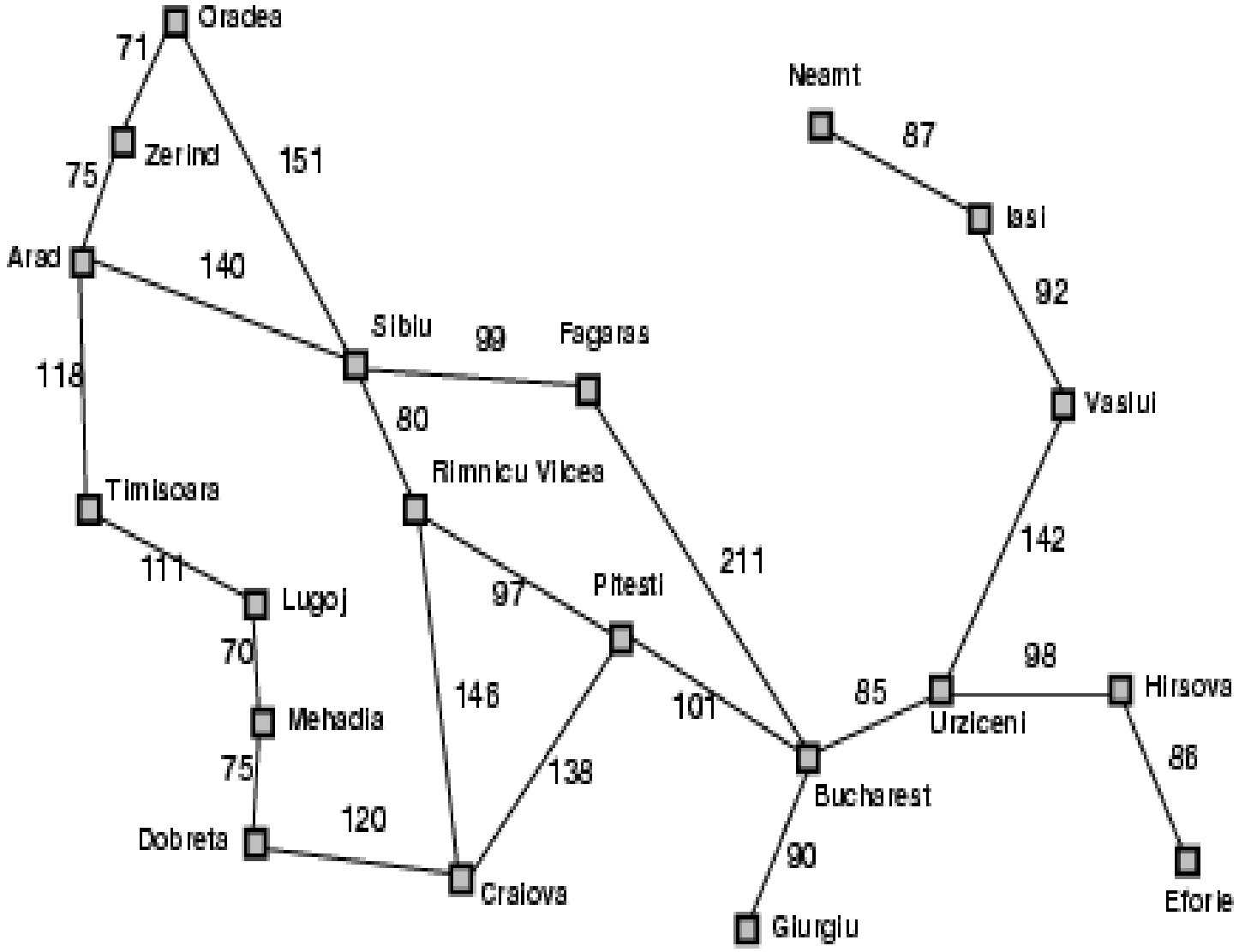
  Complete in finite space with repeated-state checking
- Time? $O(b^m)$, but a good heuristic can give dramatic improvement – *m: max. depth*
- Space? $O(b^m)$ – keeps all nodes in memory
- Optimal? No – it can start down an infinite path and never return to try other possibilities.

A* search



- Idea: minimizing the total estimated solution cost.
- Evaluation function $f(n) = g(n) + h(n)$
- $g(n)$ = the cost to reach the node n
- $h(n)$ = estimated cost from n to goal
- $f(n)$ = estimated total cost of path through n to goal

