BITS Pilani
Pilani Campus

Artificial & Computational Intelligence

DSE CLZG557

M2 : Problem Solving Agent using Search

Raja vadhana P
Assistant Professor,
BITS - CSIS

Course Plan

M1 Introduction to AI

M2 Problem Solving Agent using Search

M3 Game Playing, Constraint Satisfaction Problem

M4 Knowledge Representation using Logics

M5 Probabilistic Representation and Reasoning

M6 Reasoning over time, Reinforcement Learning

M7 AI Trends and Applications, Philosophical foundations

Module 2 : Problem Solving Agent using Search

A. Uninformed Search

B. Informed Search

C. Heuristic Functions

D. Local Search Algorithms & Optimization Problems

Problem Formulation

Problem Solving Agents

Goal Formulation

↓

Problem Formulation

↓

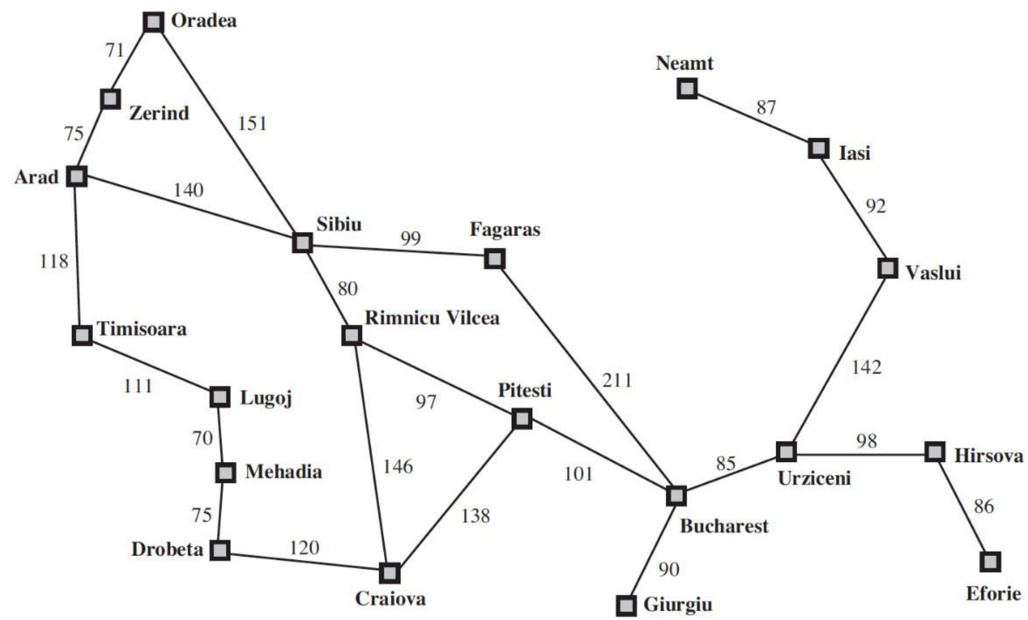
Search Phase

↓

Execution Phase

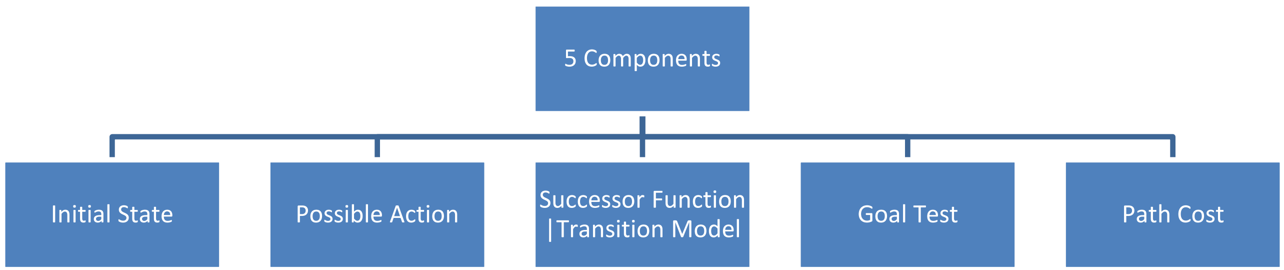
Phases of Solution Search by PSA

Assumptions – Environment :
Static
Observable
Discrete
Deterministic



Problem Solving Agents – Problem Formulation

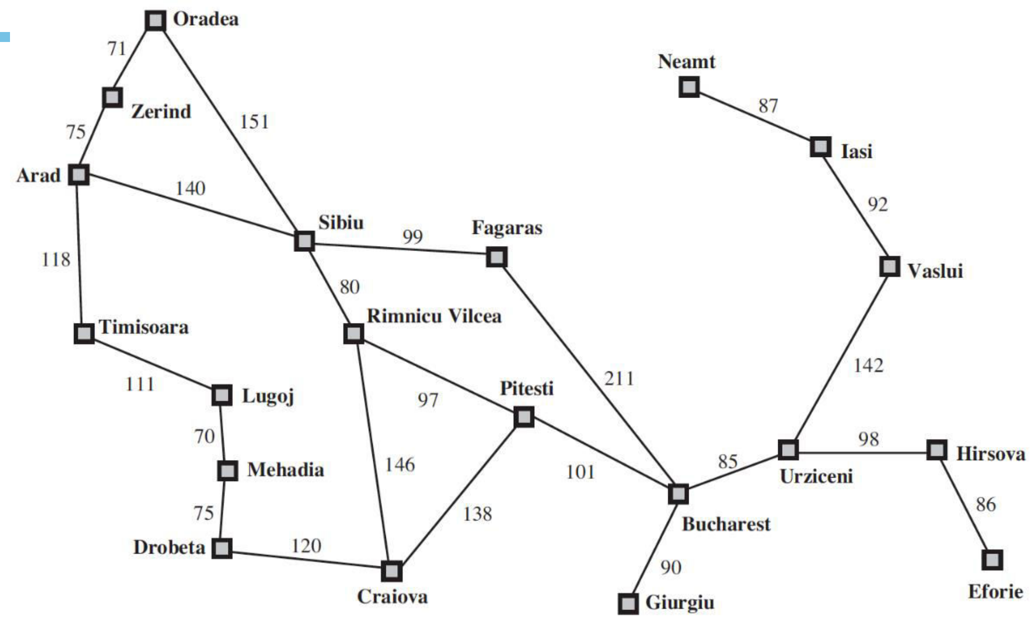
Abstraction Representation
Decide what actions under states to take to achieve a goal



A function that assigns a numeric cost to each path. A path is a series of actions. Each action is given a cost depending on the problem.

Solution = Path Cost Function + Optimal Solution

Problem Solving Agents – Problem Formulation



Initial State –E.g., *In(Arad)*

Possible Actions – *ACTIONS(s)* ➔ *{Go(Sibiu), Go(Timisoara), Go(Zerind)}*

Transition Model – *RESULT(In(Arad), Go(Sibiu)) = In(Sibiu)*

Goal Test – *IsGoal(In(Bucharest)) = Yes*

Path Cost – *cost(In(Arad), go(Sibiu)) = 140 kms*

Searching for Solutions

Choosing the current state, testing possible successor function, expanding current state to generate new state is called Traversal. Choice of which state to expand – Search Strategy

