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

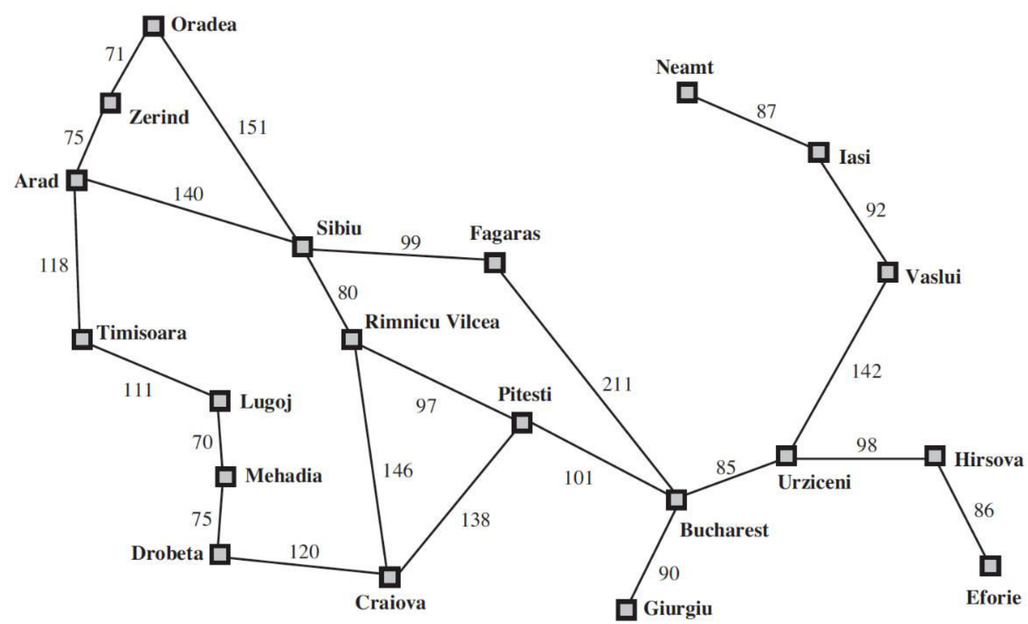
Problem Formulation

Problem Solving Agents

Goal Formulation → Problem Formulation → Search Phase → Execution Phase

Phases of Solution Search by PSA

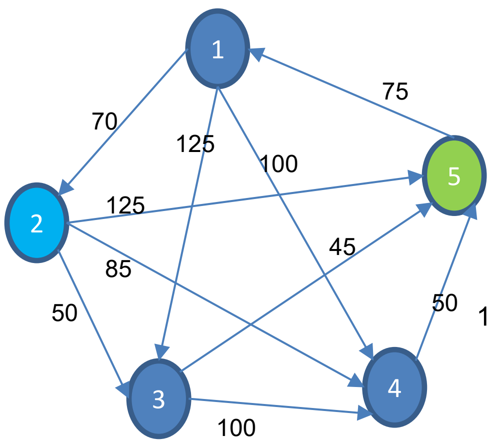
Assumptions – Environment :
Static
Observable
Discrete
Deterministic

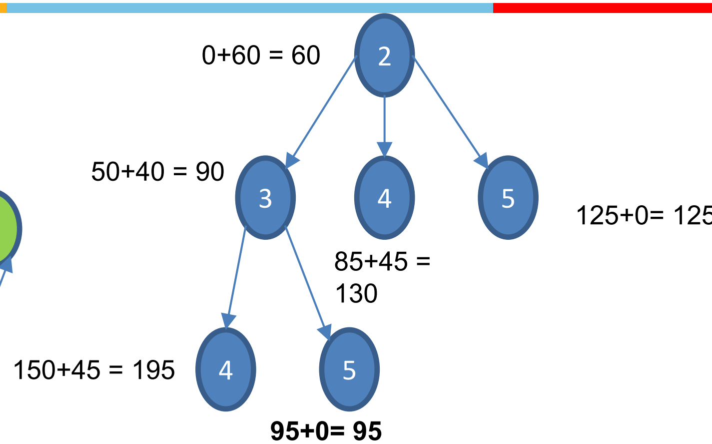


Optimality of A*

Generalize the Theorem

Check for Optimality in the presence of Admissible heuristics





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

Assume:
Optimal Goal Node A = 2-3-5
Suboptimal Goal Node B = 2-5
h(n) is admissible for all 'n' state w.r.t to Goal node 5

To Prove:
A* is optimal

Proof:

Let A & B be two goals expanded in a A* Tree
Let there be a node 'n' such that ancestor(A) ={n,…….}

<u>Step 1:</u>

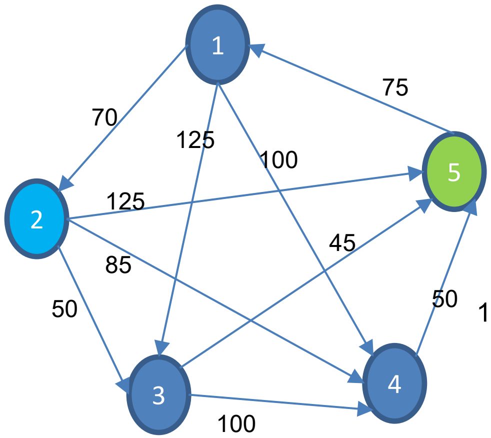
f(n) ≤ g(A) + h(A)

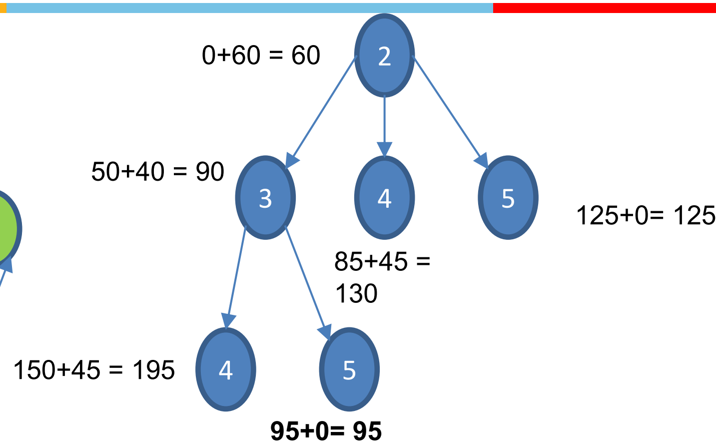
≤ g(A) [Recall h(n) ≤ h*(n) & A=Goal]

→ f(n) ≤ f(A) → eq.1

Generalize the Theorem

Check for Optimality in the presence of Admissible heuristics





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

Assume:
Optimal Goal Node A = 2-3-5
Suboptimal Goal Node B = 2-5
h(n) is admissible for all 'n' state w.r.t to Goal node 5

To Prove:
A* is optimal

Proof:

Let A & B be two goals expanded in a A* Tree
Let there be a node 'n' such that ancestor(A) ={n,…….}

<u>Proved So far:</u>
f(n) ≤ f(A) → eq.1

<u>Step 2:</u>

g(A) < g(B)

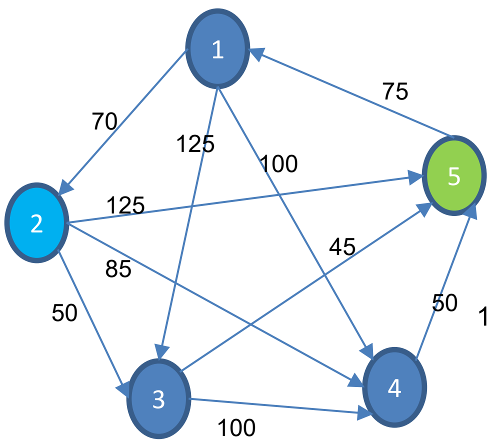
g(A) + 0 < g(B) + 0

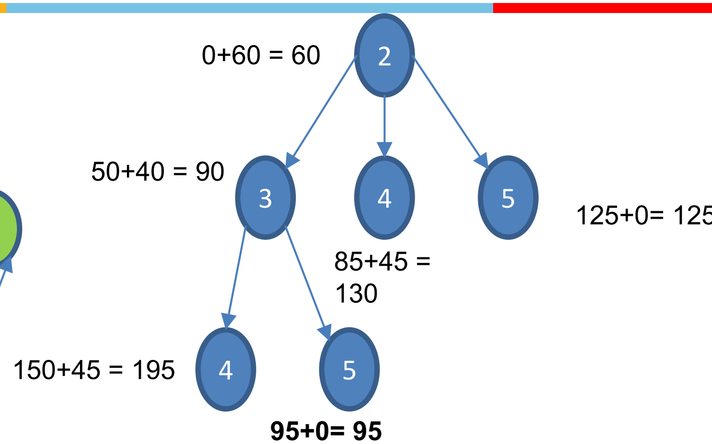
g(A) + h(A) < g(B) + h(B)

f(A) < f(B) → eq.1

Generalize the Theorem

Check for Optimality in the presence of Admissible heuristics





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

Assume:
Optimal Goal Node A = 2-3-5
Suboptimal Goal Node B = 2-5
h(n) is admissible for all 'n' state w.r.t to Goal node 5