Romania with step costs in km

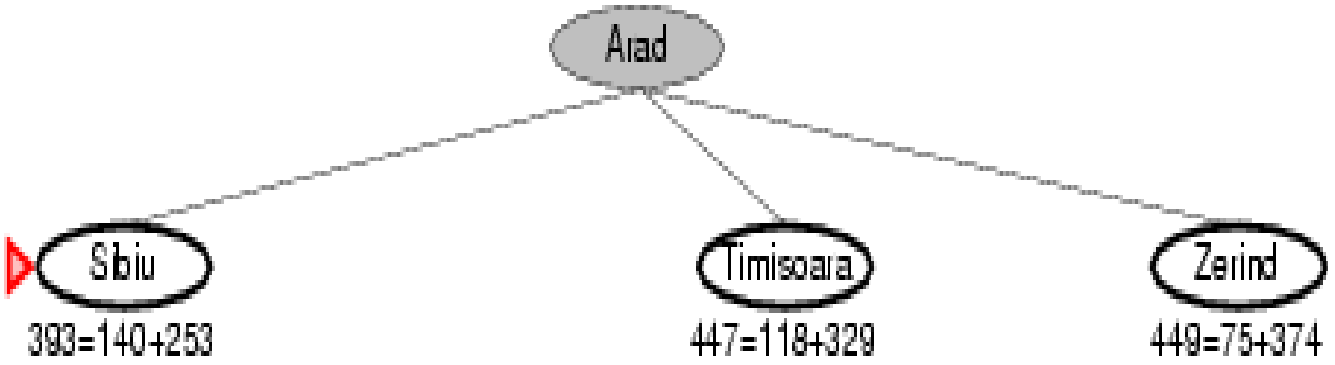


Straight-line distance to Bucharest

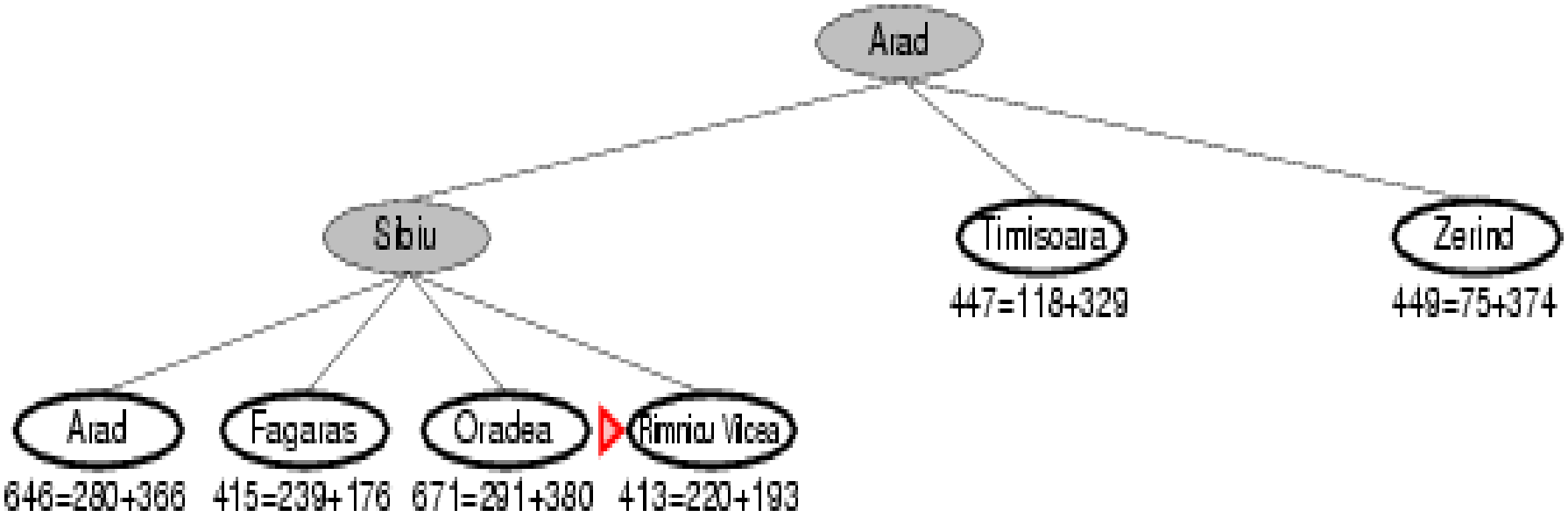
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

