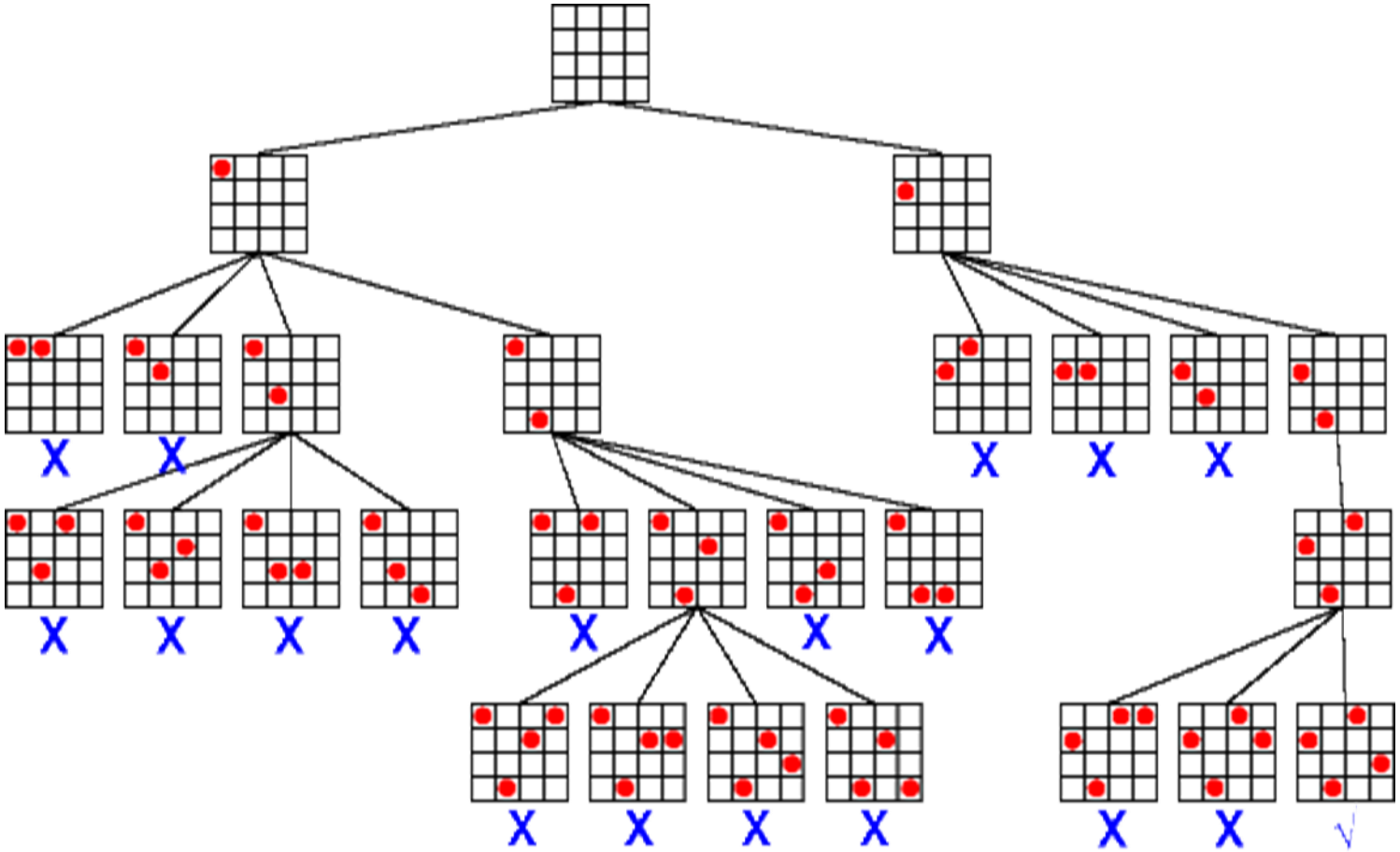
Informed (Heuristic) Search Strategies

Review of Blind Search, Duplicates

Two types of blind searches

- Tree-Search
  - The state space is a tree
    - But, if you are lucky, duplicates may not exist for your particular problem. Eg: 8-Queens, where each new queen is placed in the leftmost empty column.
- Graph-Search
  - Duplicates are essentially pruned off
    - If done with closed eyes can result in loosing the optimal.
  - Can reduce the actual space requirement considerably.

4-Queens Problem; State space is a tree. Because new queen is placed in leftmost nonempty column.



Duplicates

- For some problems, repeated states are unavoidable. This includes all problems where the actions are reversible, such as route-finding problems and sliding-blocks puzzles.
- *Algorithms that forget their history are doomed to repeat it.*
  - *Can result in getting caught in an infinite loop, or exploring the same state many times.*

Duplicates

- Simple strategy can work with Uniform Cost and BFS.
- Uniform Cost Search and BFS are admissible (Finds optimal solutions).
- New node, if is already existing either in CLOSED or in OPEN, then retain the good one and remove the bad one (Strategy tells us which is the “bad” candidate).

Duplicates

- For DFS, one can do a similar thing as done with BFS, in order to avoid infinite loops.
- But, since DFS is not an admissible algorithm, anyhow optimal path is not guaranteed.
- DFID is admissible and is similar to BFS, hence same strategy can be used.
  - Duplicates can be allowed also {No fear of infinite loop. We are limited to a finite graph each time}.

What we do in this Course?

- We **do not like duplicates**, hence if there is a way to remove duplicates (without compromising with the solution), we do it.

- So, in exams, be careful to see that duplicates are eliminated appropriately.
- Even with duplicates you may get answer, but this we treat as an inferior one and partially some marks are reduced.

Informed (Heuristic) Search Strategies

- ***Informed Search*** – a strategy that uses problem-specific knowledge beyond the definition of the problem itself.
- ***Best-First Search*** – an algorithm in which a node is selected for expansion based on an evaluation function *f(n)*
  - Traditionally the node with the lowest evaluation function is selected
  - Not an accurate name…expanding the best node first would be a straight march to the goal.
  - Choose the node that *appears* to be the best

Informed (Heuristic) Search Strategies

- There is a whole family of Best-First Search algorithms with different evaluation functions
  - Each has a heuristic function *h(n)*

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

- Example: in route planning the estimate of the cost of the cheapest path might be the straight line distance between two cities