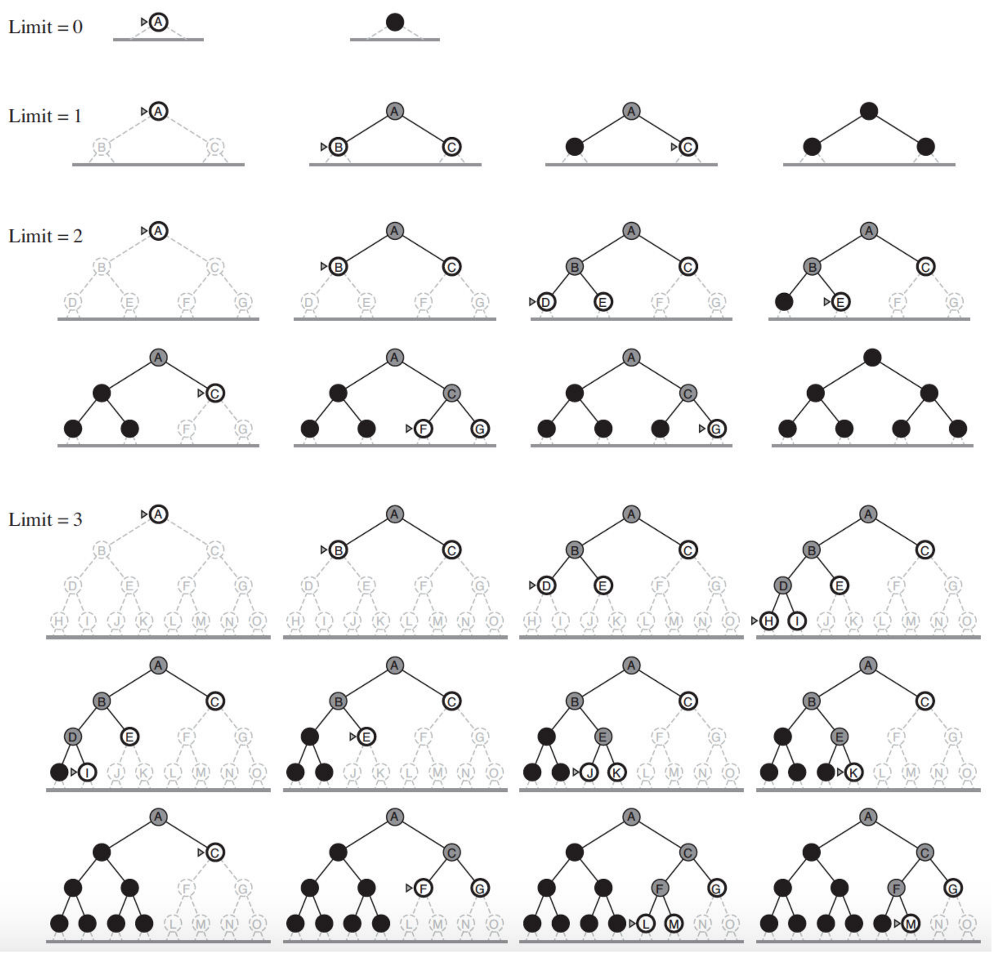
Search Strategy (under certainty)

- **Uninformed**
  - BFS, DFS, UCS
  - IDS, DLS
  - Bi-Directional
- **Informed**
  - Best First Search
  - A*
  - AO*

Uninformed Search

- BFS & its Variants
- DFS & its Variants

Iterative Deepening Depth First Search (IDS)



Iterative Deepening Depth First Search (IDS)

Run Depth Limited Search (DLS) by gradually increasing the limit l

- First with l=1, then l=2, l=3 and so on – until goal is found

Its is a combination of Depth First Search + Breadth First Search

Like DFS, memory requirement is a modest $\mathcal{O}(bd)$ where d is the depth of shallowest goal

Like BFS

- Complete when branching factor is finite
- Optimal when path cost is non decreasing function of depth

Iterative Deepening Depth First Search (IDS)

Time Complexity:

- Appears that IDS is generating a lot of nodes multiple times
- However, most of nodes are present in the lower levels which are not repeated often
- Generation of nodes
  - At level 1 - b nodes generated d times – (d)b
  - At level 2 – b^2 nodes generated d-1 times – $(d-1)b^2$
  - At level d – b^d nodes generated once – $(1)\ b^d$
- Time Complexity N(IDS) = $\mathcal{O}(b^d)$ same as BFS

IDS is the preferred uninformed search method when search space is large and depth is unknown innovate achieve lead

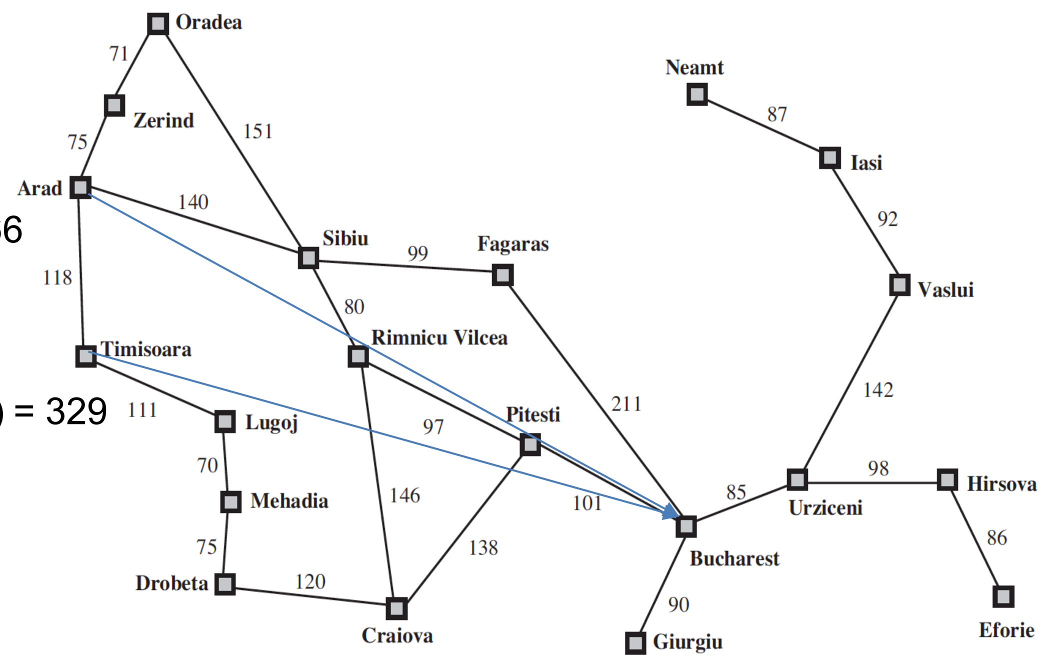
Informed Search
Greedy Best First
A*

Informed /Heuristic Search

Strategies that know if one non-goal state is more promising than another non-goal state