A* search example

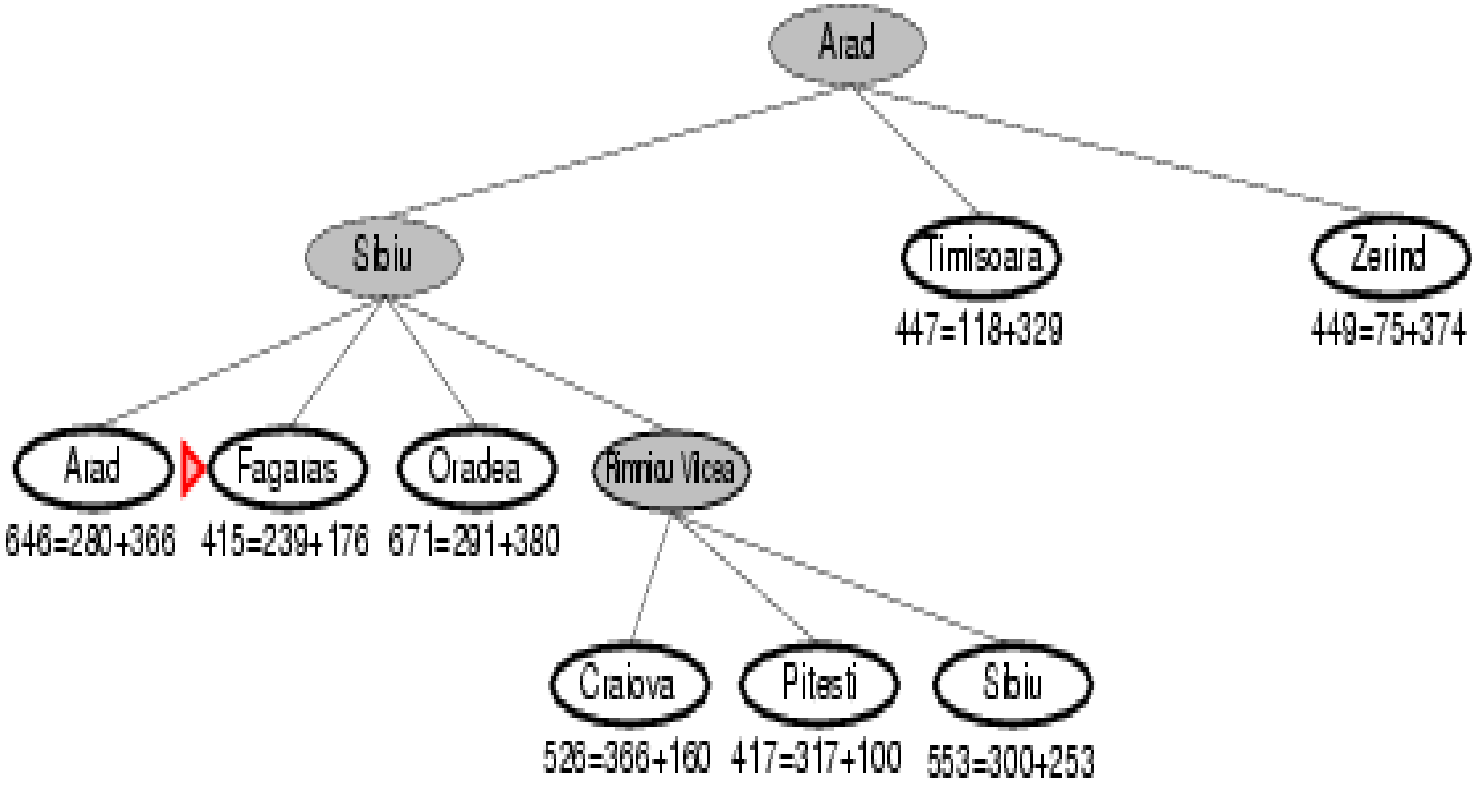
A* search example



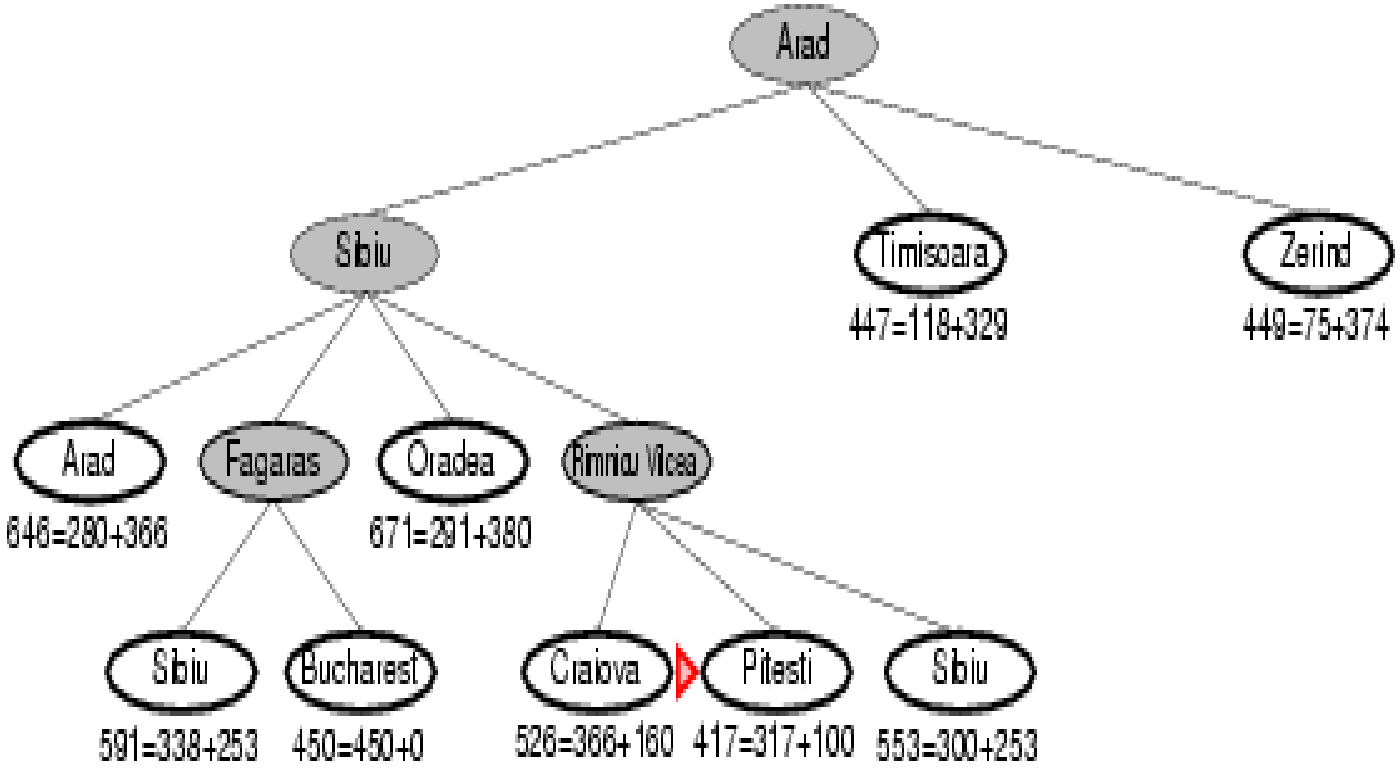
A* search example



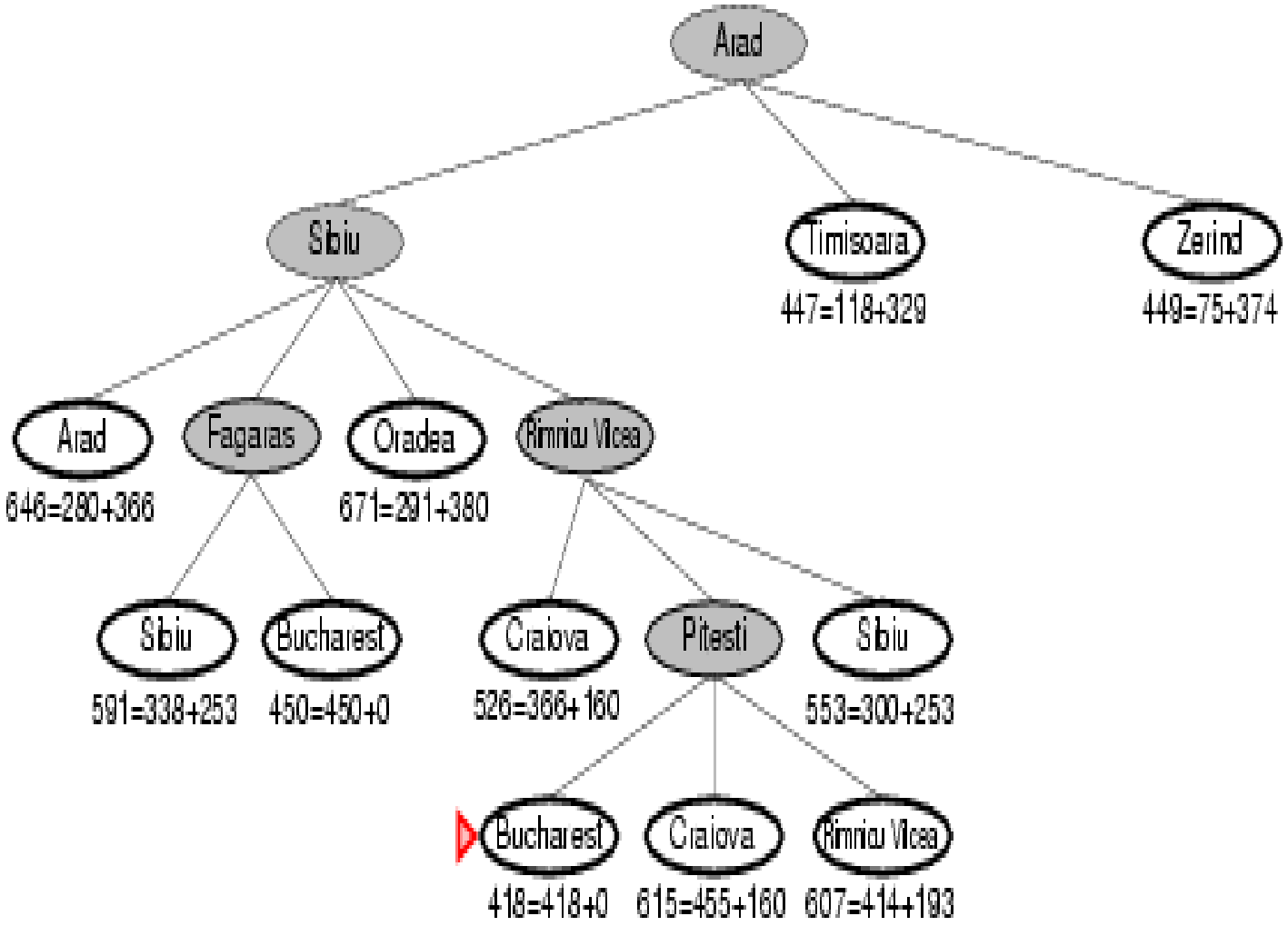
A* search example

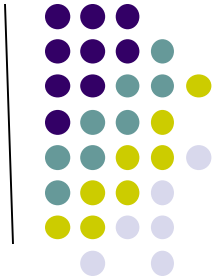


A* search example



A* search example

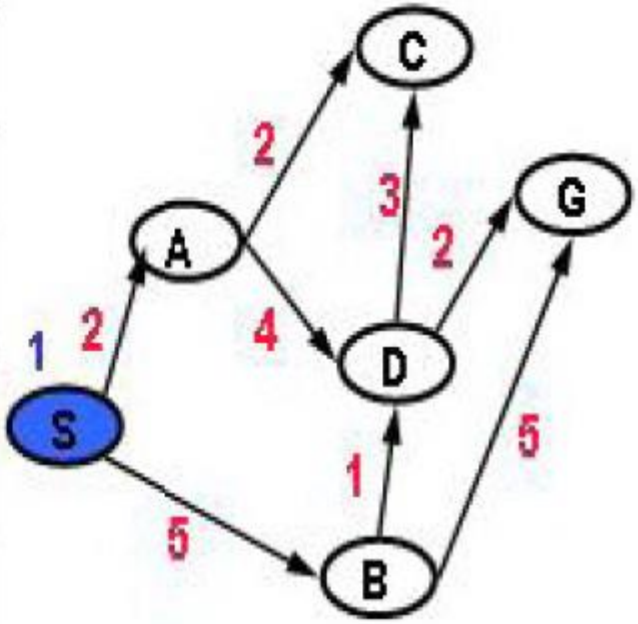




A* SEARCH – ANOTHER EXAMPLE

Pick best (by path length+heuristic) element of Q;
Add path extensions anywhere in Q

	Q