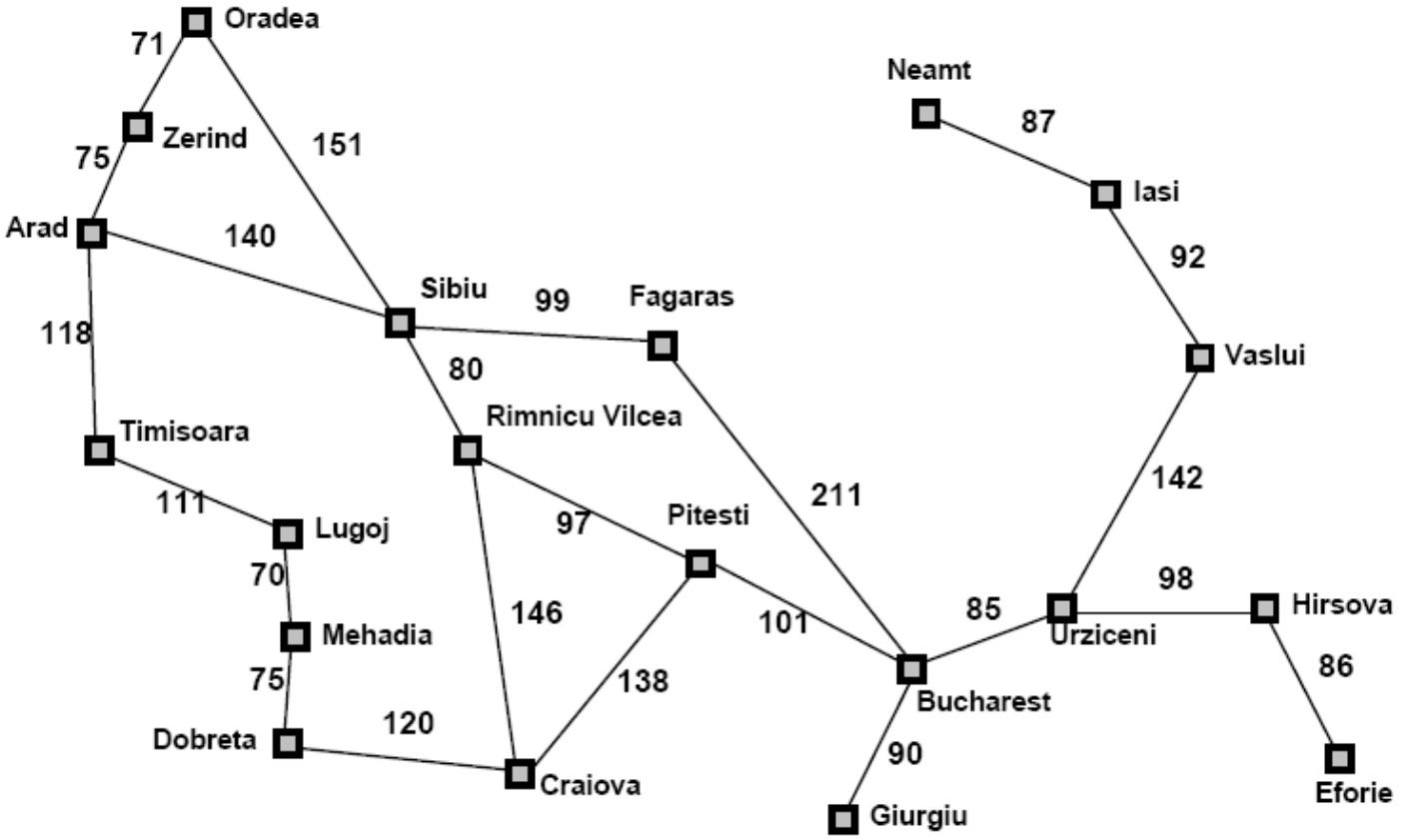
Terminology

- *g(n)* = cost from the initial state to the current state n
- *h(n)* = estimated cost of the cheapest path from node n to a goal node
- *f(n)* = evaluation function to select a node for expansion (usually the lowest cost node)

Greedy Best-First Search

- Some authors call this as *the best first search* method
- Greedy Best-First search tries to expand the node that is closest to the goal assuming it will lead to a solution quickly
  - *f(n) = h(n)*
  - It is really a “Greedy Search”
- Implementation
  - expand the “most desirable” node into the fringe queue
  - sort the queue in decreasing order of desirability
- Example: consider the straight-line distance heuristic h_{SLD}
  - Expand the node that appears to be closest to the goal

Greedy Best-First Search

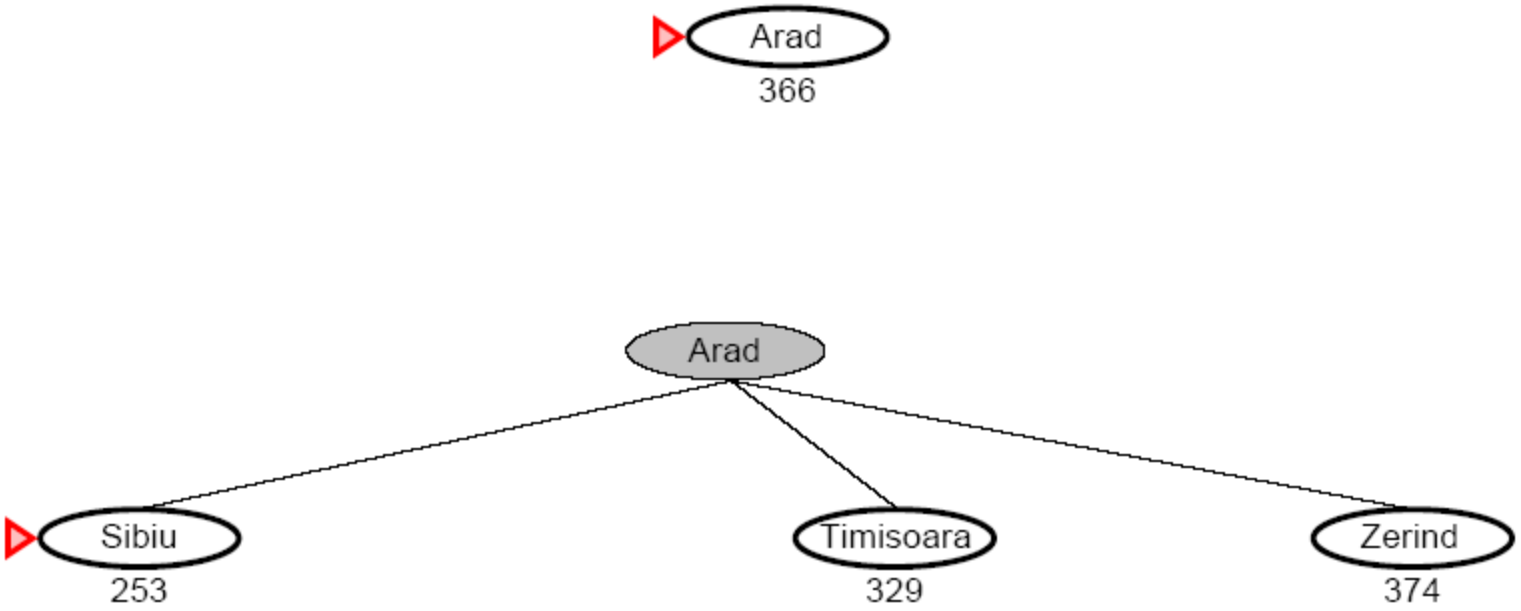


Straight−line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

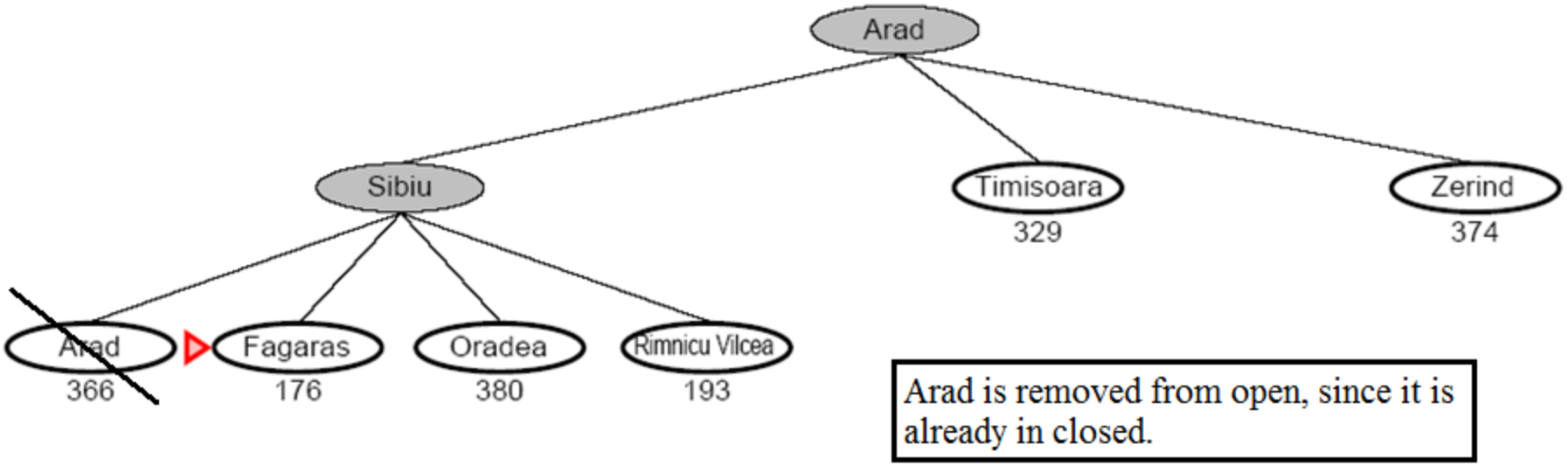
Greedy Best-First Search

- h_{SLD}(Node for Arid) = 366
- Notice that the values of h_{SLD} cannot be computed from the problem itself
- It takes some experience to know that h_{SLD} is correlated with actual road distances
  - Therefore a useful heuristic