h(Arad) = 366

h(Timisoara) = 329

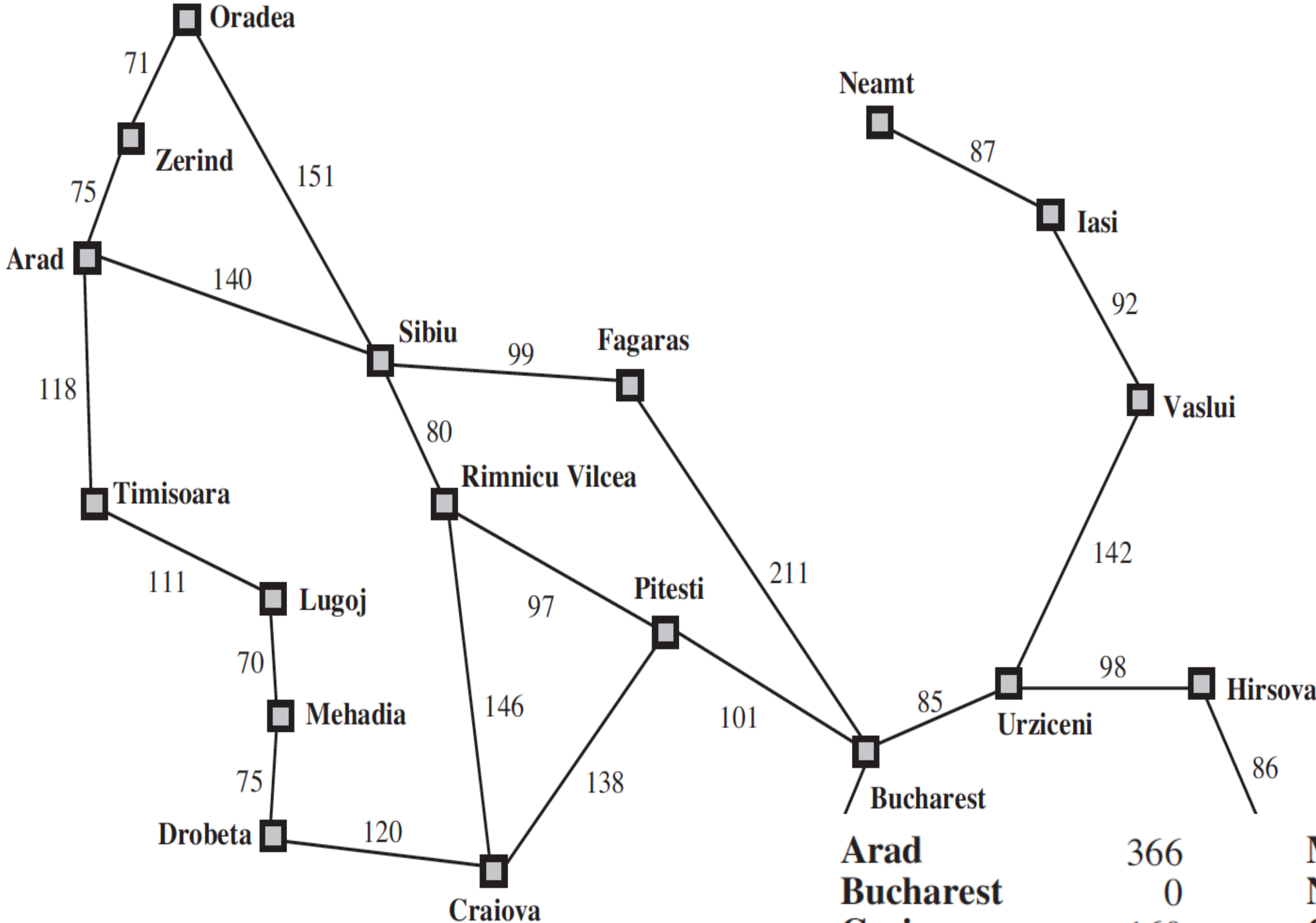


Greedy Best First Search

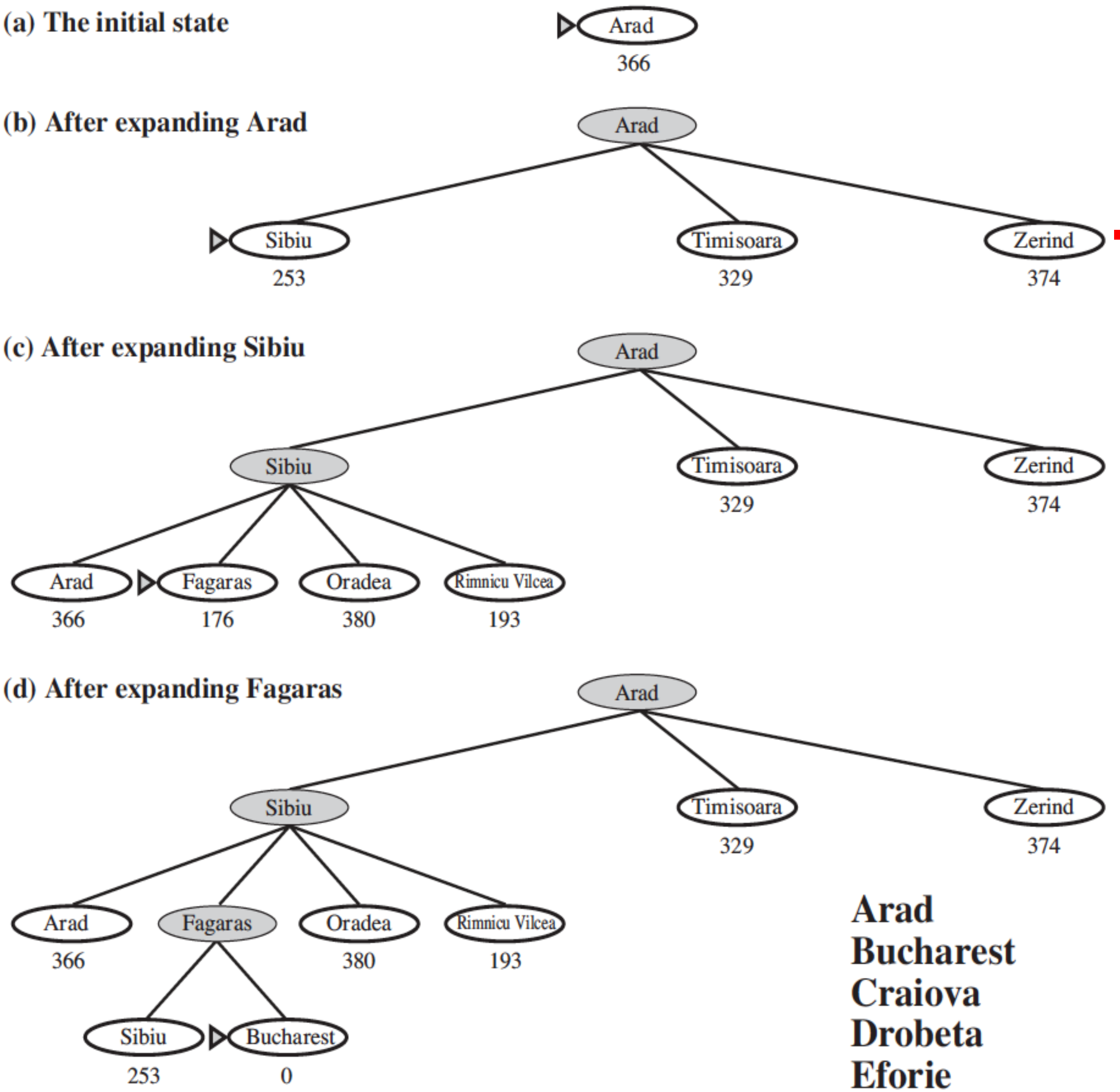
Expands the node that is closest to the goal

Thus, **f(n) = h(n)**

Arad	366	**Mehadia**	241
Bucharest	0	**Neamt**	234
Craiova	160	**Oradea**	380
Drobeta	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

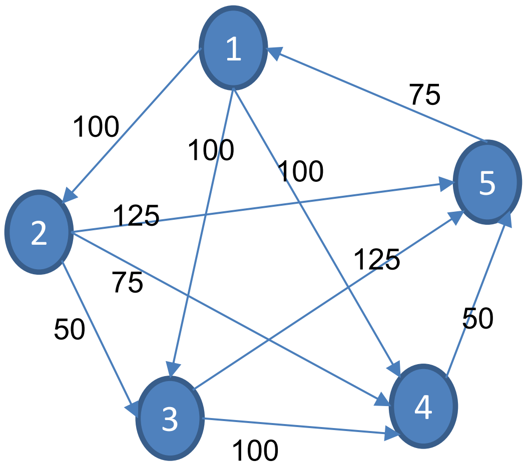


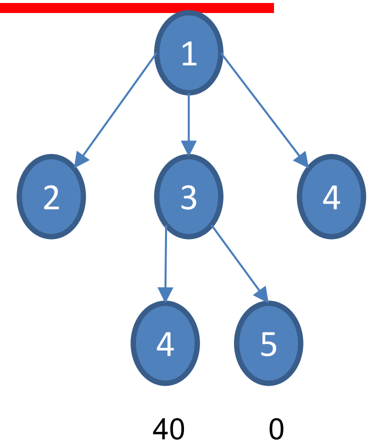
Arad	366	Mehadia	241
Bucharest	0	Neamt	234
Craiova	160	Oradea	380
Drobeta	242	Pitesti	100
Eforie	161	Rimnicu Vilcea	193
Fagaras	176	Sibiu	253
Giurgiu	77	Timisoara	329
Hirsova	151	Urziceni	80
Iasi	226	Vaslui	199
Lugoj	244	Zerind	374



Arad	366	**Mehadia**	241
Bucharest	0	**Neamt**	234
Craiova	160	**Oradea**	380
Drobeta	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

Greedy Best First Search





n	h(n)
1	60
2	120
3	30
4	40
5	0

(1)
(1 3) (1 4) (1 2)
(1 3 5) (1 3 4)

C(1-3-5) = 100 + 125 =225
Expanded : 2
Generated : 6
Max Queue Length : 3
Idea: Optimize DFS. Choose next nearest to goal in the same hill.