To Prove:
A* is optimal

Proof:

Let A & B be two goals expanded in a A* Tree
Let there be a node 'n' such that ancestor(A) ={n,…….}

Proved So far:
f(n) ≤ f(A) → eq.1
f(A) < f(B) → eq.2

Step 3:
From equations 1 & 2

f(n) ≤ f(A) < f(B)

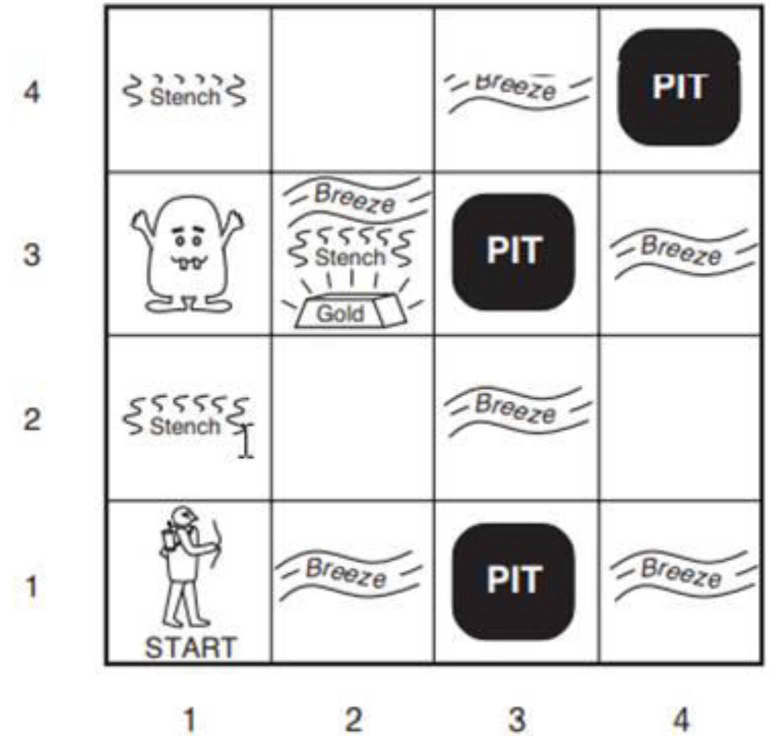
n is expanded before B
Hence the all ancestors of A and A is expanded before B . Hence A leaves the queue first.
Proved.

Learning Objective

Module 2 – so far

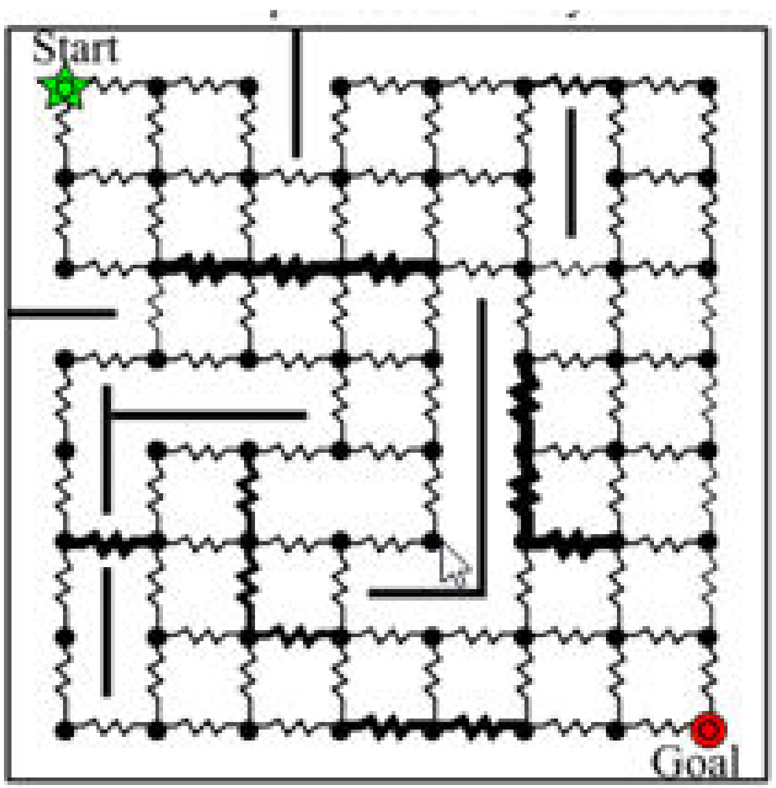
1. Create Search tree for given problem
2. Design and compare heuristics apt for given problem
3. Apply BFS/DFS & A* algorithms to the given problem
4. Differentiate between uninformed and informed search requirements
5. Differentiate between Tree and Graph search
6. Prove if the given heuristics are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