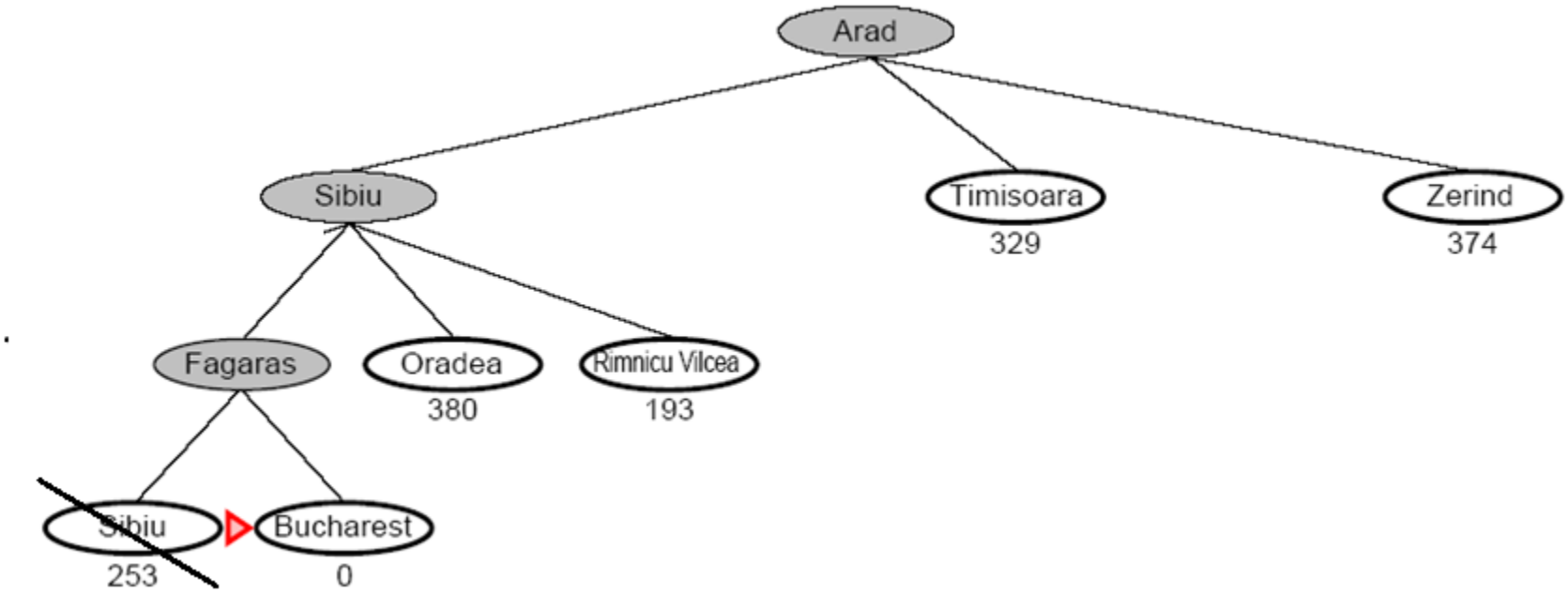
Greedy Best-First Search

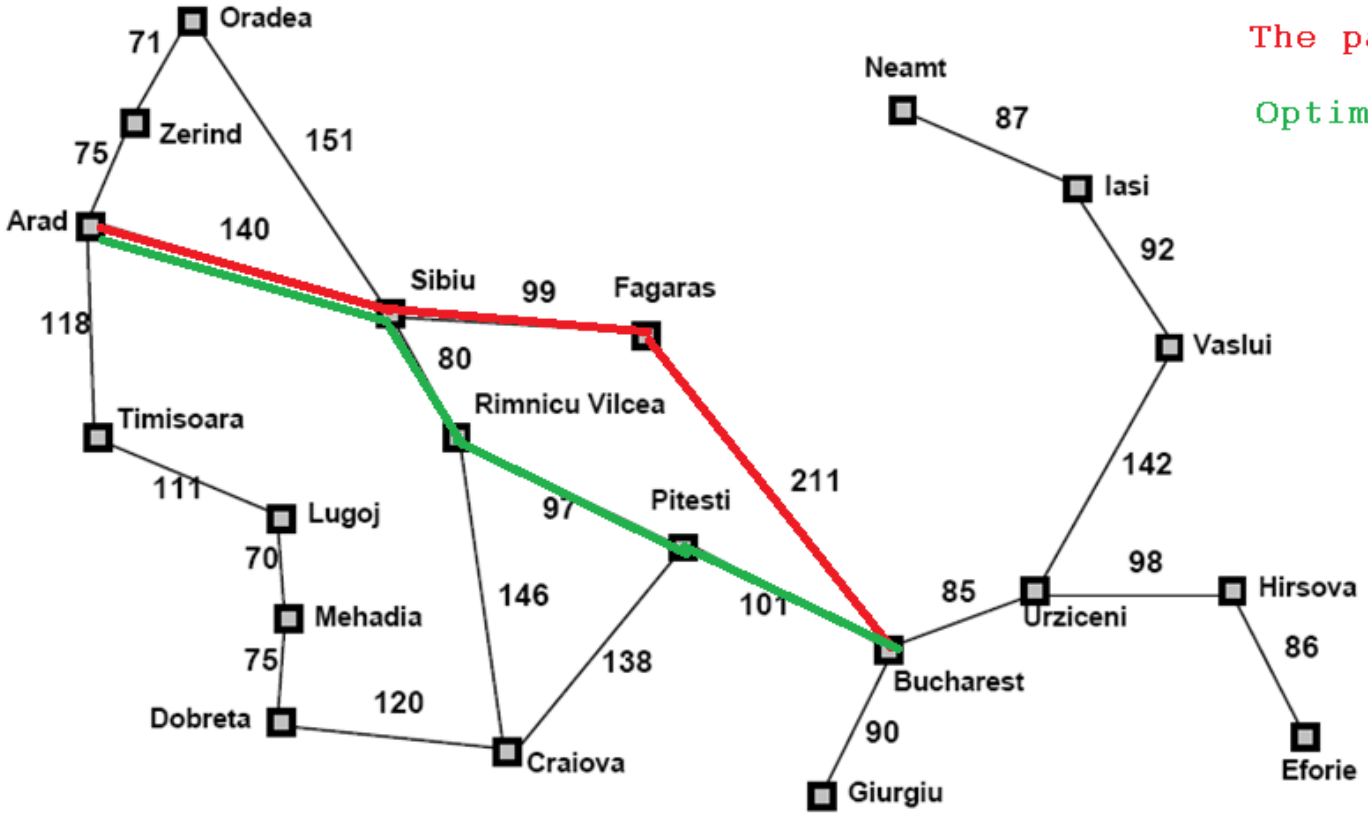


Greedy Best-First Search



Greedy Best-First Search





The path found has cost 450

Optimal path has cost 418

For 8-Puzzle

- Heuristic could be
  - number of misplaced tiles.
  - Sum of “number of moves each tile is away from its goal position”

Greedy Best-First Search

- Greedy best-first search resembles depth-first search in the way it prefers to follow a single path all the way to the goal, but will backup when it hits a dead end.
- Complete
  - Yes for finite graphs (with duplicate elimination). No, otherwise.
- Time
  - $O(b^m)$ but a good heuristic can have dramatic improvement
- Space
  - $O(b^m)$ – keeps all the nodes in memory
- Optimal
  - No!

How good is the heuristic?

- Penetrance (Nilson 1998) : nodes_in_the_solution/ total_nodes_expanded.
- Effective branching factor: Inverse of penetrance.

A Quick Review - Again

- *g(n)* = cost from the initial state to the current state n
- *h(n)* = estimated cost of the cheapest path from node n to a goal node
- *f(n)* = evaluation function to select a node for expansion (usually the lowest cost node)

A* Search

- A* (A star) is the most widely known form of Best-First search
  - It evaluates nodes by combining *g(n)* and *h(n)*
  - *f(n) = g(n) + h(n)*
  - Where
    - *g(n)* = cost so far to reach *n*
    - *h(n)* = estimated cost to goal from *n*
    - *f(n)* = estimated total cost of path through *n*

A* Search

- This combines good features of Uniform-cost search and Best first search.
- Does not explore the regions of the search space which will not have optimal solution.