1	(0 S)

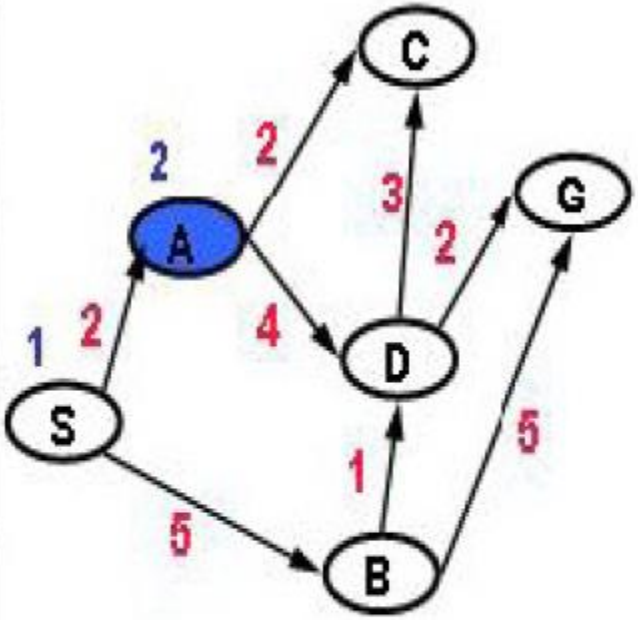


Heuristic Values

A=2	C=1	S=0
B=3	D=1	G=0

Added paths in blue; underlined paths are chosen for extension.
We show the paths in reversed order; the node's state is the first entry.

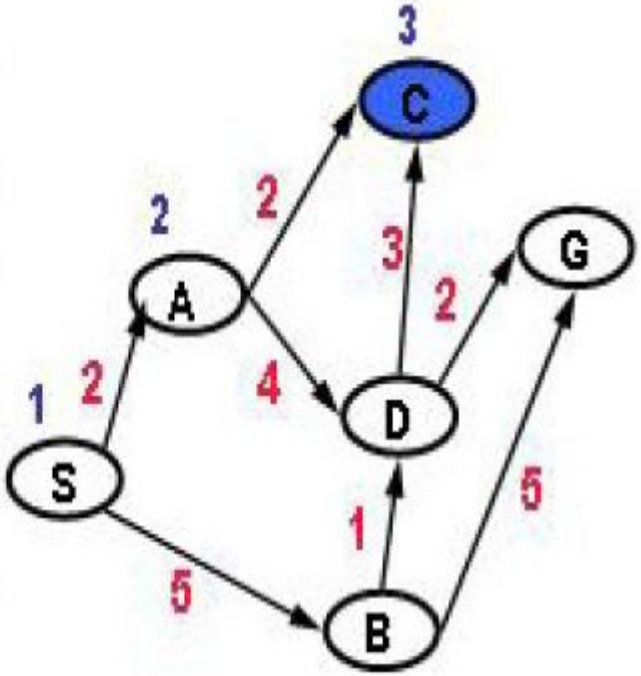
	Q
1	(0 S)
2	(4 A S) (8 B S)



Heuristic Values

A=2	C=1	S=0
B=3	D=1	G=0

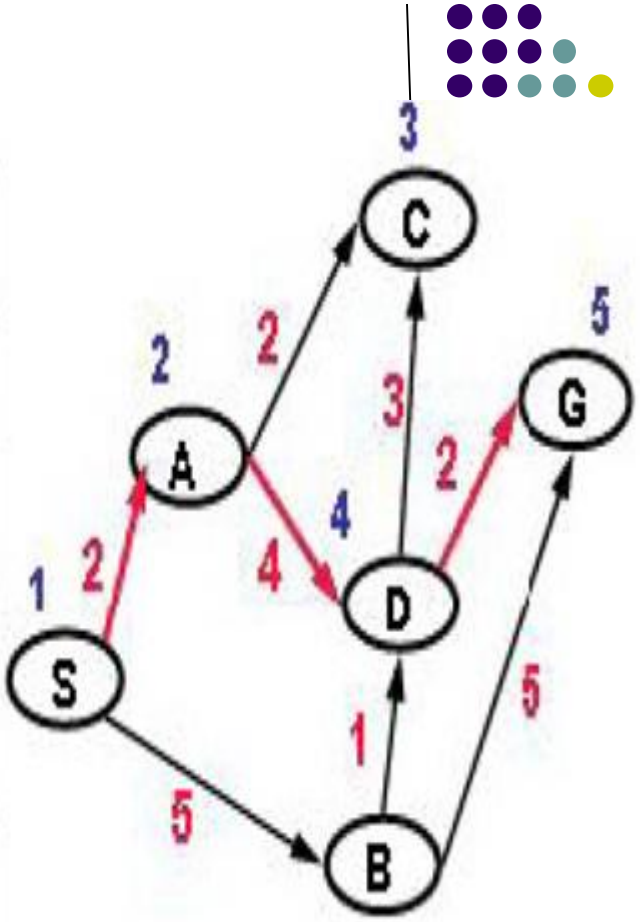
	Q
1	(0 S)
2	(4 A S) (8 B S)
3	(5 C A S) (7 D A S) (8 B S)