Greedy Best First Search

Not Optimal

- Because the algorithm is greedy
- It only optimizes for the current action

Not Complete

- Often ends up in state with a dead end as the heuristic doesn’t guarantee a path but is only an approximation

Time and Space Complexity - $\mathcal{O}(b^m)$ where m – max depth of search tree

A* Search

Expands the node which lies in the closest path (estimated cheapest path) to the goal

Evaluation function **f(n) = g(n) + h(n)**

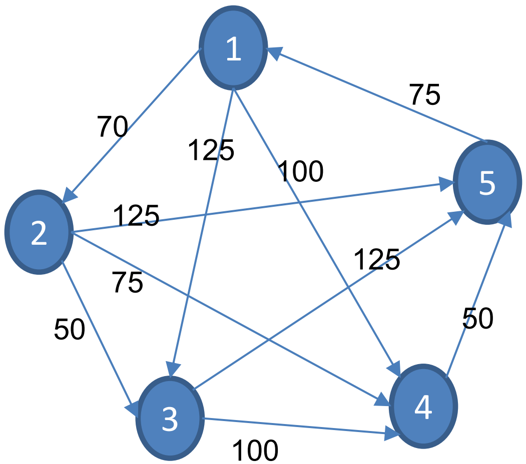
g(n) – the cost to reach the node

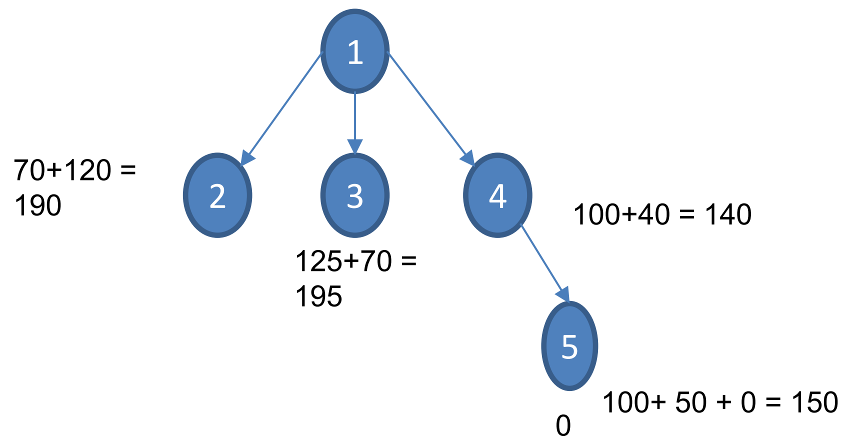
h(n) – the expected cost to go from node to goal

f(n) – estimated cost of cheapest path through node n

Arad	366	**Mehadia**	241
Bucharest	0	**Neamt**	234
Craiova	160	**Oradea**	380
Drobeta	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

A* Search





n	h(n)
1	60
2	120
3	70
4	40
5	0

(1)
(1 4) (1 2) (1 3)
(1 4 5) (1 2) (1 3)

C(1-4-5) = 100 + 150 =150
Expanded : 2
Generated : 5
Max Queue Length : 3

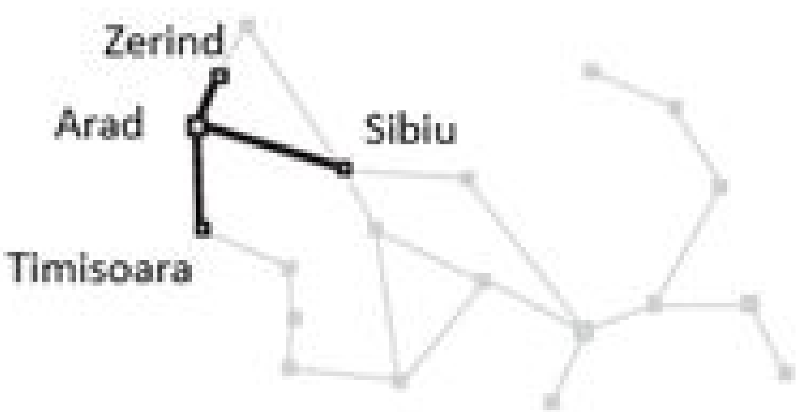
Tree Search Vs Graph Search

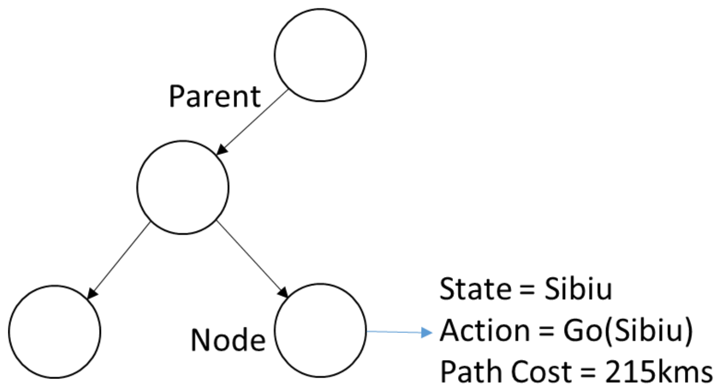
Informed /Heuristic Search

Coding Aspects

For each node n of the tree,

n.STATE	: the state in the state space to which node corresponds
n.PARENT	: the node in the search tree that generated this node
n.ACTION	: the action that was applied to parent to generate the node
n.PATH-COST	: the cost, denoted by g(n), of the path from initial state to node





Tree Search Algorithms

```
function Tree-Search (problem, strategy) returns a solution, or failure
  initialize the search tree using the initial state of problems
  loop do
    if there are no candidate for expansion
          then return failure
    choose: leaf node for expansion according to strategy
    if the node contains a goal state
          then return the corresponding solution
    else
          Expand the node
          Add the resulting nodes to the search tree
  end
```

Tree Search Vs Graph Search Algorithms

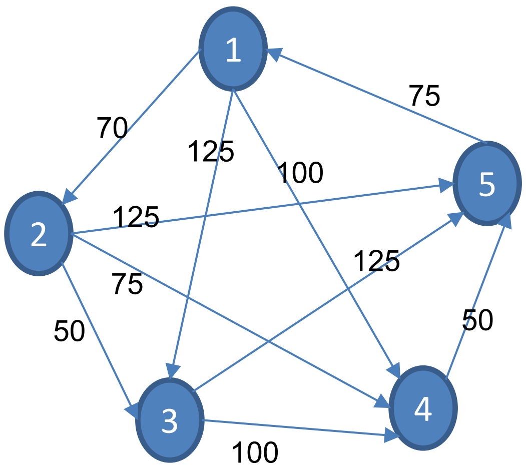
innovate achieve lead

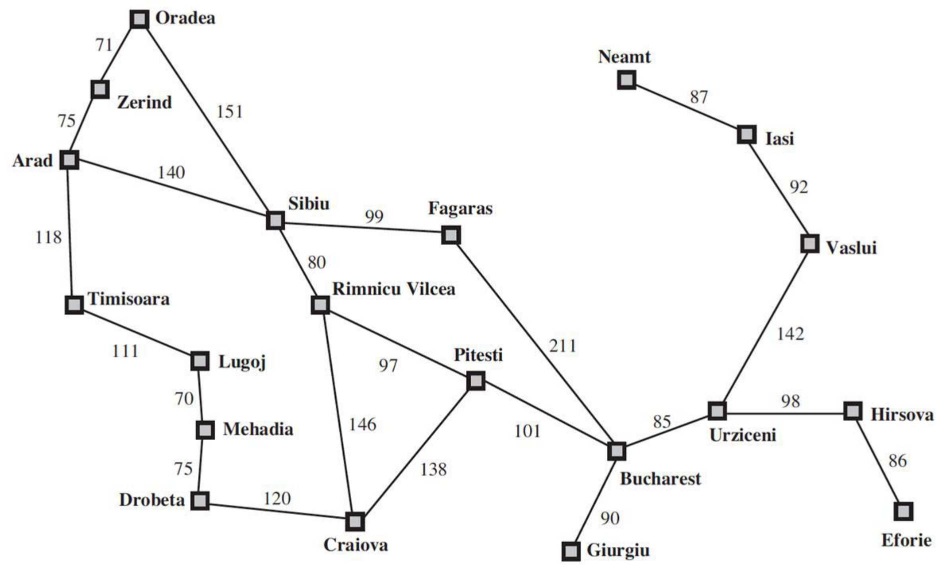
Coding Aspects

Need:

Redundant Path Problem : More than one way to reach a state from another.