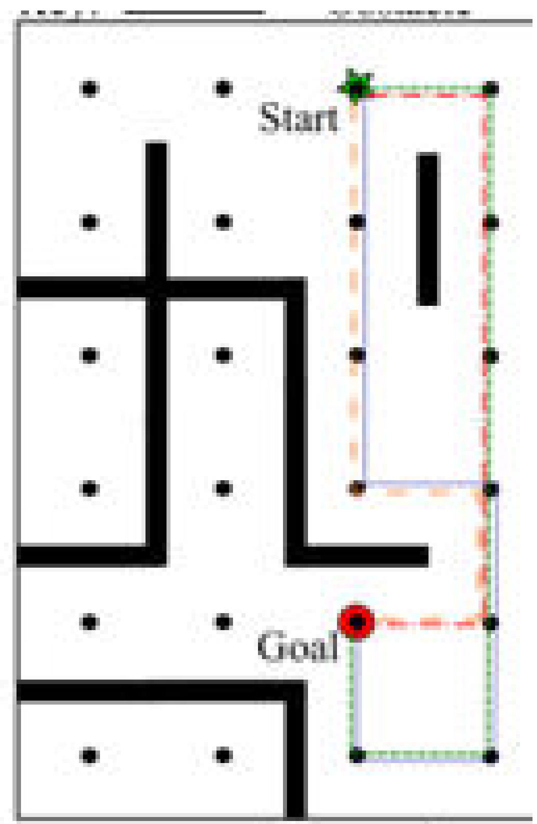
Designing Search Problem



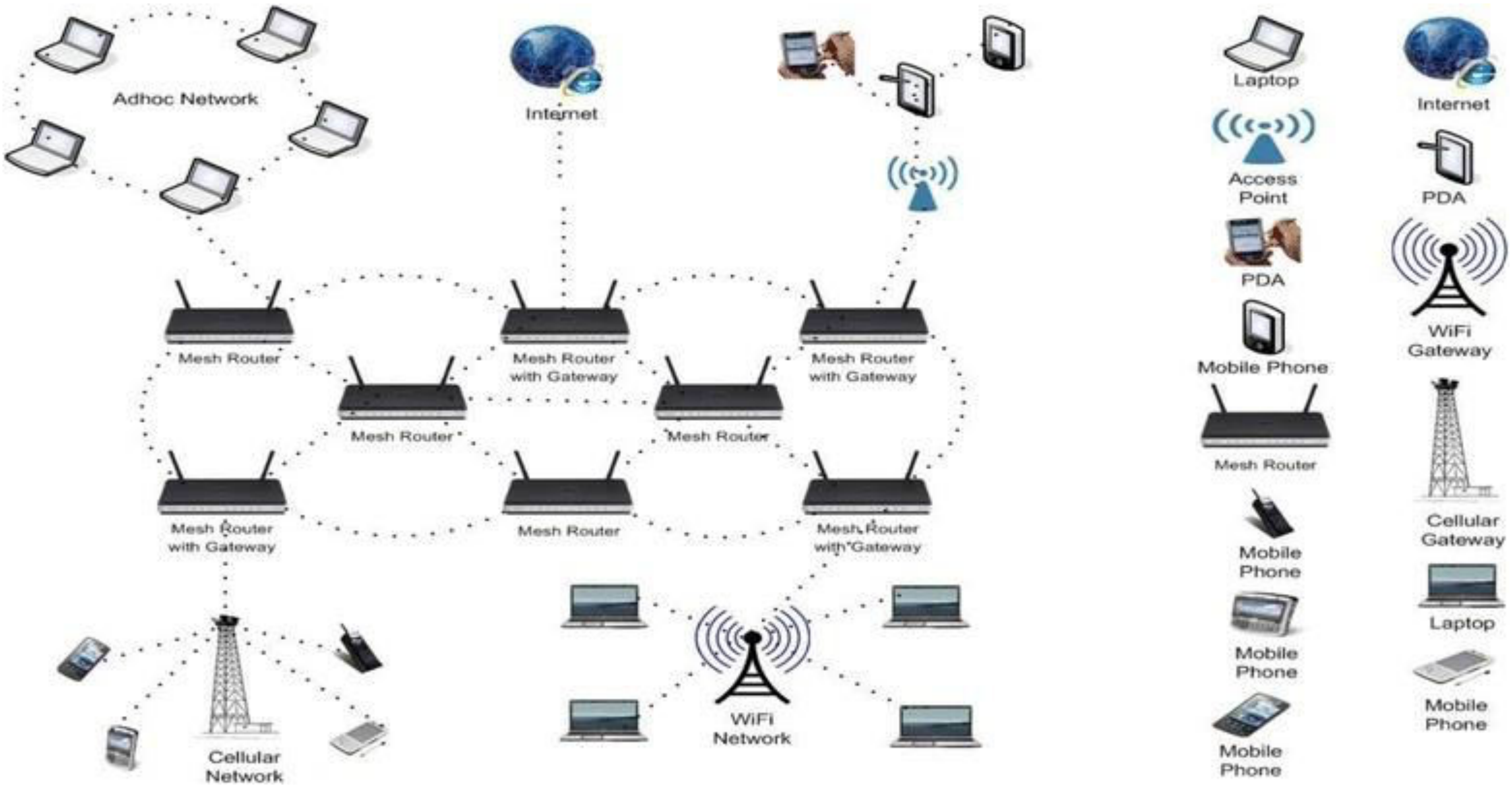
Recommend Heuristics Design

Identify the most appropriate Search Technique

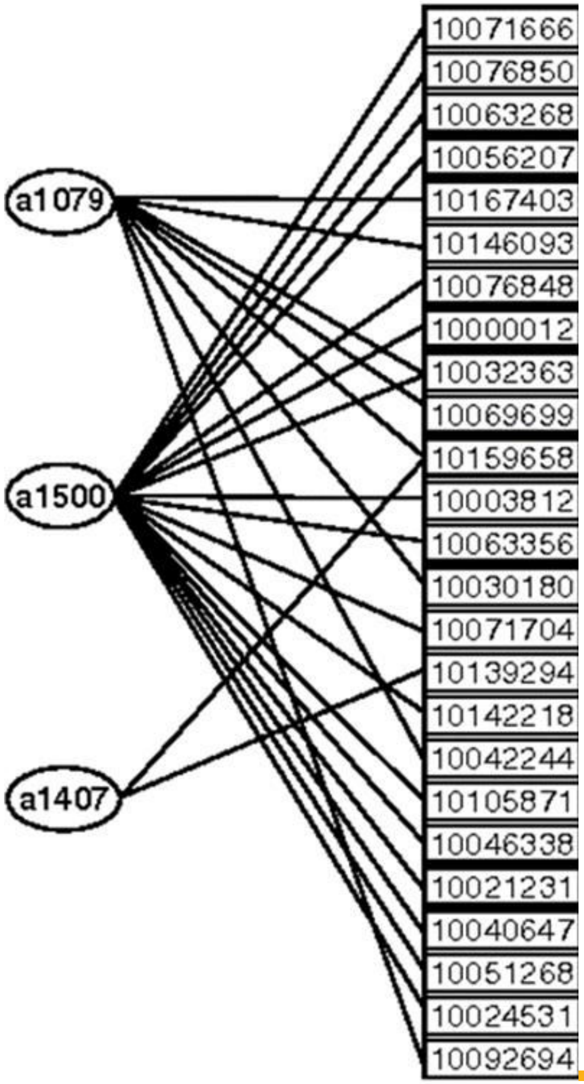


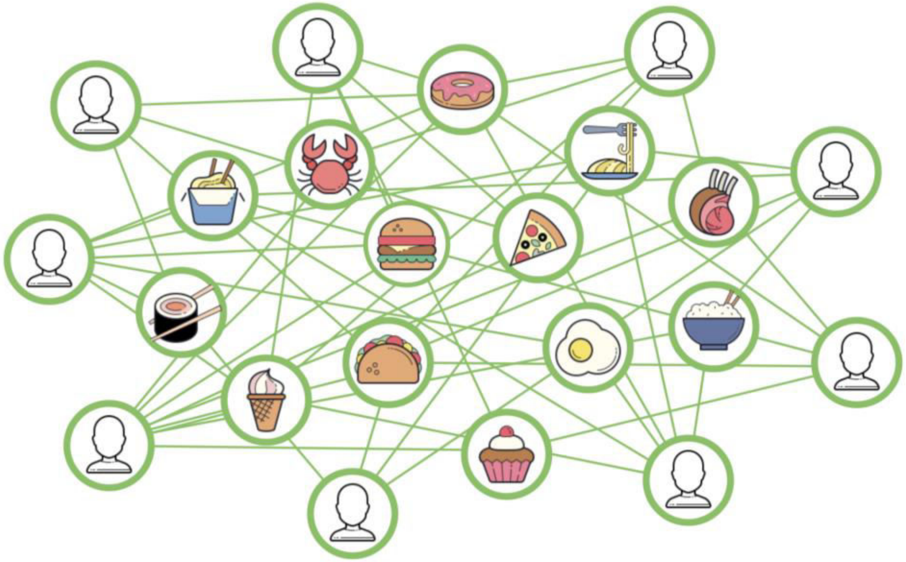


Domain/Application Specific Influence on Design



Domain/Application Specific Influence on Design

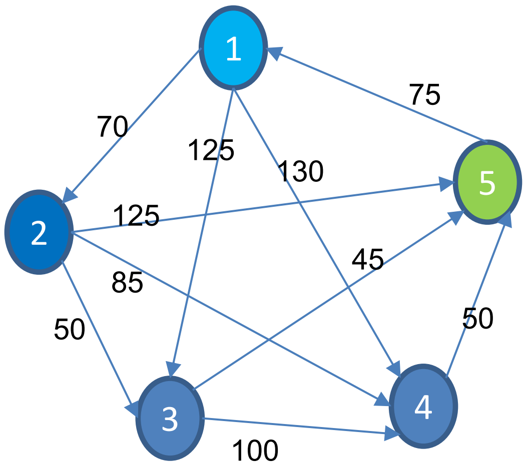




Variations of A*

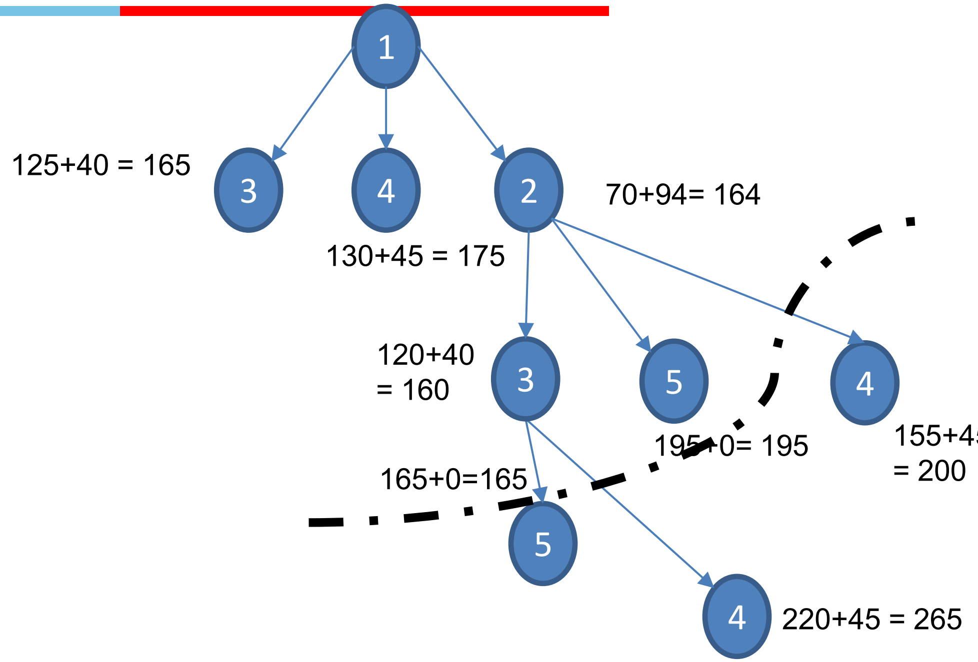
Memory Bounded Heuristics

Iterative Deepening A*



n	h(n)
1	60
2	94
3	40
4	45
5	0

Set limit for f(n)