A* Search

- When *h(n)* = actual cost to goal
  - Only nodes in the correct path are expanded
  - Optimal solution is found
- When *h(n)* < actual cost to goal
  - Additional nodes are expanded
  - Optimal solution is found
- When *h(n)* > actual cost to goal
  - Optimal solution can be overlooked

A* Search

- A* is optimal if it uses an ***admissible heuristic***
  - $h(n) \leq h^*(n)$ the true cost from node n
  - if $h(n)$ *never overestimates* the cost to reach the goal

- Example
  - h_{SLD} never overestimates the actual road distance

- A* Algorithm

---

1. Intialize : OPEN = *{S}*; CLOSED = *{ };*
   g(S) = 0; f(S) = h(S);

2. Fail: If OPEN is empty return Fail;

3. Select: Min. cost (*f( )* value) node from OPEN is selected. Let this node be *n*

4. Goal Test: If (Selected node *n* = *G*) return Success & path with its *f* value.

- A* Algorithm (contd …)

---

5. Expand: For each child *m* of *n* Do

   5a. If *m* is entirely new, add to OPEN.

   5b. If *m* has a duplicate in OPEN, retain the cheaper one in OPEN and through off the other.

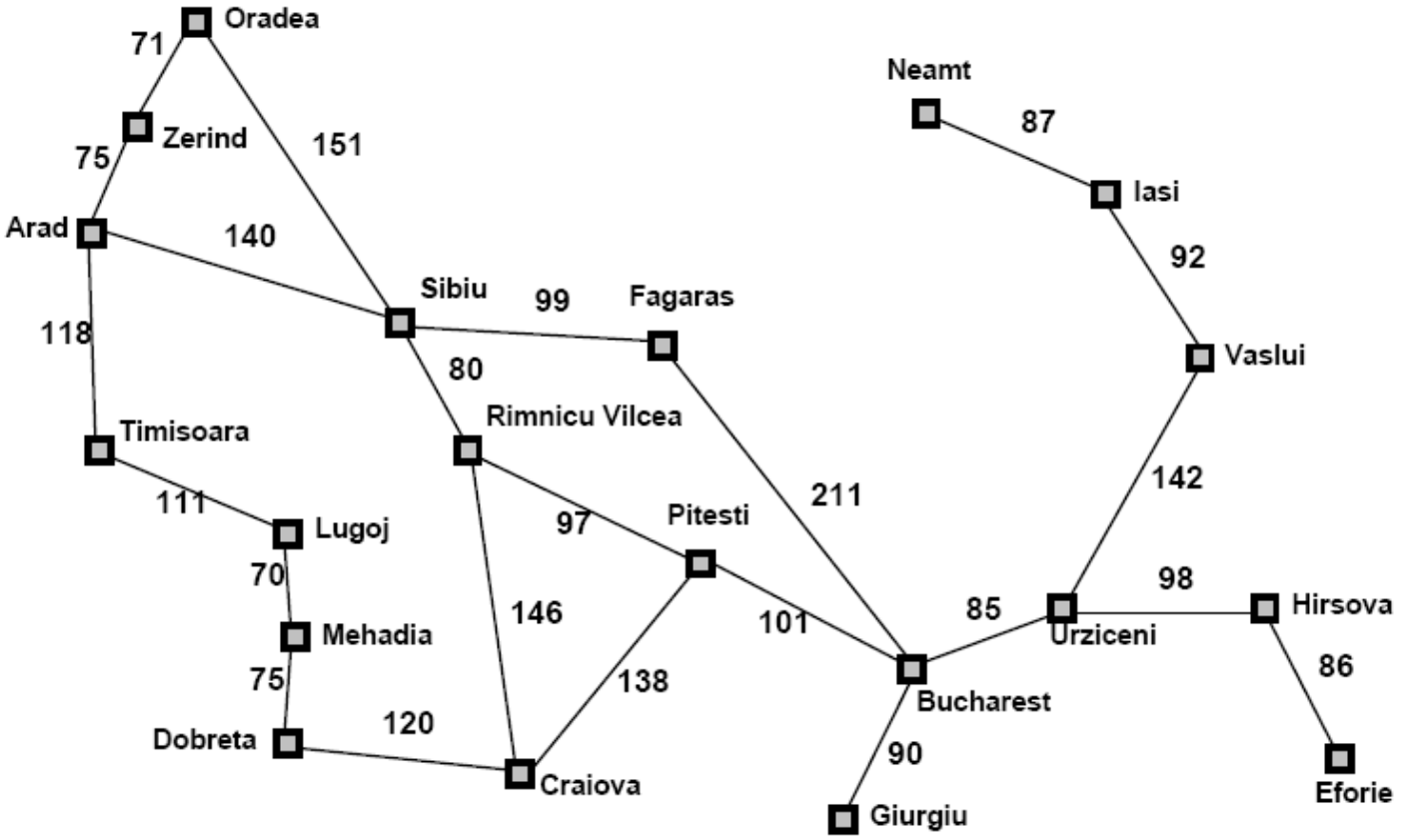
   5c. If *m* has a duplicate in CLOSED,

   (i) If new *m* has more cost then through off new *m*

   (ii) Else through off that *m* which is in CLOSED and keep the new *m* with its cost in OPEN

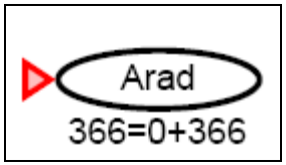
6. Goto STEP 2.

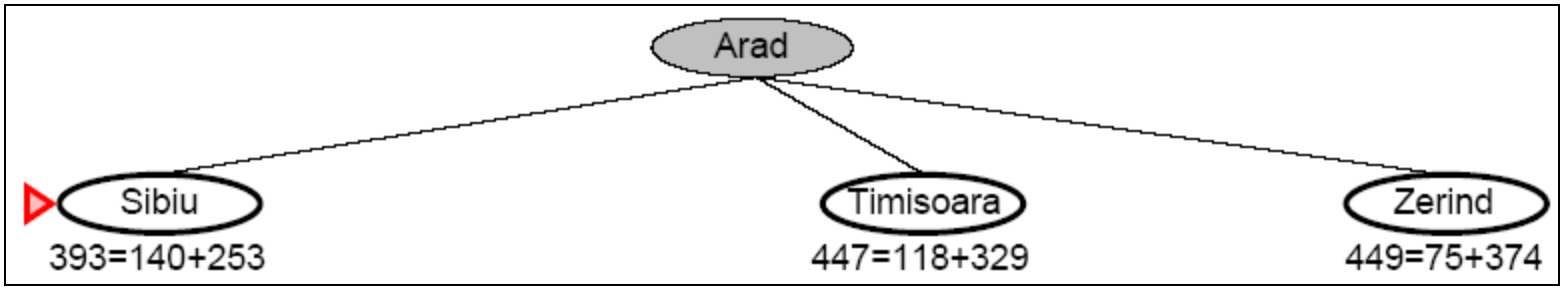
A* Algorithm -- example



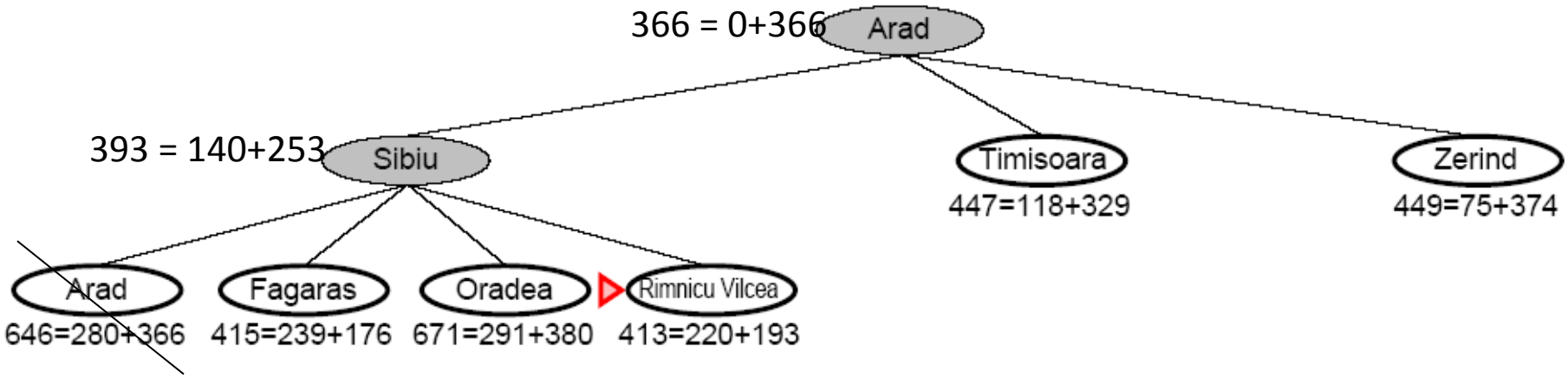
Straight−line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