Infinite Loop Path Problem





Start : 1

Goal : 3

Start : Arad

Goal : Craiova

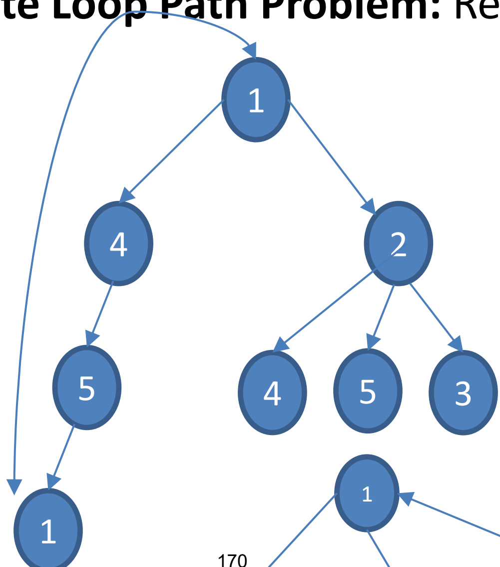
Tree Search Vs Graph Search Algorithms

Coding Aspects

Need:

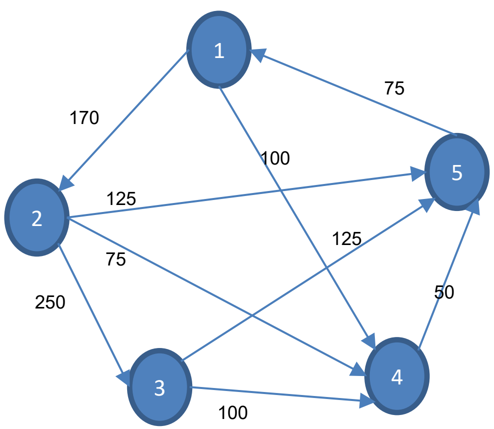
Redundant Path Problem

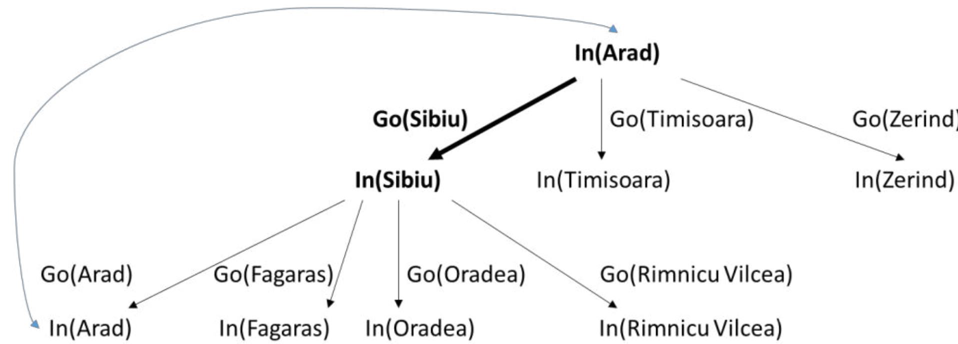
Infinite Loop Path Problem: Repeated State generated by looped path existence.



Start : 1

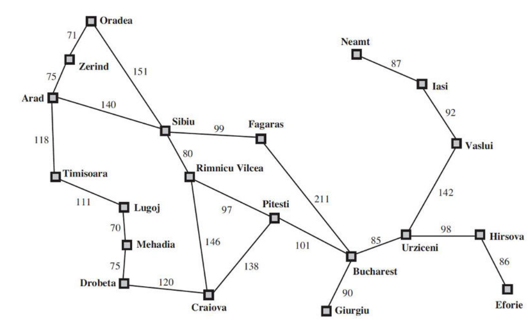
Goal : 3





Start : Arad

Goal : Craiova



Coding Aspects

For each node n of the tree,

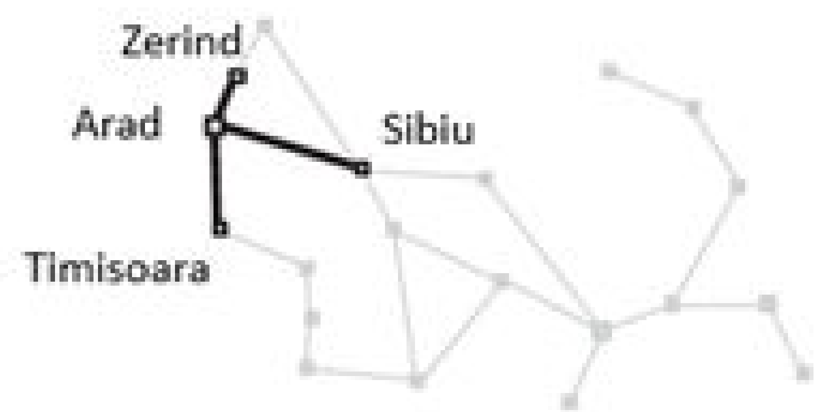
n.STATE : the state in the state space to which node corresponds

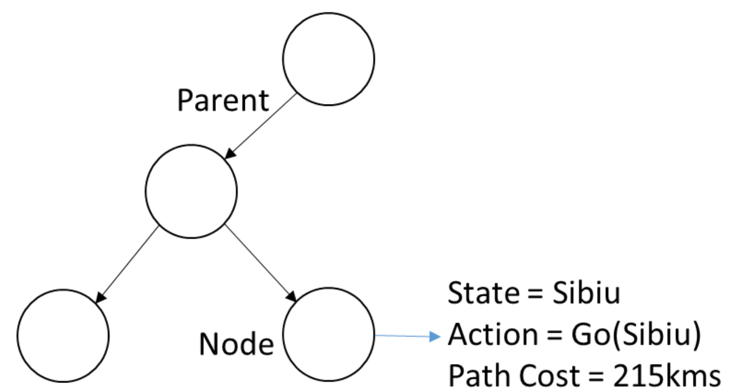
n.PARENT : the node in the search tree that generated this node

n.ACTION : the action that was applied to parent to generate the node

n.PATH-COST : the cost, denoted by g(n), of the path from initial state to node

n.VISITED : the boolean indicating if the node is already visited and tested (**or)** a global SET of visited nodes

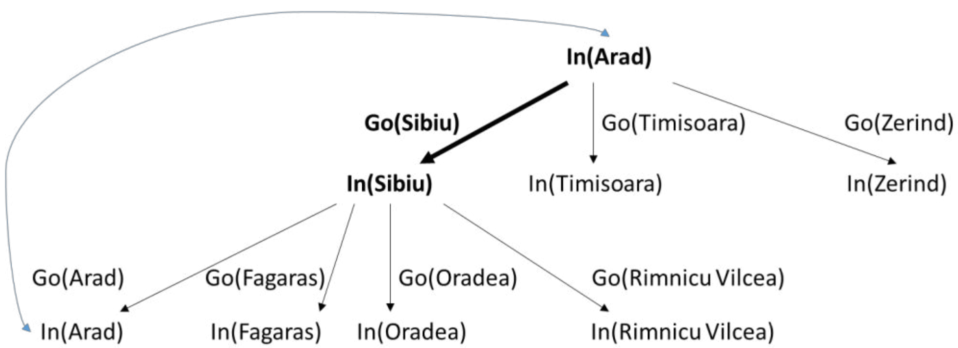




Coding Aspects

Graph-Search Algorithm

Augments the Tree-Search algorithm to solve redundancy by keeping track of states that are already visited called as Explored Set. Only one copy of each state is maintained/stored.



Graph Search Algorithms

```
function Graph-Search (problem, fringe) returns a solution, or failure
  initialize the search space using the initial state of problems memory to store the visited
fringe
  closed ← an empty set
  fringe ← Insert(Make-Node(Initial-State[problem]), fringe)
  loop do
    if fringe is empty
          then return failure
    node← Remove-Front(fringe)
    if the node contains a goal state
          then return the corresponding solution
    else
          if the node is not in closed ie., not visited yet
              Add the node to the closed set
              Expand all the fringe of the node
              Add all expanded sorted successors into the fringe
  end
```