Cut off value is the smallest of f-cost of any node that exceeds the cutoff on previous iterations

Iterative Limit : Eg

f(n) = 180

f(n) = 195

f(n) = 200

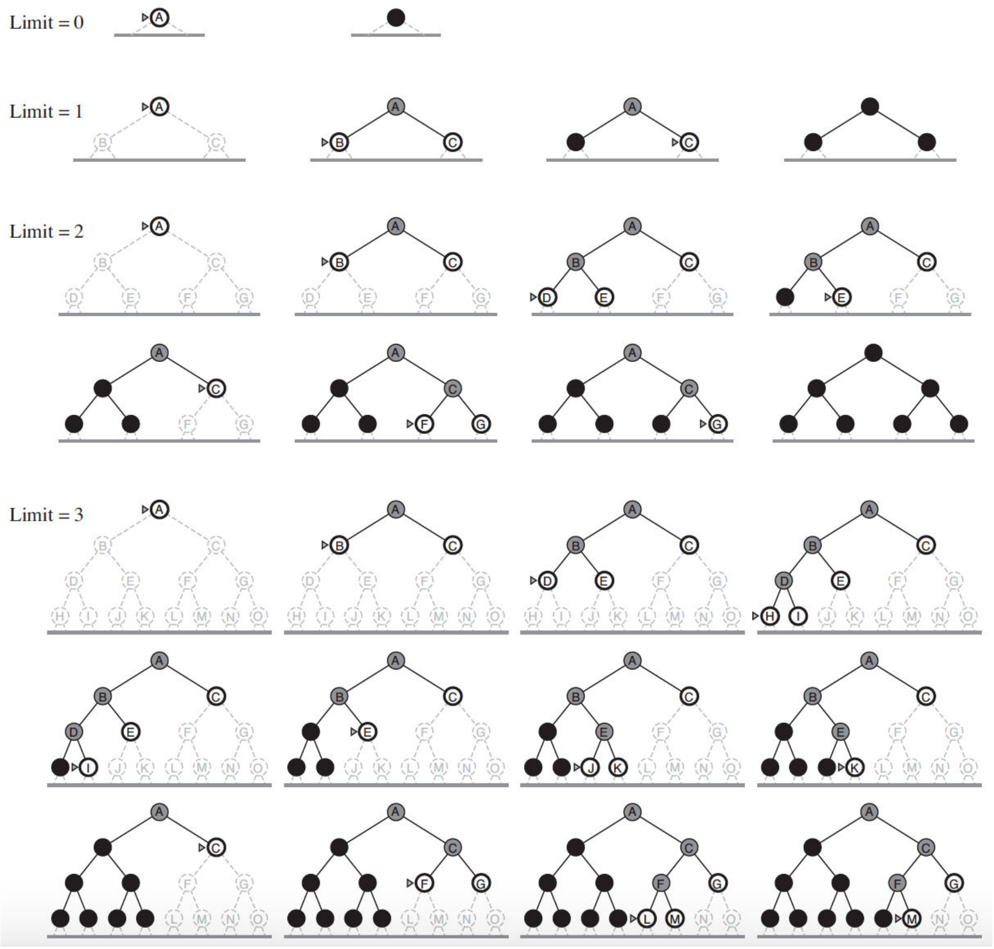
.

.

.

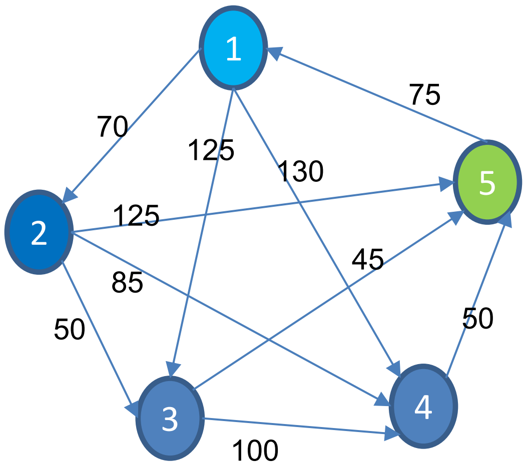
.

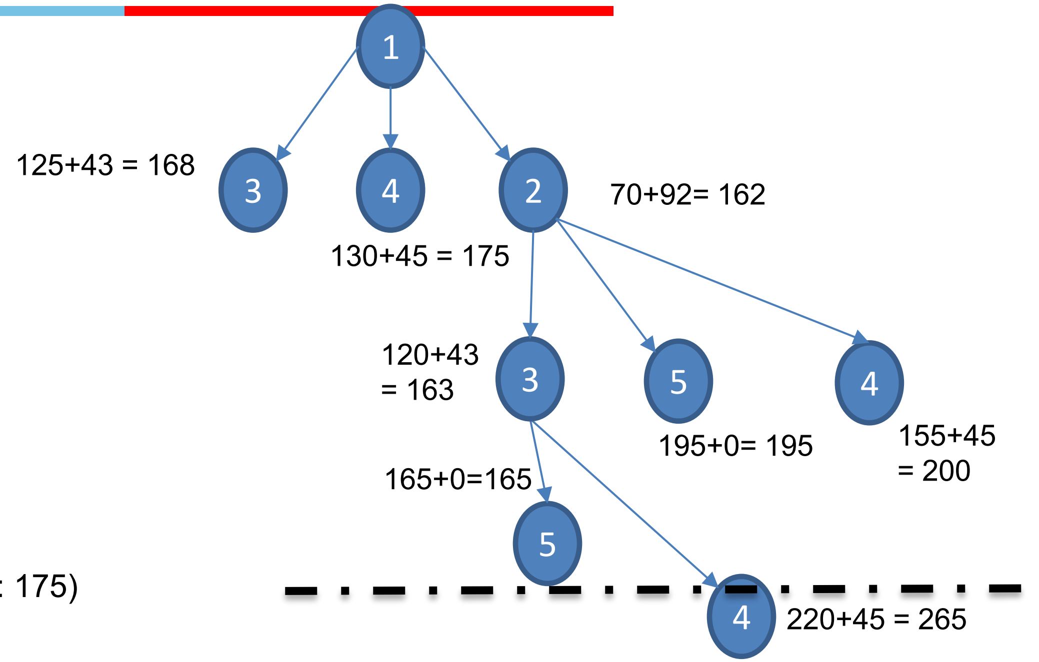
Iterative Deepening Depth First Search (IDS)



Iterative Deepening A*

Set limit for f(n)





n	h(n)
1	60
2	92
3	43
4	45
5	0

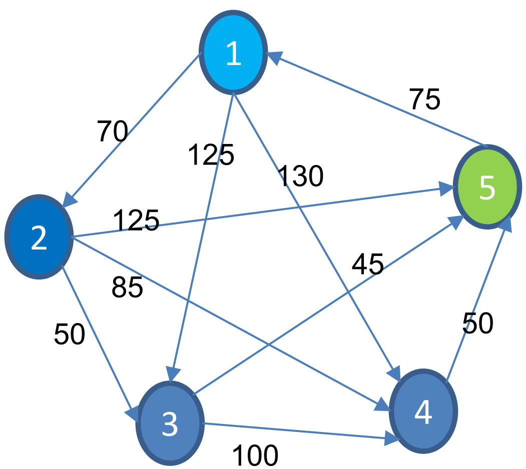
B=60
(1: 60)
TEST-F
(1 2: 162) (1 3: 168) (1 4: 175)