Heuristic Values

A=2	C=1	S=0
B=3	D=1	G=0

	Q
1	(0 S)
2	(4 A S) (8 B S)
3	(5 C A S) (7 D A S) (8 B S)
4	(7 D A S) (8 B S)
5	(8 G D A S) (8 B S) (10 C D A S)

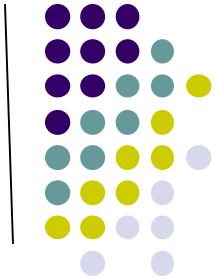


Heuristic Values

A=2	C=1	S=0
B=3	D=1	G=0

Sequence of State Expansions: S – A – C – D - G

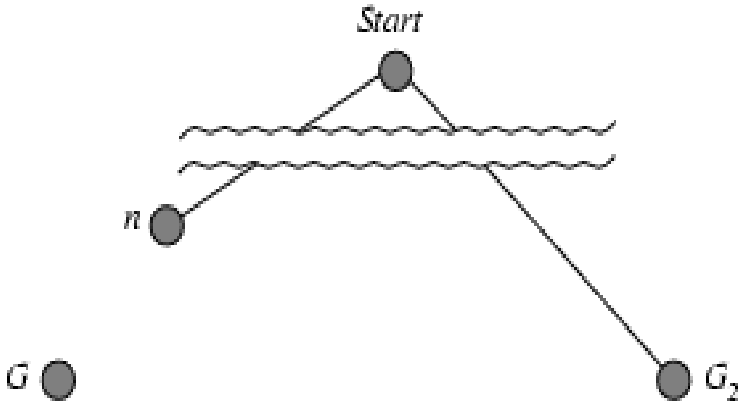
Admissible heuristics



- 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 (thinks that the cost of solving the problem is less than it actually is)
- Example: $h_{SLD}(n)$ (never overestimates the actual road distance – because the shortest path between any two points is a straight line.)
- Theorem: If $h(n)$ is admissible, A* using **`TREE-SEARCH`** is optimal

Optimality of A* (proof)

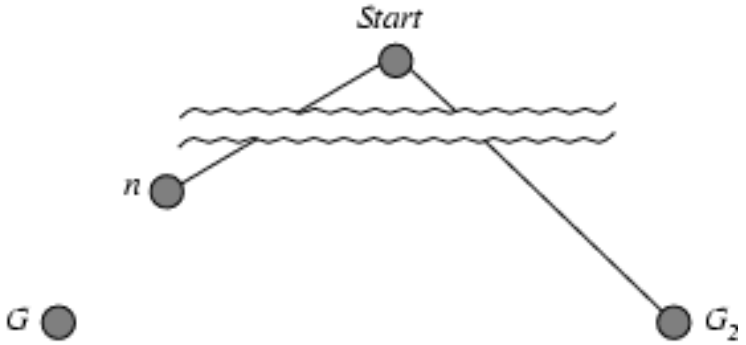
- Suppose some suboptimal goal G_2 has been generated and is in the fringe. Let *n* be an unexpanded node in the fringe such that *n* is on a shortest path to an optimal goal *G*.



- $f(G_2) = g(G_2)$ since $h(G_2) = 0$ (true for any goal node)
- $g(G_2) > g(G)$ since G_2 is suboptimal
- $f(G) = g(G)$ since $h(G) = 0$
- $f(G_2) > f(G)$ from above

Optimality of A* (proof)

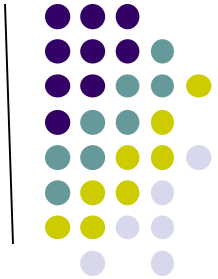
- Suppose some suboptimal goal G_2 has been generated and is in the fringe. Let *n* be an unexpanded node in the fringe such that *n* is on a shortest path to an optimal goal *G*.



- **$f(G_2)$ > $f(G)$** from above
- $h(n) \leq h^*(n)$ since h is admissible
- $g(n) + h(n) \leq g(n) + h^*(n)$
- **$f(n) \leq f(G)$**

Hence $f(G_2) > f(n)$, and A^* will never select G_2 for expansion

Consistency

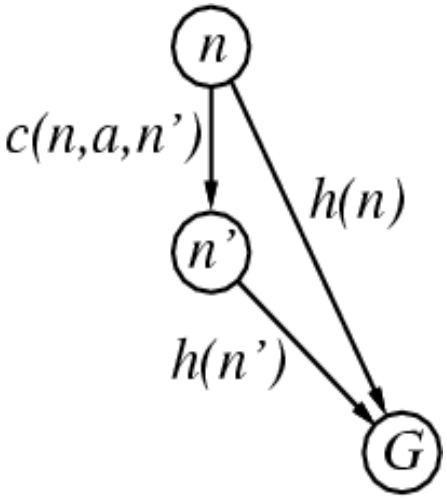


- If we use the GRAPH-SEARCH algorithm instead of TREE-SEARCH, then this proof breaks down.
- GRAPH-SEARCH can discard the optimal path if it is not the first one generated.
- So, suboptimal solutions can be returned.
- To fix this problem, the solution is to ensure that the optimal path to any repeated state is always the first one followed – as in the case with uniform-cost search.
  - we impose an extra requirement on $h(n)$ – consistency (monotonicity)

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')* (triangle inequality)

(each side of a triangle cannot be longer than the sum of the other two sides.)

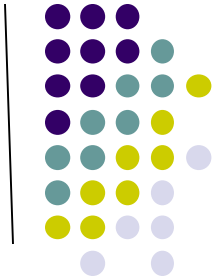


- If *h* is consistent, we have

f(n') = g(n') + h(n')
= g(n) + c(n,a,n') + h(n')
≥ g(n) + h(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