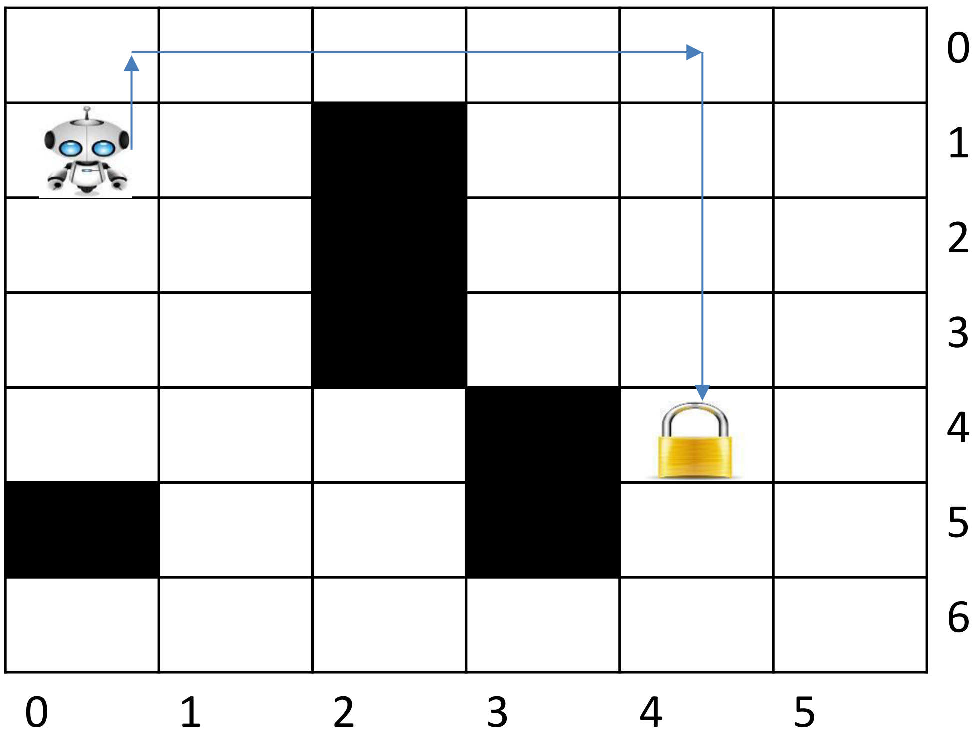
Path finding Robot

Successor Function Design

N-W-E-S

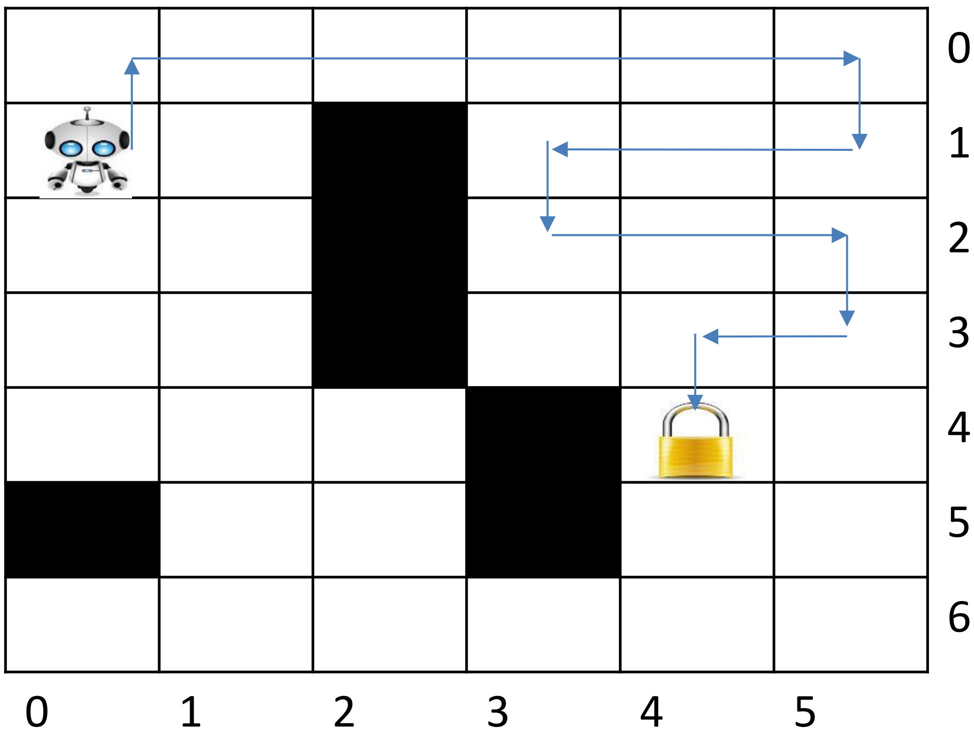
BFS

Demo



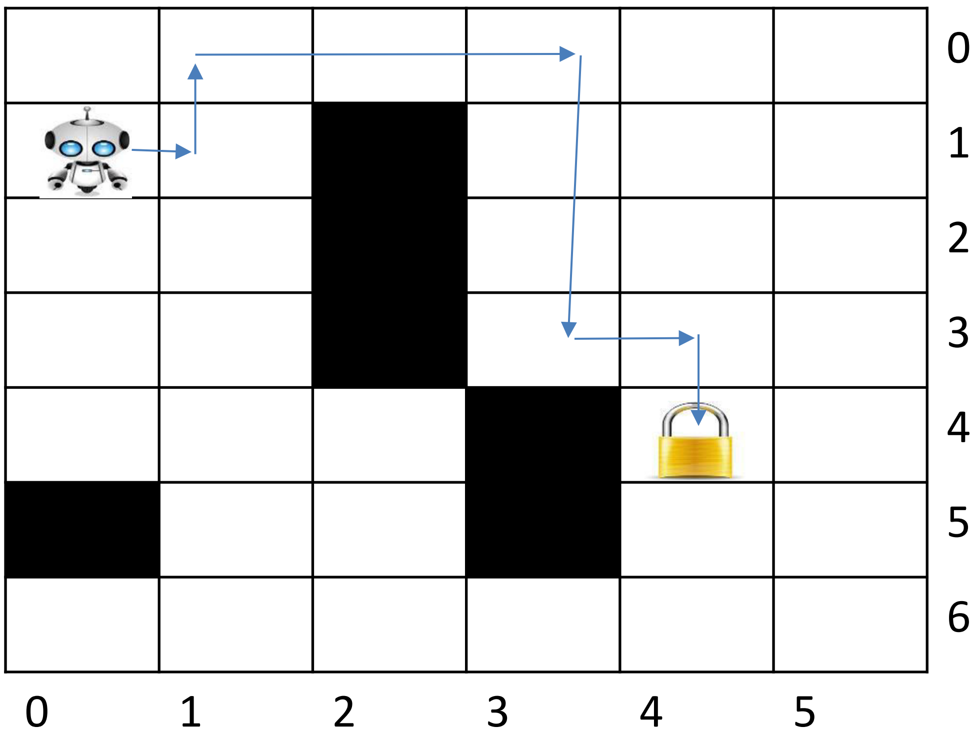
DFS :

Demo



A*

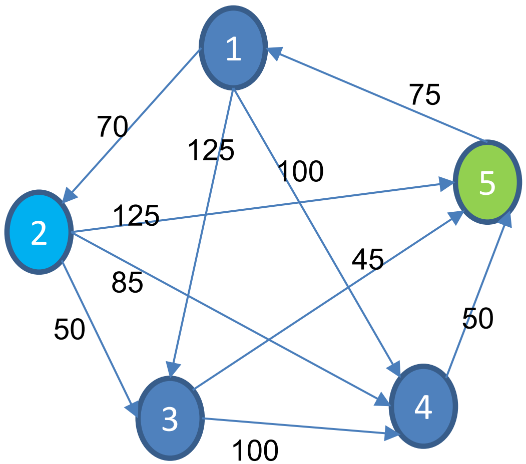
Demo

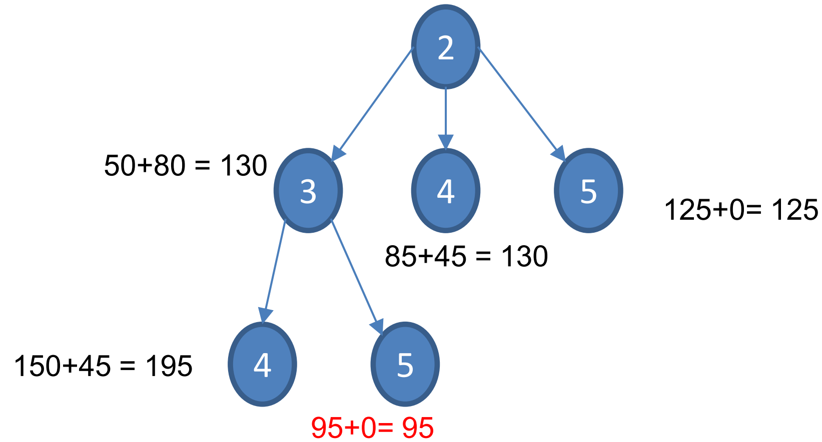


Optimality of A*

A* Search

Test for Optimality





n	h(n)
1	60
2	120
3	80
4	45
5	0

(2)
(2 5) (2 3) (2 4)

Can we make A* Optimal?