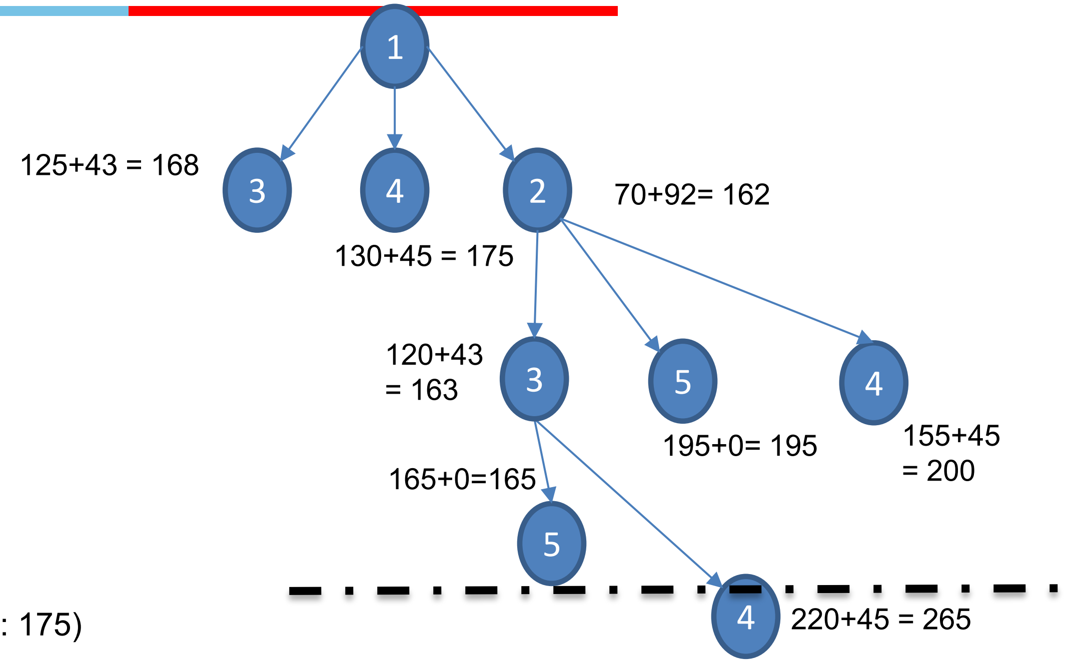
B=162
(1: 60)
TEST-F
(1 2: 162) (1 3: 168) (1 4: 175)
TEST-F
(1 2 3: 163) (1 2 4: 200) (1 2 5: 195) (1 3: 168) (1 4: 175)

B=163
(1: 60)
TEST-F
(1 2: 162) (1 3: 168) (1 4: 175)
TEST-F
(1 2 3: 163) (1 2 4: 200) (1 2 5: 195) (1 3: 168) (1 4: 175)
TEST-F

Iterative Deepening A*

Set limit for f(n)





n	h(n)
1	60
2	92
3	43
4	45
5	0

B=163

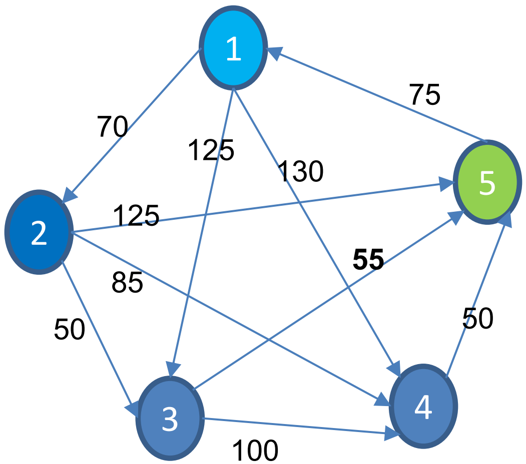
(1: 60)
TEST-F
(1 2: 162) (1 3: 168) (1 4: 175)
TEST-F
(1 2 3: 163) (1 2 4: 200) (1 2 5: 195) (1 3: 168) (1 4: 175)
TEST-F
(1 2 3 5: 165) (1 2 3 4: 265) (1 2 4: 200) (1 2 5: 195) (1 3: 168) (1 4: 175)

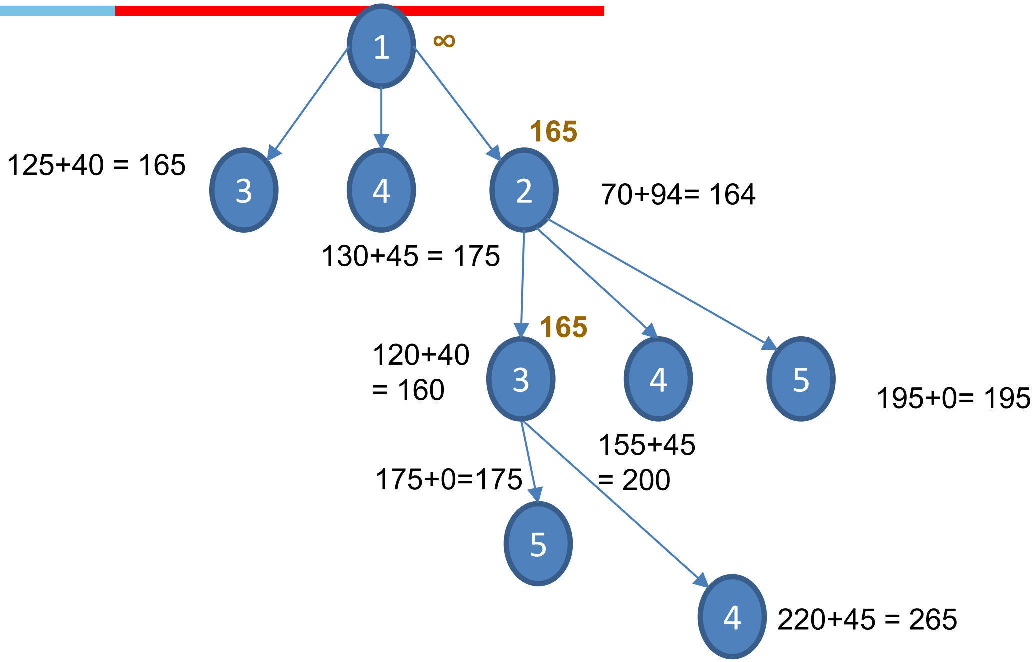
B=165

(1: 60)
TEST-F
(1 2: 162) (1 3: 168) (1 4: 175)
TEST-F
(1 2 3: 163) (1 2 4: 200) (1 2 5: 195) (1 3: 168) (1 4: 175)
TEST-F
(1 2 3 5: 165) (1 2 3 4: 265) (1 2 4: 200) (1 2 5: 195) (1 3: 168) (1 4: 175)