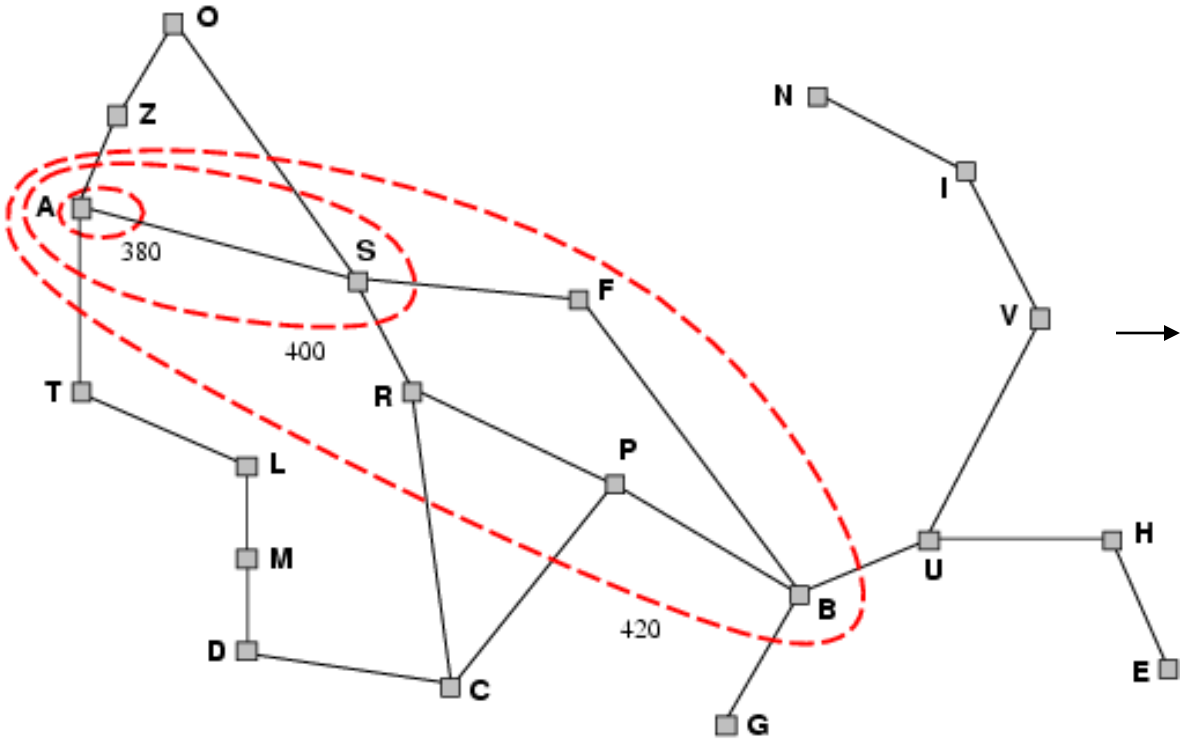
Consistent heuristics



- Consistency is a stricter requirement than addmissibility.
  - all the admissible heuristics we discussed so far are also consistent.
  - e.g. h_{SLD} - general triangle inequality is satisfied when each side is measured by SLD.
- Because the sequence of nodes expanded by A* using GRAPH-SEARCH is in nondecreasing order of $f(n)$, the first goal node selected for expansion must be an optimal solution.

Optimality of A*

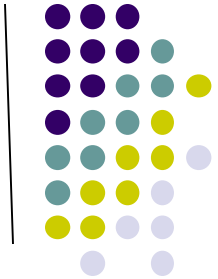
- A* expands nodes in order of increasing *f*-values.
- Gradually adds "*f*-contours" of nodes in the state space.
- Contour *i* has all nodes with $f=f_i$, where $f_i < f_{i+1}$



Map of Romania showing contours at *f=380, f=400, f=420*

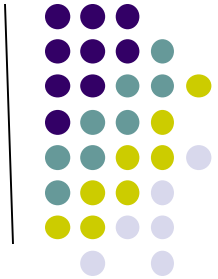
Nodes inside a given contour have f-costs less than or equal to the contour value.

Properties of A*



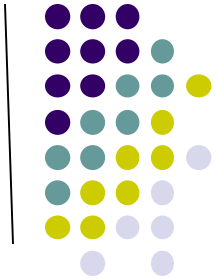
- Complete? Yes
- Time? Exponential
- Space? Keeps all nodes in memory
- Optimal? Yes

Uniform Cost (UC) versus A*



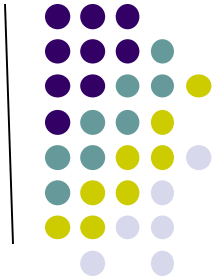
- **UC** is trying to identify the shortest path to every state in the graph in order.
- It has no particular bias for finding a path to a goal early in the search.
- We can introduce such a bias by means of heuristic function **h(N)**, which is an estimate **(h)** of the distance from a state to the goal.

Uniform Cost (UC) versus A*



- Instead of enumerating paths in order of just length (**g**), enumerate paths in terms of
  f = estimated total path length = g + h.
- An estimate that always underestimates the real path length to the goal is called **admissible**.
- For example, an estimate of 0 is admissible (but useless).

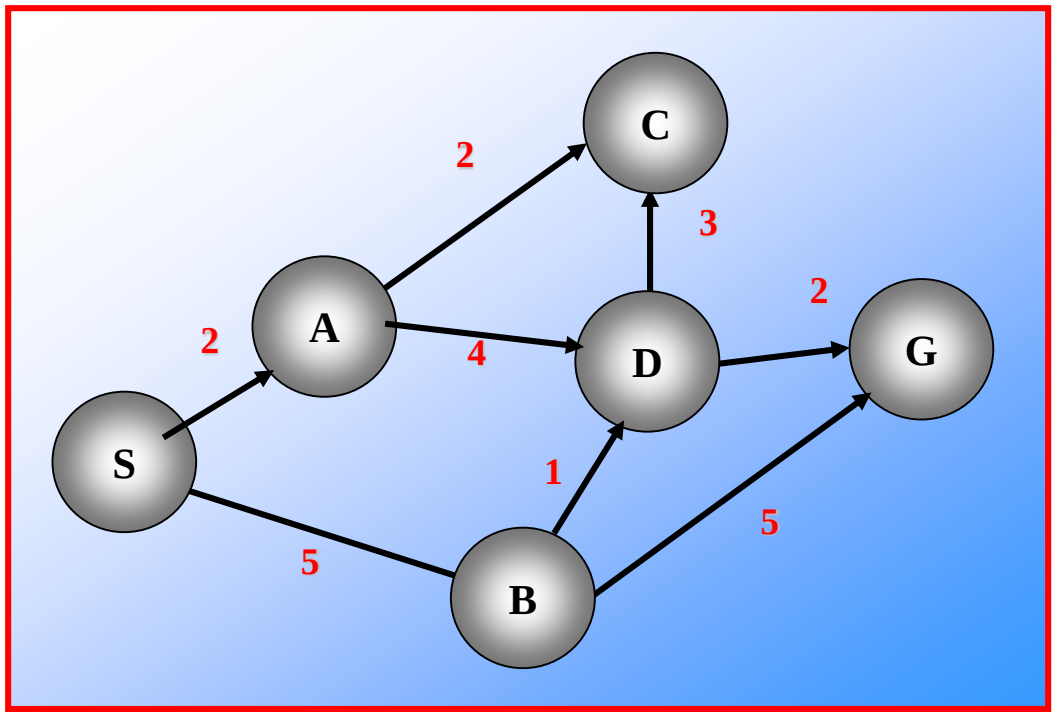
Uniform Cost (UC) versus A*



- Straight line distance is admissible estimate for path length in Euclidean space.
- Use of an admissible estimate guarantees that UC will still find the shortest path.
- **UC with an admissible estimate** is known as **A* Search**.