Test for Admissibility

Expands the node which lies in the closest path (estimated cheapest path) to the goal

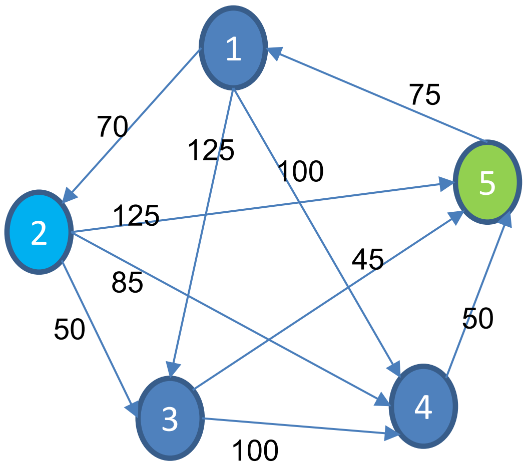
Evaluation function **$f(n) = g(n) + h(n)$**

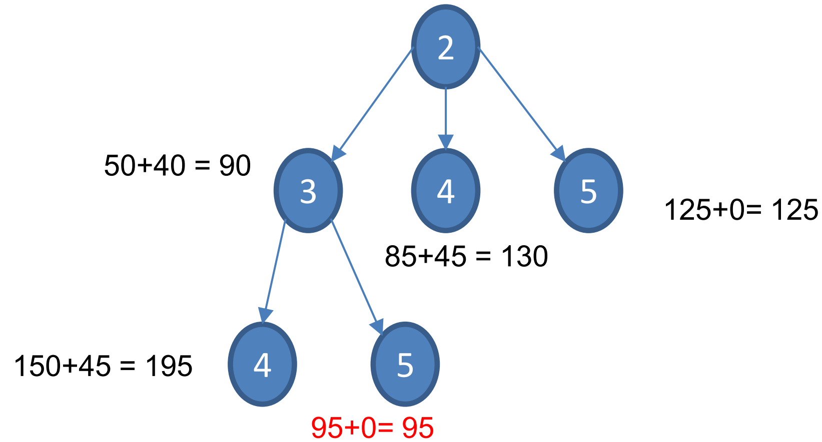
- $g(n)$ – the cost to reach the node
- $h(n)$ – the expected cost to go from node to goal
- $f(n)$ – estimated cost of cheapest path through node n

A heuristic is admissible or optimistic if , **$0 \le h(n) \le h^*(n)$**, where $h^*(n)$ is the actual cost to reach the goal

A* Search

Check for Optimality in the presence of Admissible heuristics





n	h(n)
1	60
2	120
3	40
4	45
5	0

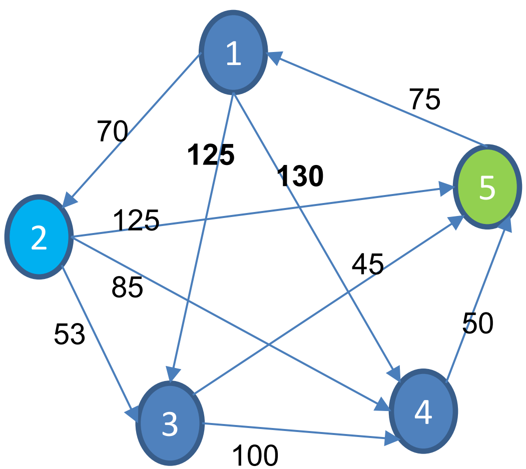
(2)
TEST-F
(2 3) (2 5) (2 4)
TEST-F

(2 3 5) (2 3 4) (2 5) (2 4)
TEST-P

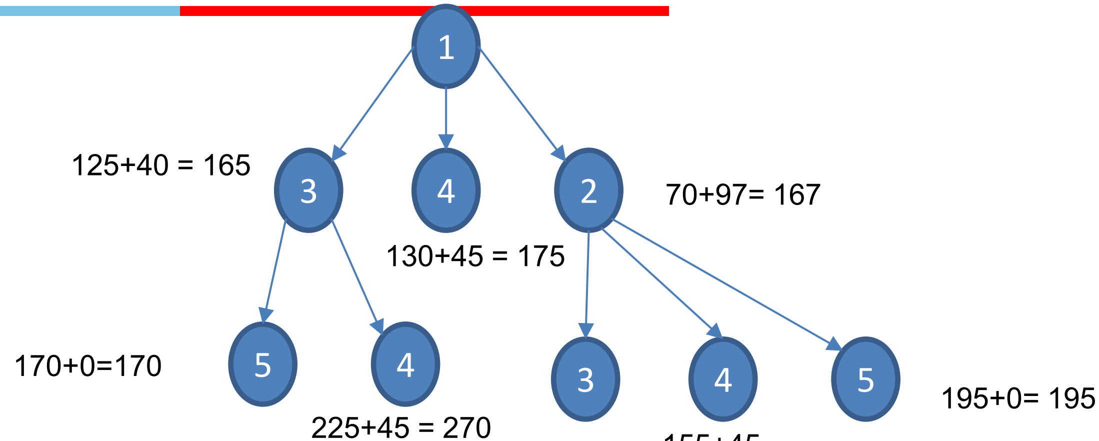
(2 3 5) is de-queued first => A* is optimal if h(n) is admissible

A* Search

Is the heuristics consistent?



n	h(n)
1	60
2	97
3	40
4	45
5	0



(1)
TEST-F
(1 3) (1 2) (1 4)
TEST-F

(1 2) (1 3 5) (1 4)(1 3 4)
(TEST-F)

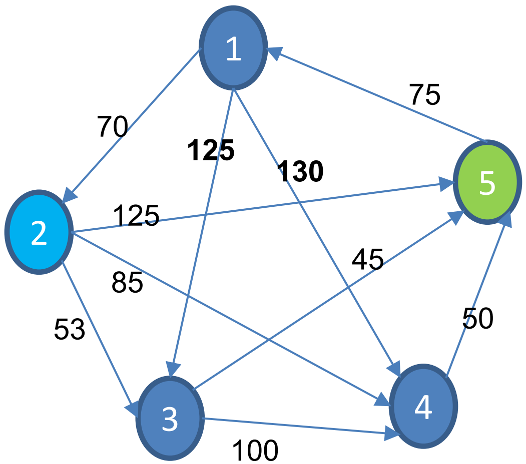
(1 2 3) (1 3 5) (1 4) (1 2 5) (1 2 4) (1 3 4)
(TEST-F)

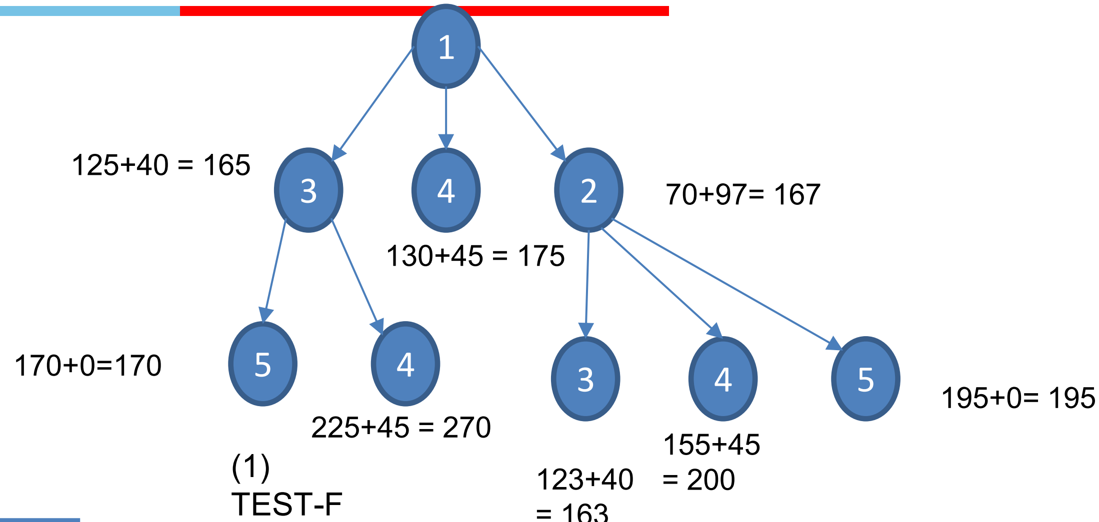
(1 3 5) (1 4) (1 2 5) (1 2 4) (1 3 4)
TEST-P