Recursive Best First Search A*

Remember the next best alternative f-Cost to regenerate



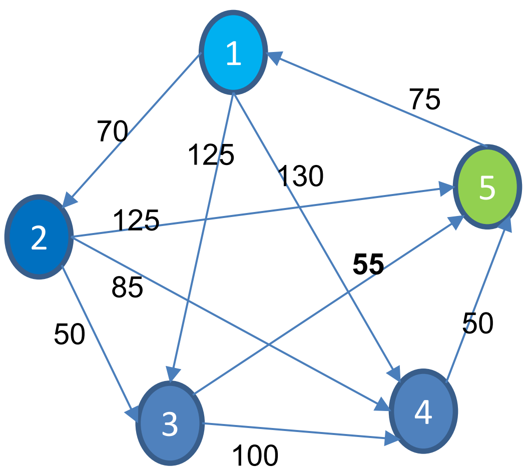


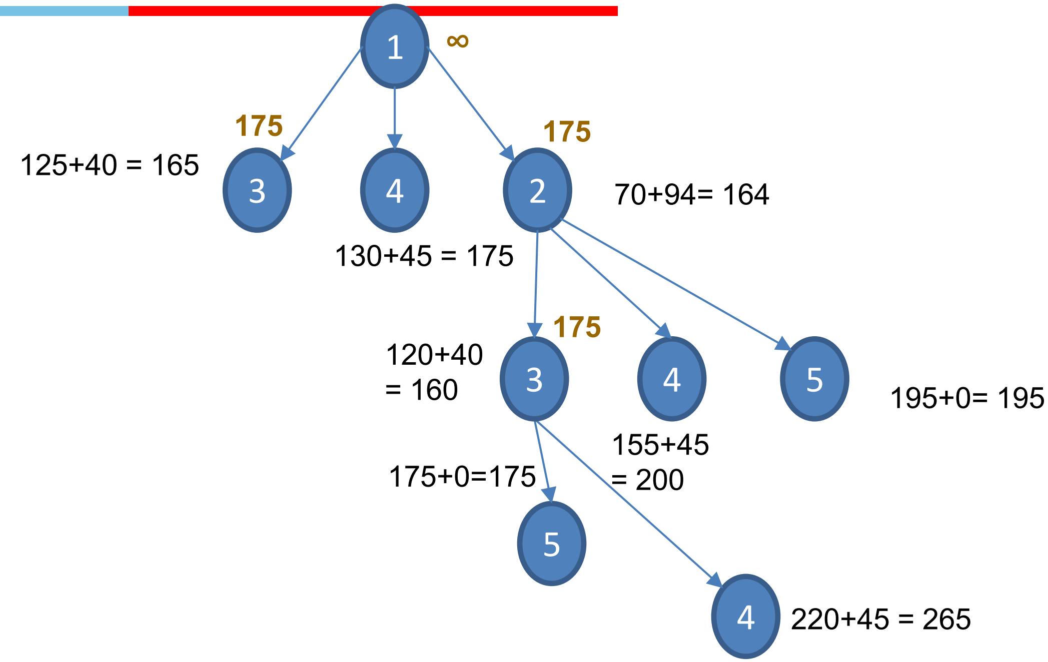
n	h(n)
1	60
2	94
3	40
4	45
5	0

--

Recursive Best First Search A*

Remember the next best alternative f-Cost to regenerate



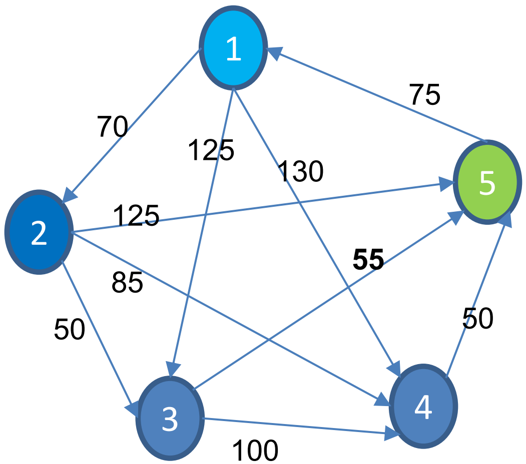


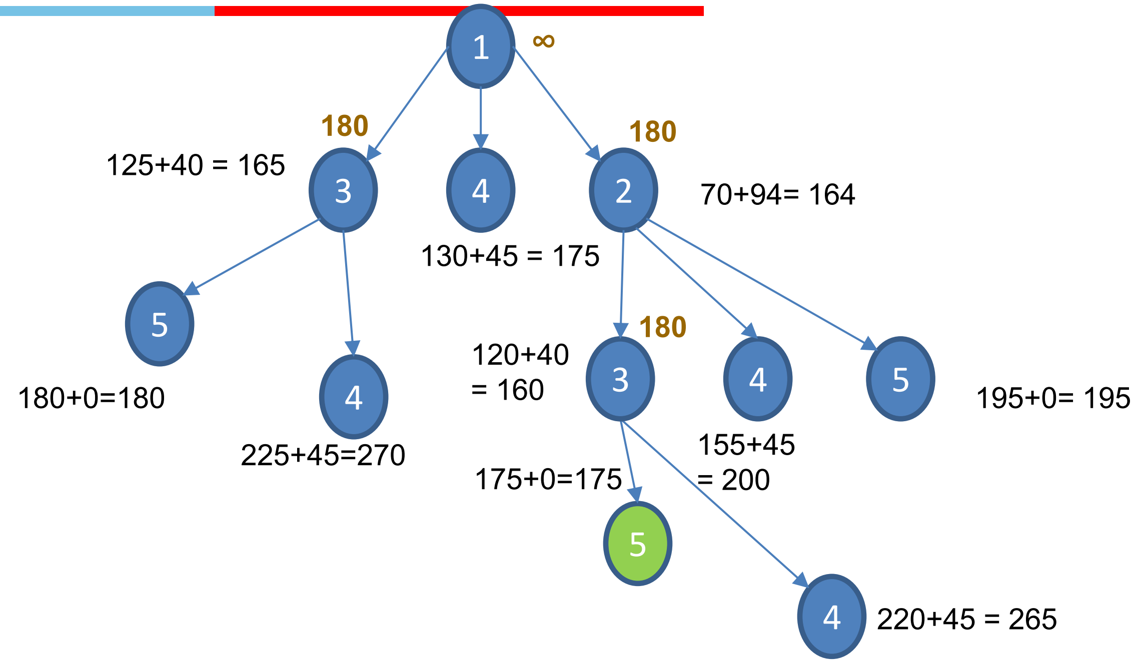
n	h(n)
1	60
2	94
3	40
4	45
5	0

--

Recursive Best First Search A*

Remember the next best alternative f-Cost to regenerate





n	h(n)
1	60
2	94
3	40
4	45
5	0

If the current best leaf value > best alternative path

Best leaf value of the forgotten subtree is backed up to the ancestors

Recursion unwinds

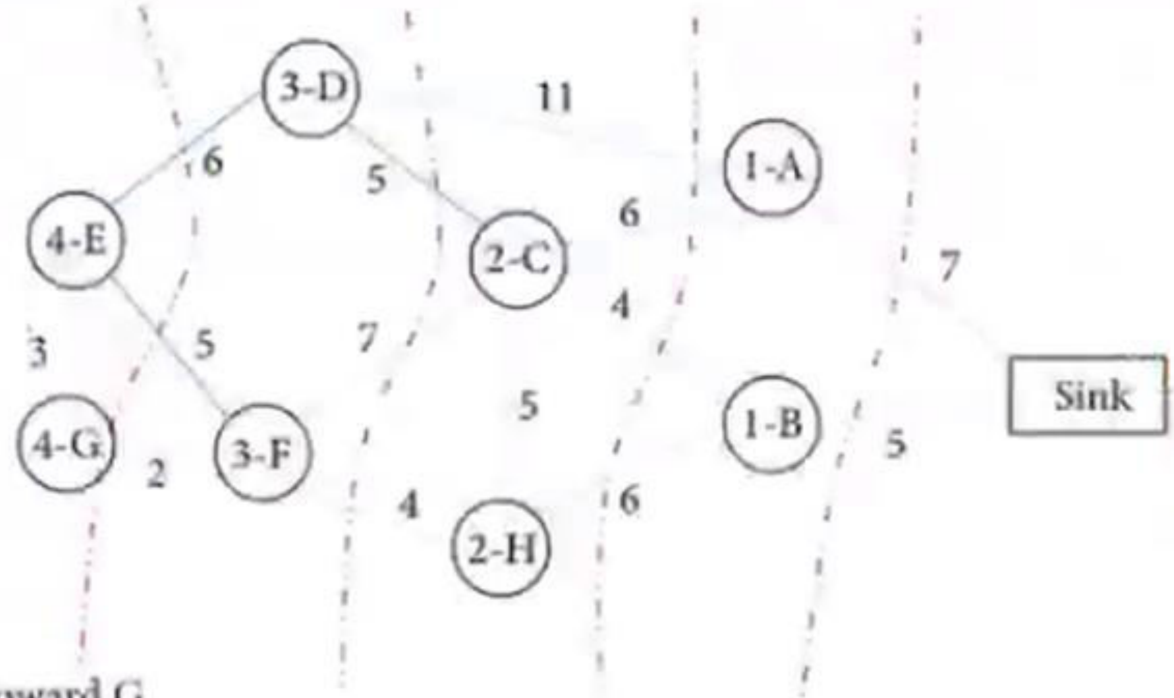
Else

Continue expansion

Space Usage = O(bd) very less

A	14.1
B	11.3
C	8.2
H	6.6
F	2
E	3
D	4.8

Heuristic values toward G



innovate achieve lead

SMA*

Simplified Memory Bounded A*

Simplified Memory Bounded A* (SMA*)

Remember the next best alternative path to backtrack

- Avoids repeated state generation as far as its memory limitation allows
- Remembers the nodes not just the f-cost of next alternative exploration
- Prunes the worst f-cost leaf nodes

If Goal Test (leaf) FAILS or Memory is unavailable

    Drop the shallowest and highest f-cost leaf on node n

If Memory is available

    Expand the deepest lowest f-cost leaf of node **n**

    Update f-cost(n) = **max**(f-cost(n) , f-cost(leaves))

    If the f-limit < f-cost(n)

        f-cost(n) =∞

Simplified Memory Bounded A* (SMA*)