A* Search

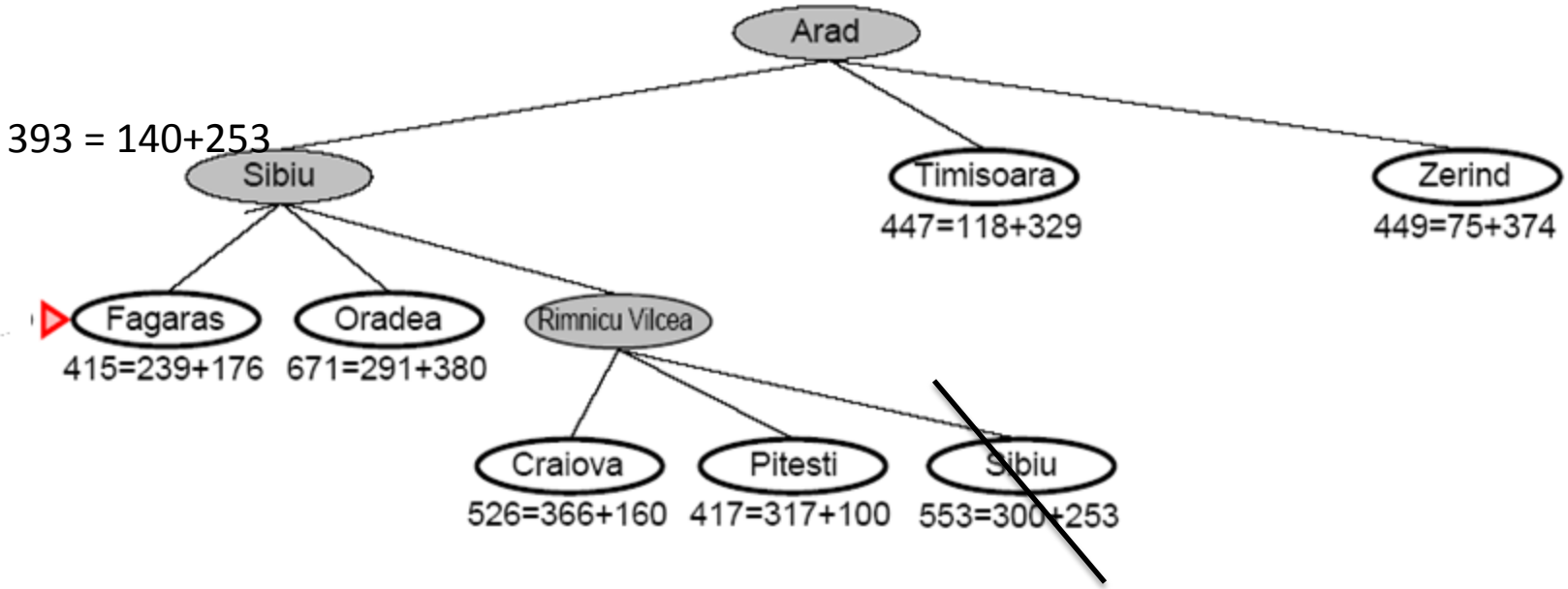

Arad
366=0+366


Arad
Sibiu
393=140+253
Timisoara
447=118+329
Zerind
449=75+374

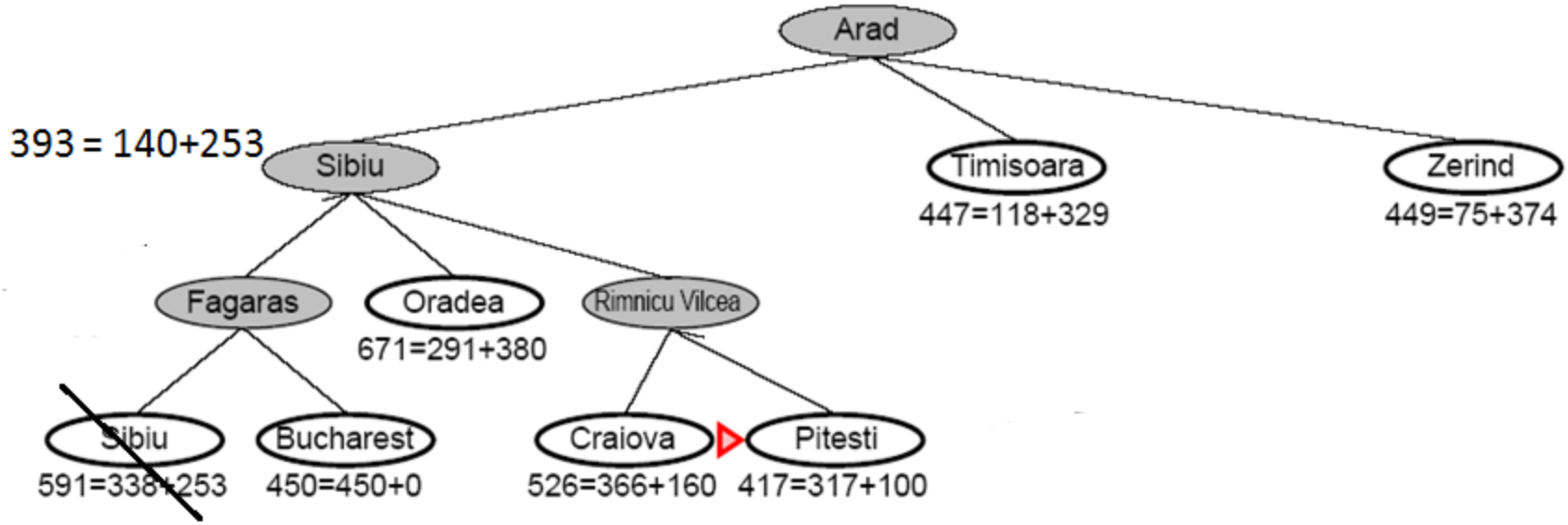
A* example contd…



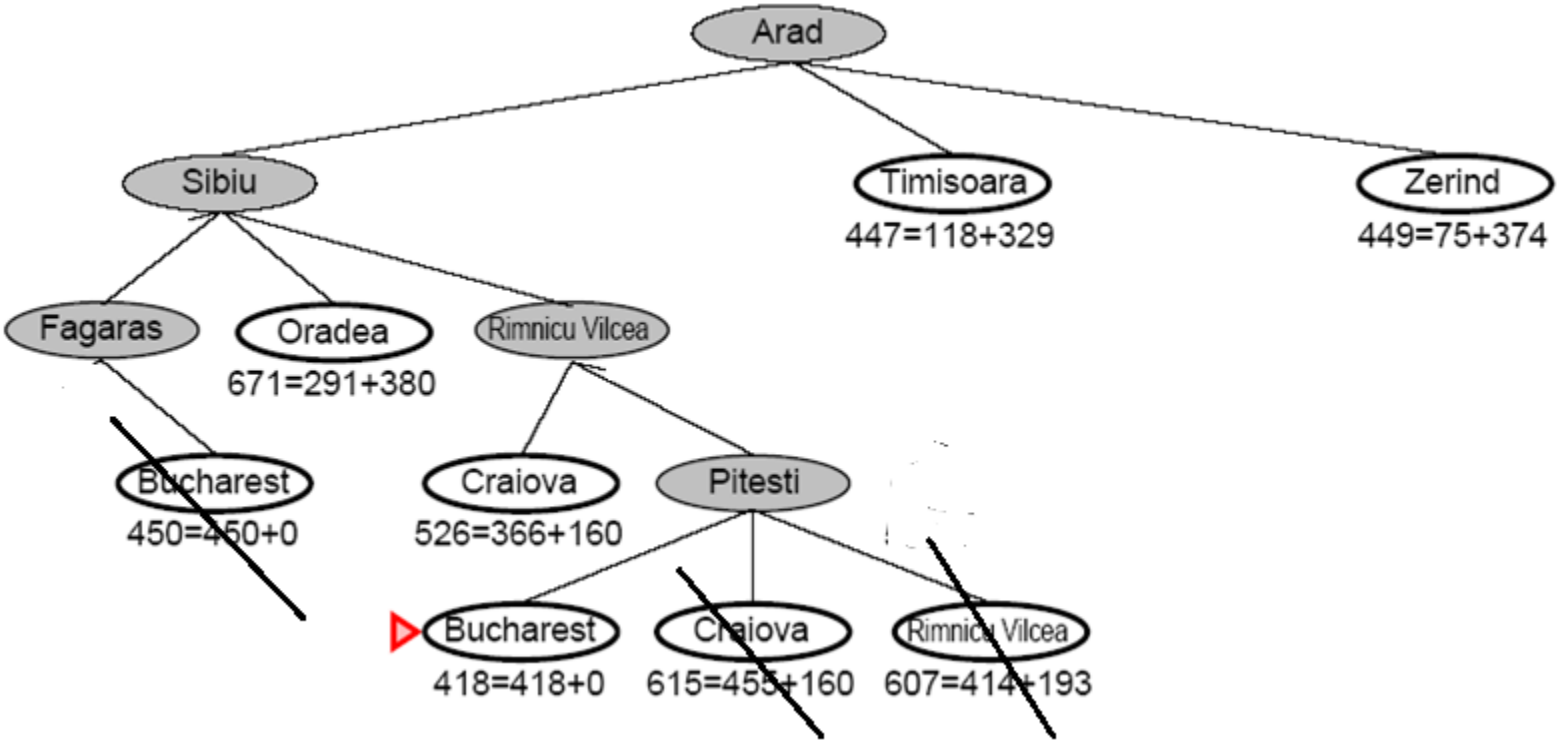
A* Search


393 = 140+253
Arad
Sibiu
Timisoara
447=118+329
Zerind
449=75+374
Fagaras
415=239+176
Oradea
671=291+380
Rimnicu Vilcea
Craiova
526=366+160
Pitesti
417=317+100
Sibiu
553=300+253

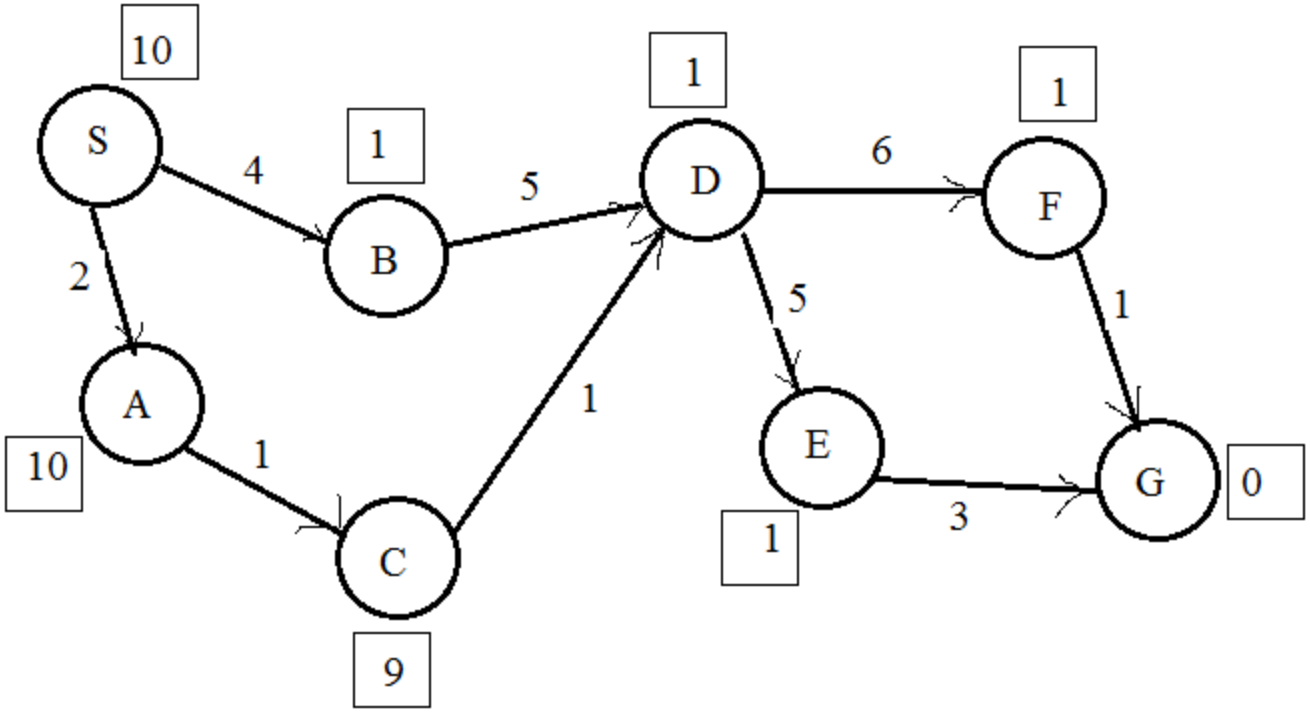
A* Search