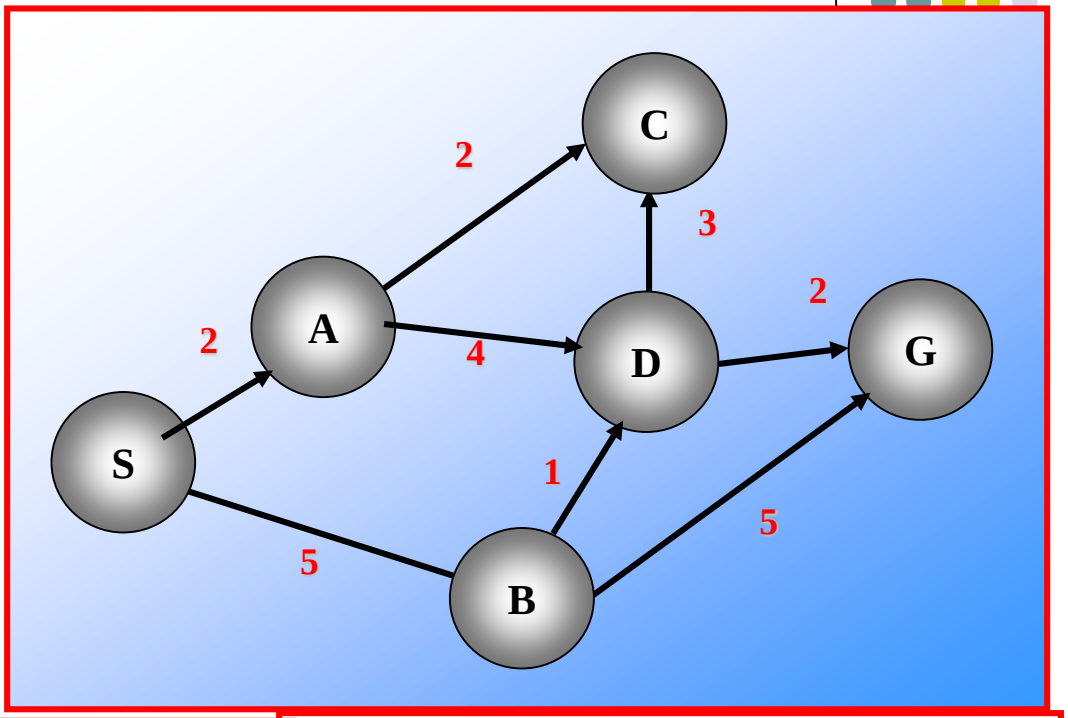
Not all heuristics are admissible!!

Given the link lengths in the figure, is the table of heuristic values that we used in our earlier Best-First example an **admissible heuristic**?



Heuristic values:
A=2, B=3, C=1, D=4, S=10, G=0

Not all heuristics are admissible!!



Heuristic values:
A=2, B=3, C=1, D=4, S=10, G=0

No!

- A is ok
- B is ok
- C is ok
- **D** is **too big**, needs to be <= 2
- S is too big, can always use 0 for start

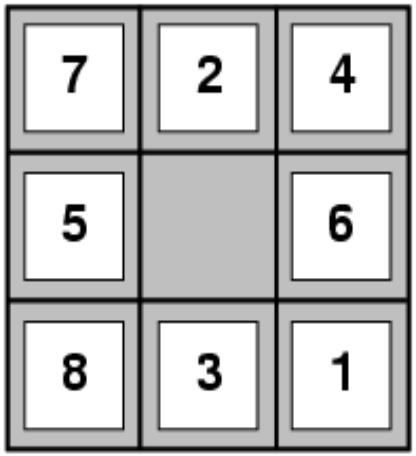
Heuristic Functions

E.g., for the 8-puzzle:

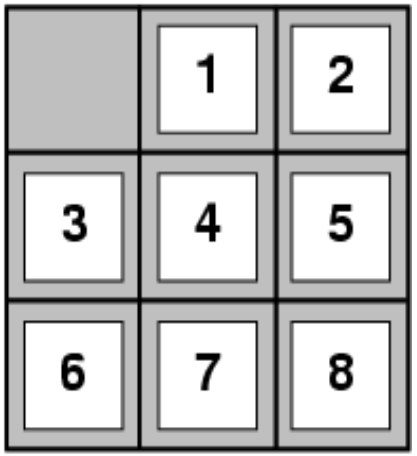
- $h_1(n)$ = number of misplaced tiles
- $h_2(n)$ = total Manhattan distance

(i.e., no. of squares from desired location of each tile)

- $h_1(S) = ?$
- $h_2(S) = ?$



Start State



Goal State

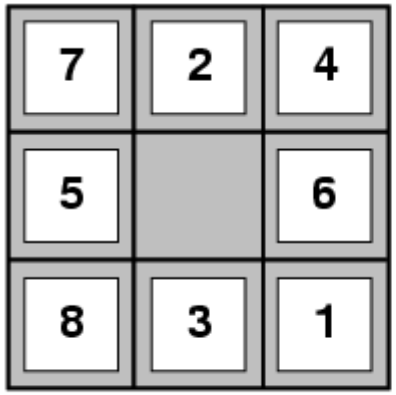
The solution is 26 steps long.

Admissible heuristics

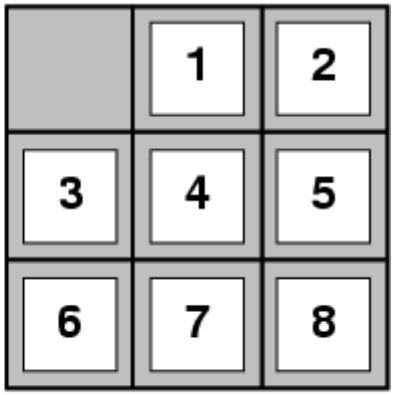
E.g., for the 8-puzzle:

- $h_1(n)$ = number of misplaced tiles
- $h_2(n)$ = total Manhattan distance

(i.e., no. of squares from desired location of each tile)



Start State



Goal State

- $h_1(S) = ?$ 8
- $h_2(S) = ?$ 3+1+2+2+3+2+2+3 = 18
- Neither of these overestimates the true solution cost.