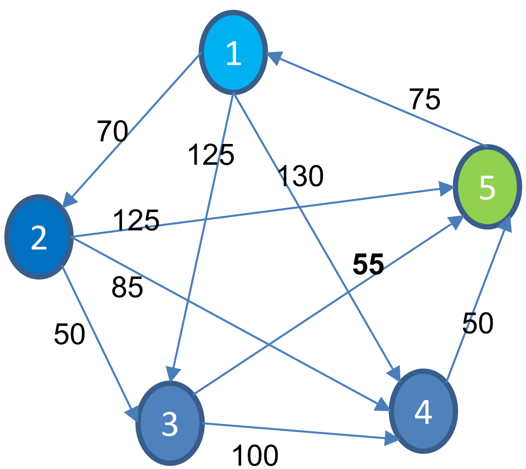
Remember the next best alternative path to backtrack

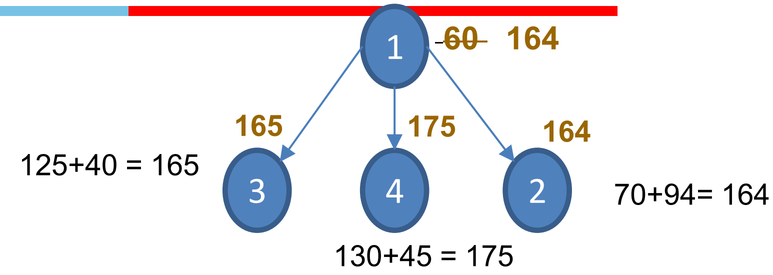
```
function SMA*(problem) returns a solution sequence
  inputs: problem, a problem
  static: Queue, a queue of nodes ordered by f-cost

  Queue ← MAKE-QUEUE({MAKE-NODE(INITIAL-STATE[problem])})
  loop do
     if Queue is empty then return failure
     n ← deepest least-f-cost node in Queue
     if GOAL-TEST(n) then return success
     s ← NEXT-SUCCESSOR(n)
     if s is not a goal and is at maximum depth then
        f(s) ← ∞
     else
        f(s) ← MAX(f(n), g(s)+h(s))
     if all of n's successors have been generated then
        update n's f-cost and those of its ancestors if necessary
     if SUCCESSORS(n) all in memory then remove n from Queue
     if memory is full then
        delete shallowest, highest-f-cost node in Queue
        remove it from its parent's successor list
        insert its parent on Queue if necessary
     insert s on Queue
  end
```

Simplified Memory Bounded A* (SMA*)

Remember the next best alternative path to backtrack





n	h(n)	Successors
1	60	{2,3,4}
2	94	
3	40	
4	45	
5	0	

Assume memory limit = 4 Nodes

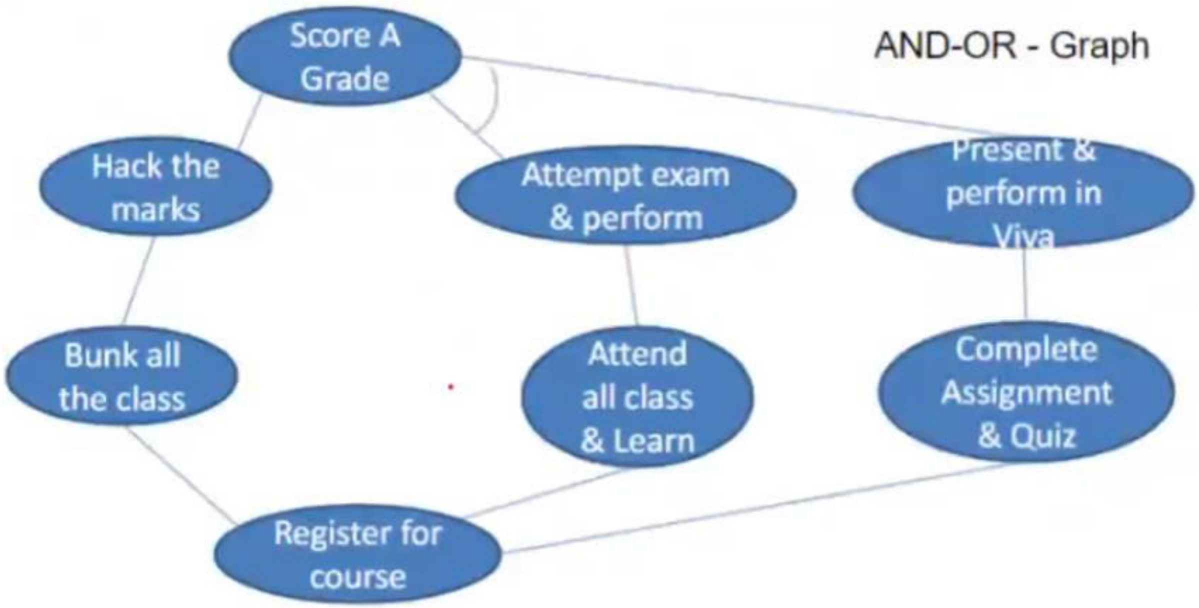
```
function SMA*(problem) returns a solution sequence
  inputs: problem, a problem
  static: Queue, a queue of nodes ordered by f-cost

  Queue ← MAKE-QUEUE({MAKE-NODE(INITIAL-STATE[problem])})
  loop do
     if Queue is empty then return failure
     n ← deepest least-f-cost node in Queue
     if GOAL-TEST(n) then return success
     s ← NEXT-SUCCESSOR(n)
     if s is not a goal and is at maximum depth then
        f(s) ← ∞
     else
        f(s) ← MAX(f(n), g(s)+h(s))
```

Queue-Ordered by f-cost	Path : f-cost | depth
	1 : 60 | 0
	1-2 : 164 | 1
	1-3 : 165 | 1
	1-4 : 175 | 1

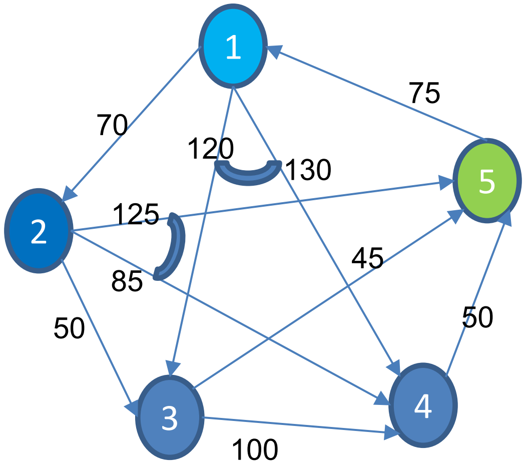
AO* - Idea

AND-OR Graph



AO* - Idea

AND-OR Graph



n	h(n)
1	60
2	92
3	43
4	45
5	0

Module 2 : Problem Solving Agent using Search

A. Uninformed Search

B. Informed Search

C. Heuristic Functions

D. Local Search Algorithms & Optimization Problems

Learning Objective

1. Apply A* variations algorithms to the given problem
2. Compare given heuristics for a problem and analyze which is the best fit
3. Design relaxed problem with appropriate heuristic design
4. Prove the designed relaxed problem heuristic is admissible

Design of Heuristics

Heuristic Design

- **Effective Branching Factor**
- Good Heuristics
- Notion of Relaxed Problems
- Generating Admissible Heuristics

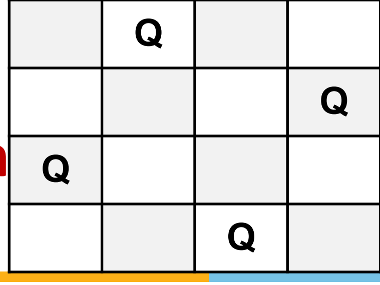
Effective branching factor (b^*):

If the algorithm generates N number of nodes and the solution is found at depth d, then