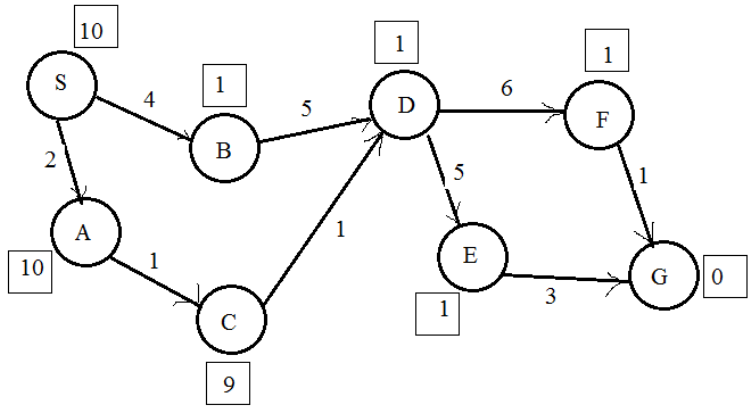

Arad
393 = 140+253
Sibiu
Timisoara
447=118+329
Zerind
449=75+374
Fagaras
Oradea
671=291+380
Rimnicu Vilcea
Sibiu
591=338+253
Bucharest
450=450+0
Craiova
526=366+160
Pitesti
417=317+100

A* Search

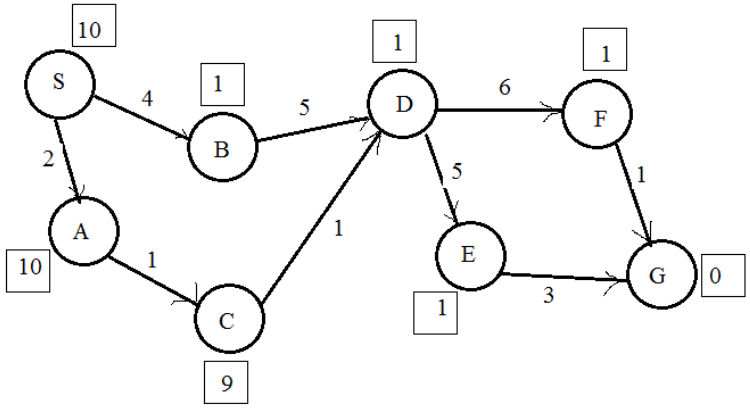

Arad
Sibiu
Timisoara
447=118+329
Zerind
449=75+374
Fagaras
Oradea
671=291+380
Rimnicu Vilcea
Bucharest
450=450+0
Craiova
526=366+160
Pitesti
Bucharest
418=418+0
Craiova
615=455+160
Rimnicu Vilcea
607=414+193