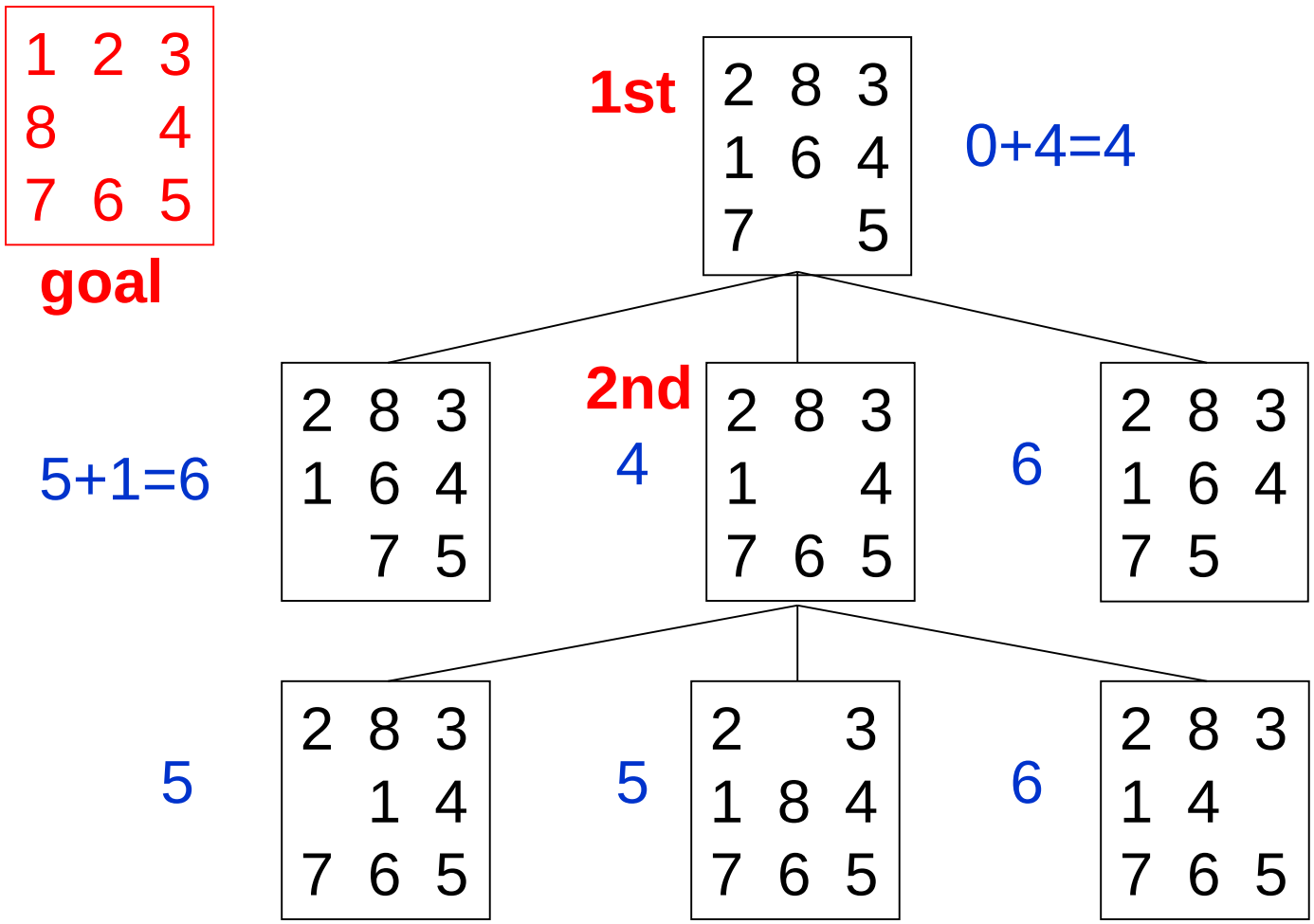
Dominance

- If $h_2(n) \geq h_1(n)$ for all n (both admissible)
- then h_2 dominates h_1
- h_2 is better for search

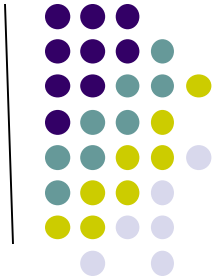
- Typical search costs (average number of nodes expanded):

- *d=12* IDS = 3,644,035 nodes
  $A^*(h_1)$ = 227 nodes
  $A^*(h_2)$ = 73 nodes
- *d=24* IDS = too many nodes
  $A^*(h_1)$ = 39,135 nodes
  $A^*(h_2)$ = 1,641 nodes

8 Puzzle Using Number of Misplaced Tiles with A* Search

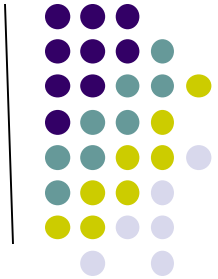


Relaxed problems



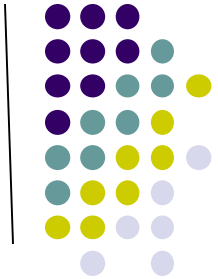
- A problem with fewer restrictions on the actions is called a relaxed problem
- The cost of an optimal solution to a relaxed problem is an admissible heuristic for the original problem
- If the rules of the 8-puzzle are relaxed so that a tile can move anywhere, then $h_1(n)$ gives the shortest solution
- If the rules are relaxed so that a tile can move to any adjacent square, then $h_2(n)$ gives the shortest solution
- **Key Point** : the optimal solution cost of a relaxed problem is no greater than the optimal solution cost of the real problem.

Inventing admissible heuristic functions



- If a problem definition is written down in a formal language, it is possible to construct relaxed problems automatically (ABSOLVER)
- If 8-puzzle is described as
  - A tile can move from square A to square B if
  - A is horizontally or vertically adjacent to B and B is blank

Inventing admissible heuristic functions



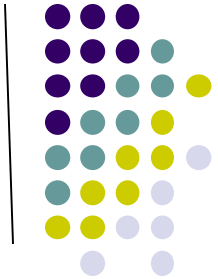
- A relaxed problem can be generated by removing one or both of the conditions

  (a) A tile can move from square A to square B if A is adjacent to B

  (b) A tile can move from square A to square B if B is blank

  (c) A tile can move from square A to square B
- h2 can be derived from (a) – h2 is the proper score if we move each tile into its destination
- h1 can be derived from (c) – it is the proper score if tiles could move to their intended destination in one step

SUMMARY



- Heuristic functions estimate costs of shortest paths.
- Good heuristics can dramatically reduce search cost.
- Greedy best-first search expands lowest h.
  - incomplete and not always optimal.
- A* search expands lowest g + h.
  - complete and optimal.
  - also optimally efficient.
- Admissible heuristics can be derived from exact solution of relaxed problems.