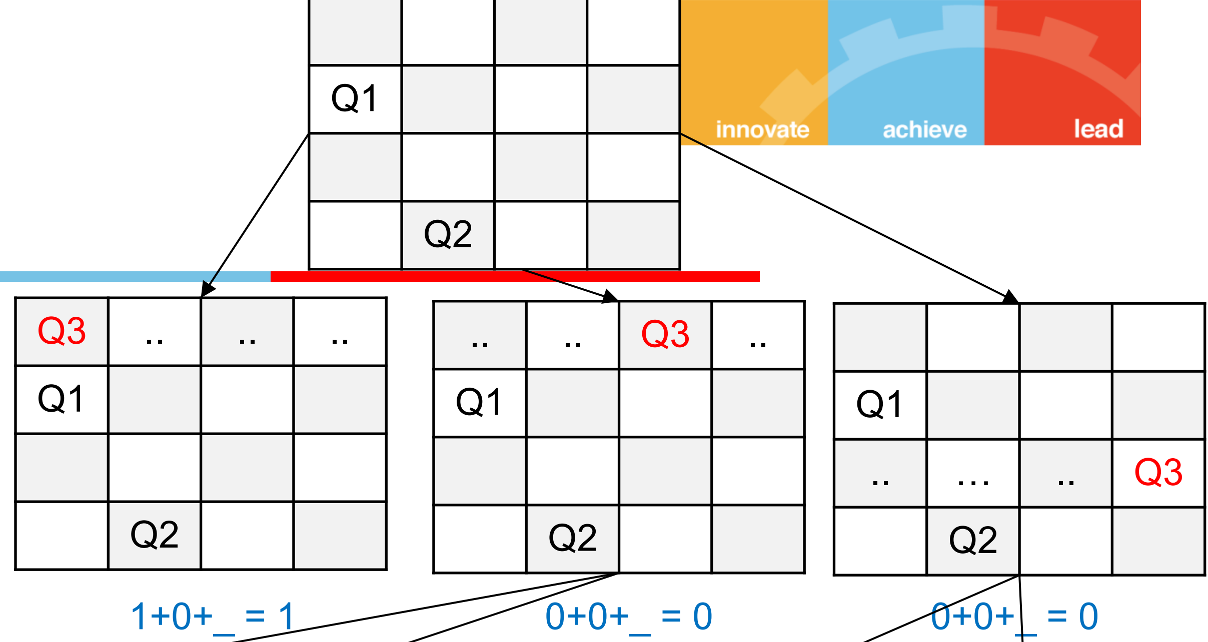
$$N + 1 = 1 + (b^*) + (b^*)^2 + (b^*)^3 + \ldots + (b^*)^d$$

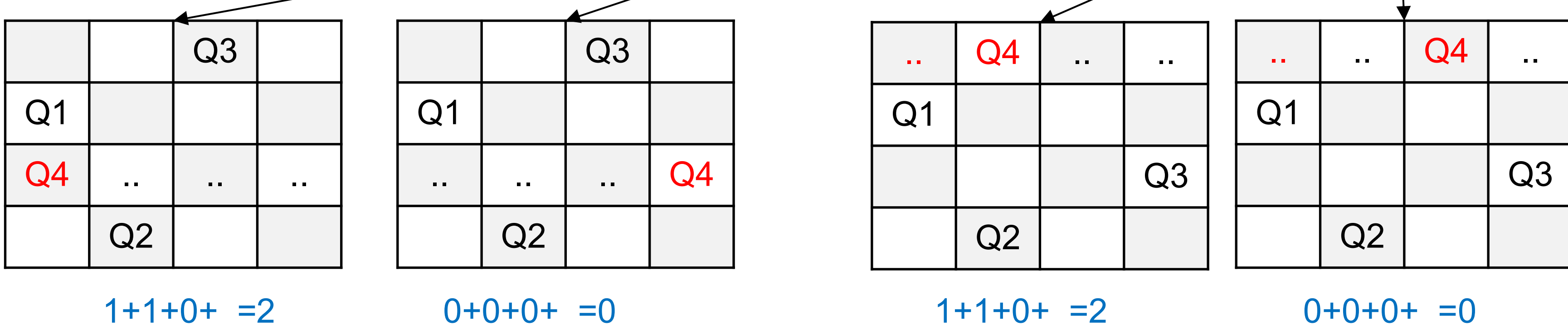
Design of Heuristics

N-Queen



- Construct the search tree by considering one row of the board at a time
- State space graph of relaxed problem is a super graph of original state space because of removal of restrictions

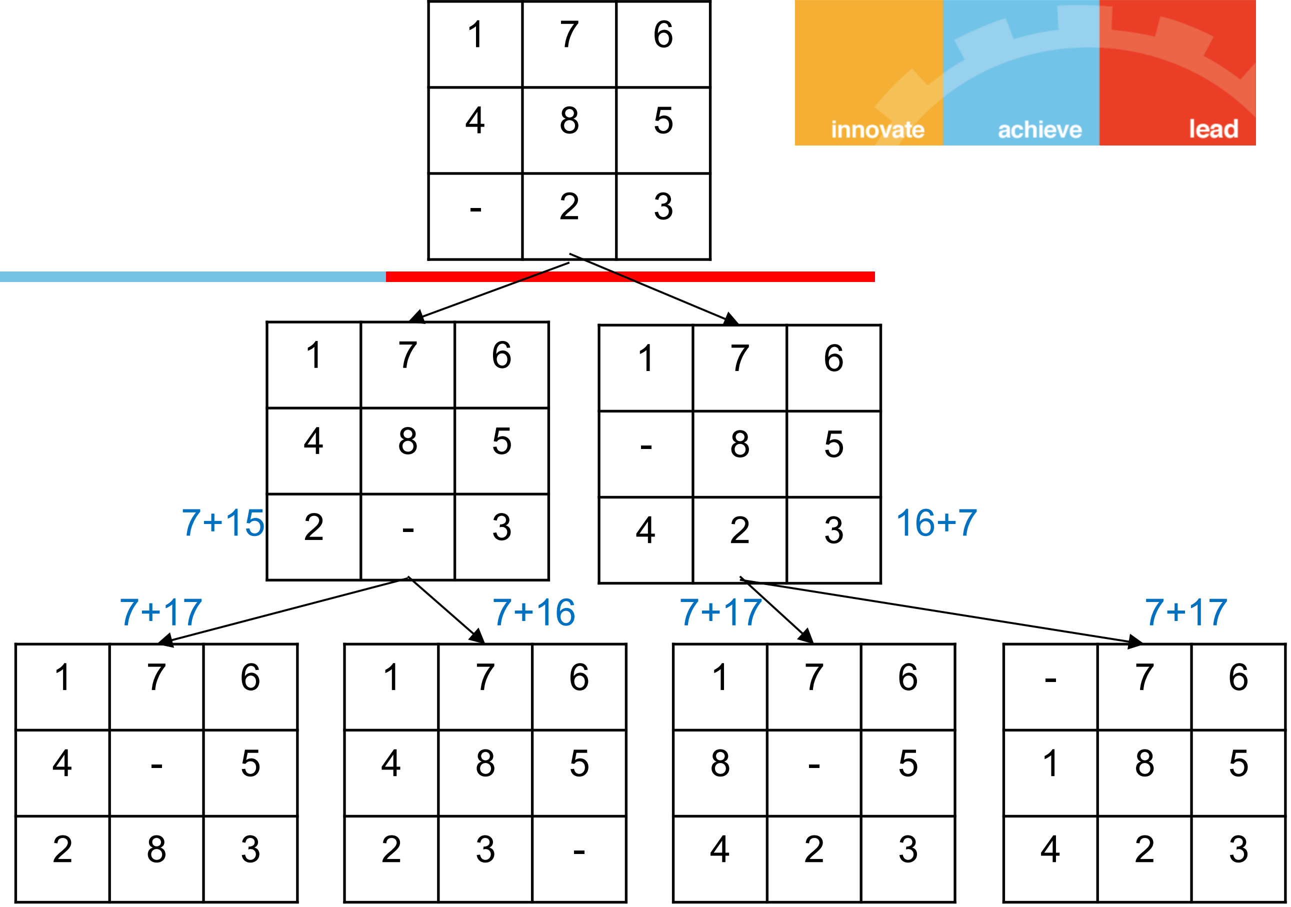




Initial State	Possible Actions	Transition Model	Goal Test	Path Cost	No.Of.States
< Xi , Yi >	Place in any non-occupied row in board		isValid Non-Attacking	Transition + Valid Queens	n!

N-Tile

-	1	2
3	4	5
6	7	8



Initial State	Possible Actions	Transition Model	Goal Test	Path Cost	No.Of.States
<LOC, ID>	Move Empty to near by Tile		LOC=ID+1	Transition + Positional + Distance+ Other approaches	9!

innovate achieve lead

Next Class Plan

- Heuristic Design – Some More Examples
- Comparison of Heuristics
- Local Search Optimization Algorithms

Required Reading: AIMA - Chapter # 3.3, 3.4, 3.5, 4.1, 4.2

Thank You for all your Attention

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