Example





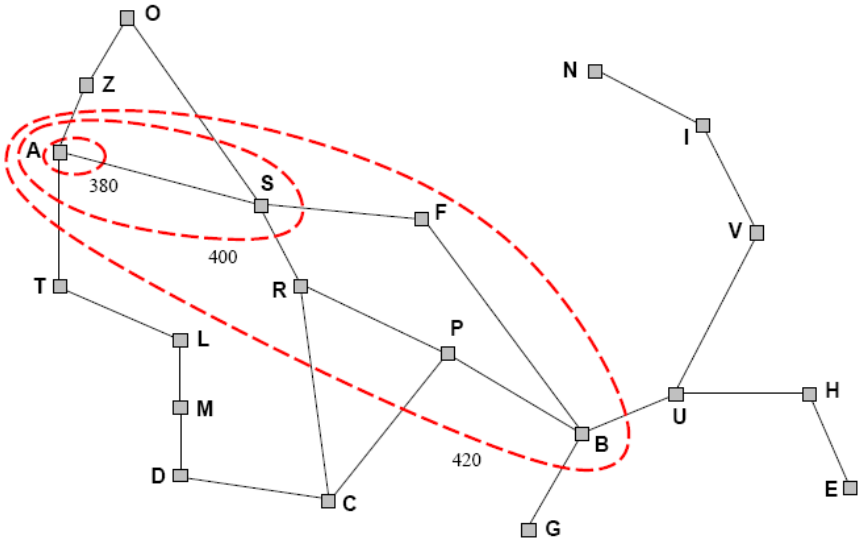
	OPEN	CLOSED
1	(S,-, 0+10)	
2	(B,S,4+1), (A,S,2+10)	(S,-, 0+10)
3	(D,B,9+1) , (A,S,2+10)	(S,-,0+10), (B,S,4+1)
4	(A,S,2+10), (E,D,14+1), (F,D,15+1)	(S,-,0+10), (B,S,4+1), (D,B,9+1)
5	(C,A,3+9), (E,D,14+1), (F,D, 15+1)	(S,-,0+10), (B,S,4+1), (D,B,9+1),(A,S, 2+10)
6	(D,C,4+1), (E,D,14+1), (F,D,15+1)	(S,-,0+10), (B,S,4+1), (A,S,2+10), (C,A,3+9) /* D is removed from CLOSED */
7	(E,D,9+1), (F,D,10+1)	(S,-,0+10), (B,S,4+1), (A,S,2+10), (C,A,3+9),(D,C,4+1)



STEP No.	OPEN	CLOSED
7	(E,D,9+1), (F,D,10+1)	(S,-,0+10), (B,S,4+1), (A,S,2+10), (C,A,3+9),(D,C,4+1)
8	(F,D,10+1), (G,E,12+0)	(S,-,0+10), (B,S,4+1), (A,S,2+10), (C,A,3+9),(D,C,4+1),(E,D,9+1)
9	**(G,F,11+0)**	(S,-,0+10), (B,S,4+1), (A,S,2+10), (C,A,3+9),(D,C,4+1),(E,D,9+1),(F,D,10+1)
TERMINATE with SUCCESS {You may need to build the path from S to G and return that with its cost (11)}		

A* Search

- A* expands nodes in increasing f value
  - Gradually adds f-contours of nodes (like breadth-first search adding layers)
  - Contour *i* has all nodes $f=f_i$ where $f_i < f_{i+1}$



A* Search

- Complete
  - Yes, unless there are infinitely many nodes with $f <= f(G)$
- Time
  - Exponential in [relative error of h x length of soln]
  - The better the heuristic, the better the time
    - Best case h is perfect, $O(d)$ i.e., Linear in d
    - Worst case $h = 0$, $O(b^d)$ same as BFS, i.e., exponential in d.
- Space
  - Keeps all nodes in memory and save in case of repetition
  - This is $O(b^d)$ or worse
  - A* usually runs out of space before it runs out of time
- Optimal
  - Yes, cannot expand f_{i+1} unless f_i is finished

Through a Lemma we prove this as a Theorem

A* IS ADMISSIBLE -- PROOF

Lemma 1: In OPEN there always exists a node m such that
g(m) = g(m).*

Proof: (by induction)

g(S) = g(S)* Basis {*Call this stage 1 of the algorithm*}

Let at stage *i* , in the OPEN we have node *n* such that *g(n) = g*(n).*

We show that at stage *i+1* also there must exist a node with this property.

Two cases, (i) *f(n*) is the least value in OPEN, (ii) *f(n)* is not least.

Case (i) : Node *n* is chosen for EXPANSION. Let *n1* be the successor of *n* with least edge cost among all successors of *n*.

Since for *n*, *g(n) = g*(n),* the least cost path from *S* to *n* has this cost, *g(n1) = g*(n1).* And, *n1* is in OPEN now.

Case (ii) : *n* is not chosen, hence is still in OPEN.

- **Corollary 1**: If h is an underestimated one, then in OPEN there always exists a node m such that $f(m) \leq C^*$.

  Proof: Since there exists m such that $g(m) = g^*(m)$,

  We get, $f(m) = g^*(m) + h(m)$

  $\leq g^*(m) + h^*(m) = C^*$.

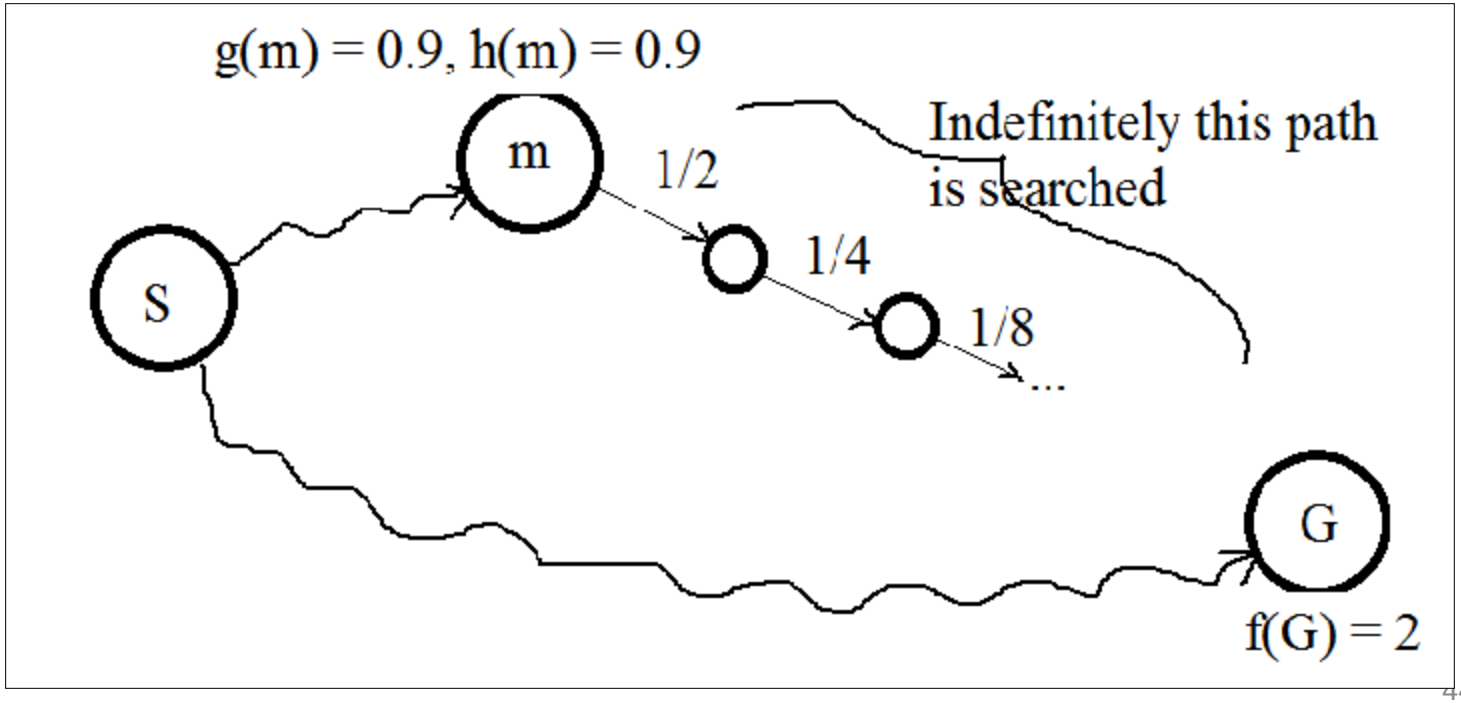
Theorem 1: If for any node *n, h(n) ≤ h*(n)* and cost of any edge is at-least ε, then A* is admissible. {Even for infinite graphs, if the conditions are satisfied then A* is admissible}