Even though in this new reduced h(2) though promising is restricted in graph search algorithms!!!!!
1-2-3-5 = 168
1-3-5 = 170

A* Search

Is the heuristics consistent?





n	h(n)	(1-2-3-4-5)
1	60	h(1)-h(2) ≤ g(1-2) → 60 ≤ g(1-2) +h(2) 60 ≤ 70+120
2	97	h(2)-h(3) ≤ g(2-3) 97 ≤ 53+40
3	40	h(3)-h(4) ≤ g(3-4) 40 ≤ 100+45
4	45	h(4)-h(5) ≤ g(4-5) 45 ≤ 50+0
5	0	

(1)
TEST-F
(1 3) (1 2) (1 4)
TEST-F

(1 2) (1 3 5) (1 4)(1 3 4)
(TEST-F)

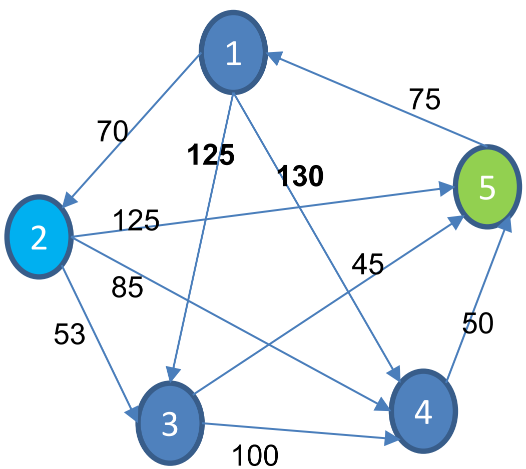
(1 2 3) (1 3 5) (1 4) (1 2 5) (1 2 4) (1 3 4)
(TEST-F)

(1 3 5) (1 4) (1 2 5) (1 2 4) (1 3 4)
TEST-P

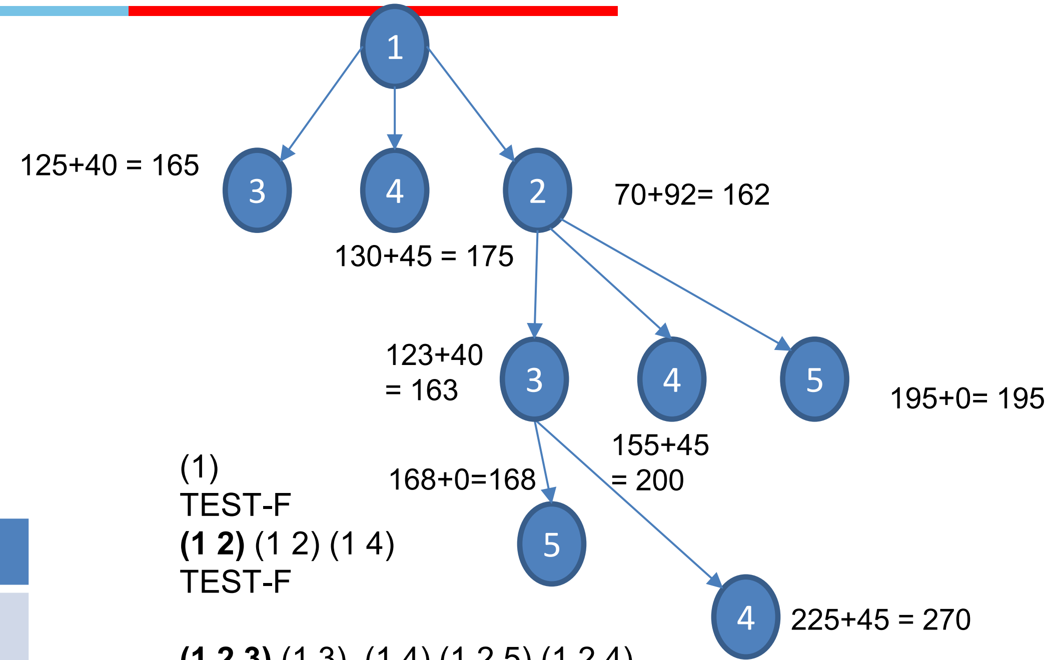
Even though in this new reduced h(2) though promising is restricted in graph search algorithms!!!!!

A* Search

Does the consistent heuristics work?



n	h(n)	(1-2-3-4-5)
1	60	h(1)-h(2) ≤ g(1-2) → 60 ≤ g(1-2) +h(2) 60 ≤ 70+120
2	**92**	h(2)-h(3) ≤ g(2-3) 92 ≤ 53+40
3	40	h(3)-h(4) ≤ g(3-4) 40 ≤ 100+45
4	45	h(4)-h(5) ≤ g(4-5) 45 ≤ 50+0
5	0	



(1)
TEST-F
(1 2) (1 2) (1 4)
TEST-F

(1 2 3) (1 3) (1 4) (1 2 5) (1 2 4)
(TEST-F)

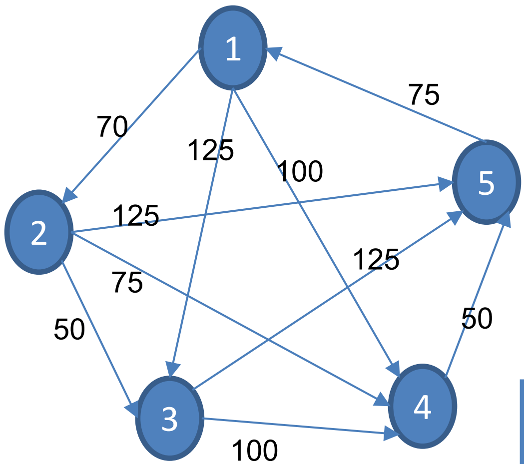
(1 3) (1 2 3 5) (1 4) (1 2 5) (1 2 4) (1 2 3 4)
(TEST-F)

(1 2 3 5) (1 4) (1 2 5) (1 2 4) (1 2 3 4)
TEST-P

Inference:
The value of evaluation function along the path toward the goal never decreases

A* Search

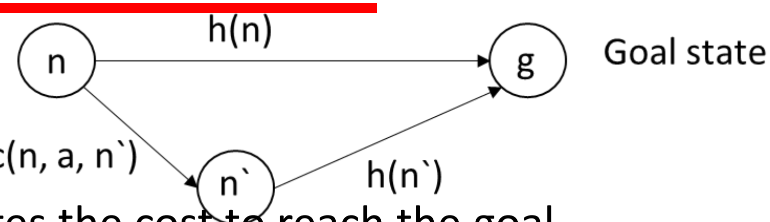
Is the heuristic designed leads to optimal solution?



---

n	h(n)	Is Admissible? h(n) <= h*(n)	Is Consistent? For every arc (i,j): h(i) <= g(i,j) + h(j)
1	80	Y	N (5→1) : 190<=155
2	60	N	Y (1→2) : 80<=130
3	0	Y	
4	200	Y	Y (1→4) : 80<=300 Y (2→4) : 60<=275
5	190	Y	Y (2→5) : 60<=315 Y (4→5) : 200<=240

A* Search



<u>Optimal on condition</u>

h(n) must satisfies two conditions:

- Admissible Heuristic – one that never overestimates the cost to reach the goal
- Consistency – A heuristic is consistent if for every node n and every successor node n` of n generated by action a, h(n) <= c(n, a, n`) + h(n`)

<u>Complete</u>

- If the number of nodes with cost <=C* is finite
- If the branching factor is finite
- A* expands no nodes with f(n) > C*, known as pruning

<u>Time Complexity</u> - $\mathcal{O}\left(b^{\Delta}\right)$ where the absolute error $\Delta = h^{*} - h$

Learning Objective: Students should be able to ,

1. Create Search tree for given problem
2. Design and compare heuristics apt for given problem
3. Apply BFS & A* algorithms to the given problem
4. Differentiate between uninformed and informed search requirements
5. Differentiate between Tree and Graph search
6. Prove if the given heuristics are admissible and consistent

Next class Plan

- Variations of A*
- Design of Heuristics
- Local Search (Start)

Required Reading: AIMA - Chapter # 3.3, 3.4, 3.5

Thank You for all your Attention

Note : Some of the slides are adopted from AIMA TB materials