Proof: (*by contradiction*)

Let the algorithm returned a suboptimal path. Let the cost of this solution be *f(G) > C*.*
The algorithm found that G in the OPEN has least *f( )* value.
But, there is a node m in OPEN such that *f(m) ≤ C*.*
That is, there is a node m in OPEN with *f(m) < f(G).*
Hence, the contradiction.

Why cost of any edge is at-least ε is required?

- **Let f(G) = 2.**
- **Let f(m) = 1.8**
- **Assume m is the min cost one in OPEN.**
- **Let S->m ->… ->G is the optimal path.**
- **The algorithm will never reach G through this optimal path**.



Other Properties of A*

- Lemma 2: For every node *n* expanded by A*, $f(n) \leq C^*$.

Proof: by contradiction.

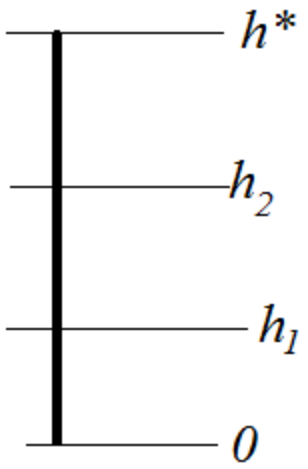
Suppose *p* is a node *s.t.* $f(p) > C^*$, but *p* is chosen for expansion. That is, $f(p)$ is the least value for nodes in OPEN.

But, there is a node *m* in OPEN such that
$f(m) \leq C^*$. *{as per Corollary 1}*

So, $f(p)$ is not the least value in OPEN. Hence *p* is not chosen.

A* Further Properties

- A* with more informed heuristic will do more focused search.

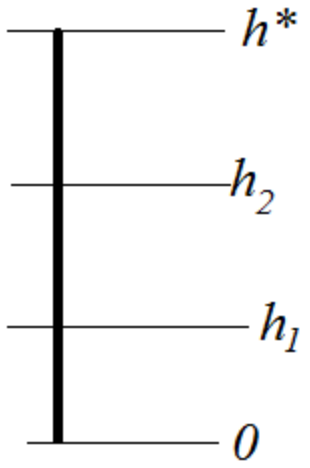


For each node n,

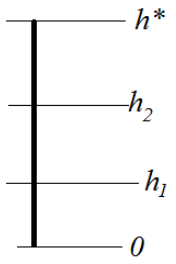
$$0 \leq h_1(n) \leq h_2(n) \leq h^*(n)$$

Then we say h_2 is ***more informed*** than h_1

(h_2 ***dominates*** h_1)



- A* with h_2 will visit less number nodes than that with h_1



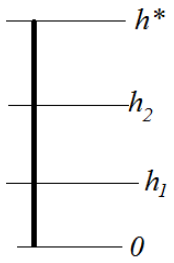
- Lemma 3: If A* with h_2 expands a node *n*, then A* with h_1 also expands *n.*
- *Proof: by induction.*

Basis: For the node *S,* the statement is true.

IH: For all nodes upto depth *k* from *S,* the statement is true.

IS: We have to show that for nodes at depth *k+1* the statement is true.

Induction Step Proof is by Contradition.



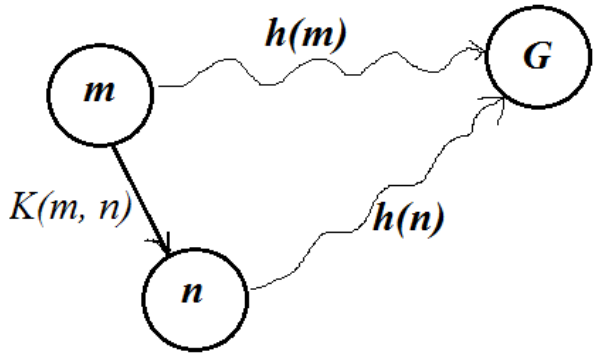
- Let there exist a node *L* at depth *k+1*, s.t., *L* is expanded by A* with h_2, but not with h_1 (That is, A* with h_1 terminated without expanding the node *L*).
- $f_2(L) \leq C^* \leq f_1(L)$
- $g_2(L)+h_2(L) \leq g_1(L) + h_1(L)$
- But, we know, $g_2(L) \geq g_1(L)$ {Since upto depth *k*, nodes visited by A* with h_2 are also visited by A* with h_1 }
- So, $g_2(L)+h_2(L) \leq g_2(L) + h_1(L)$
- So, $h_2(L) \leq h_1(L)$, which is **false**.

The Monotone (Consistency) Property

- For every node *m* and for any successor *n* of *m*, we have

  h(m) ≤ h(n) + k(m,n)

  where *k(m,n)* is the cost of the edge from *m* to *n*.
- More strict restriction than admissibility
- Similar to *triangle inequality*



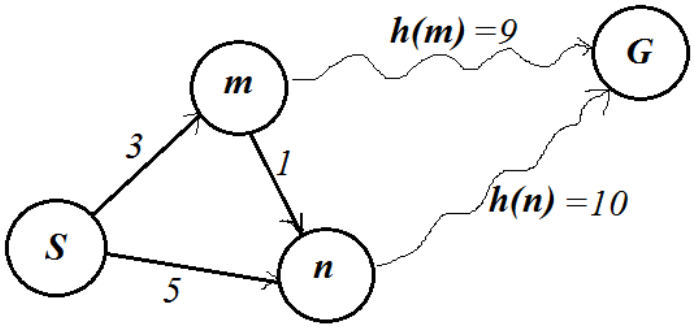
Advantages of Monotone Property

- If the monotone property holds for the heuristic function then at the time when A* picks a node n for expansion, we are guaranteed g(n) = g*(n).
- A closed node never becomes open …
  - A new node in OPEN, even though has a duplicate in CLOSED, it is safe to throw away the new node.
  - Number of duplicates generated will reduce.

    {*Recall, closed node becoming open means all its children also come to open list subsequently*}
- Proof is left as an Exercise (reading assignment).

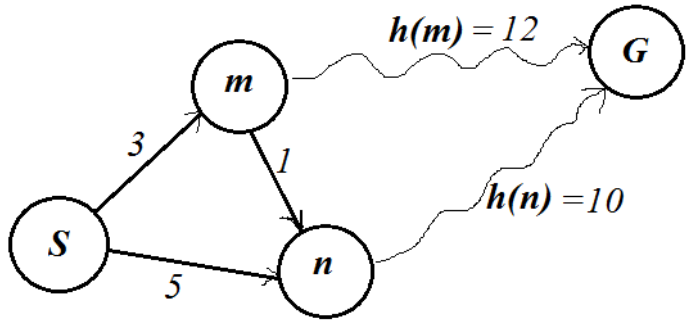
An illustration

- $h(m) \leq h(n) + k(m,n)$



Monotone Property Satisfied
m is Expanded, g(m)=g(m)=3*

- $h(m) \leq h(n) + k(m,n)$



Monotone Property Not Satisfied
n is Expanded, g(n) ≠ g(n) = 4*
g(n) = 5; n is closed
n can become open in future,
since its f value can reduce