# MODULE - II

# SEARCH STRATEGIES

## Heuristic Search

**20MCA188 ARTIFICIAL INTELLIGENCE (Elective-2)**

**Selma Joseph A.**

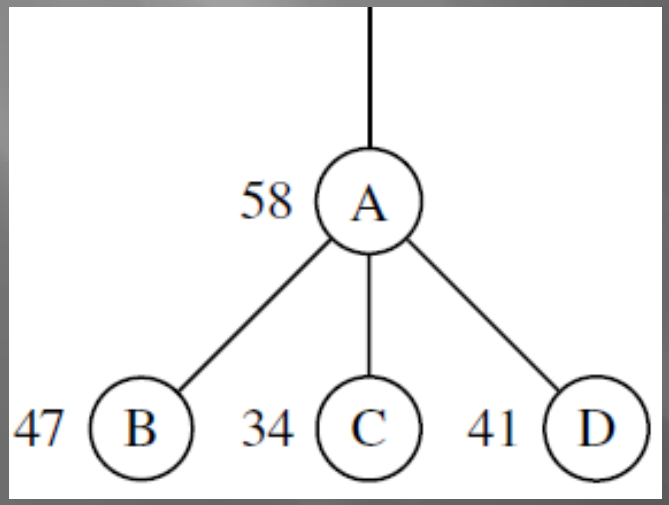
# Heuristic Search

A **heuristic** is a technique designed for solving a problem more quickly when classic methods are too slow or for finding an approximate solution when classic methods fail to find any exact solution. The objective of a heuristic is to produce a solution in a reasonable time that is good enough for solving the problem at hand.

## Heuristic function

A heuristic function, also called simply a heuristic, is a function that ranks alternatives in search algorithms at each branching step based on available information to decide which branch to follow.

**Example:**




Values of a heuristic function

- There are three different branches, say AB, AC and AD, along which we can continue the search for the goal node.
- Decide which one of these nodes is to be selected.
- Obtain an estimate of the cheapest path from the nodes to the goal node.
- Choose to proceed along the branch AC.
- These estimated costs define a heuristic function for the problem.

# Heuristic Search

## Definition

The heuristic function for a search problem is a function h(n) defined as follows:

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

## Remark

Unless otherwise specified, it will be assumed that a heuristic function h(n) has the following properties:

1. **h(n) ≥ 0 for all nodes n.**
2. **If n is the goal node, then h(n) = 0.**

# Admissible heuristic functions

A heuristic function is said to be **admissible** if it never overestimates the cost of reaching the goal, i.e. , the cost it estimates to reach the goal is not higher than the lowest possible cost from the current point in the path.
Let h(n) be a heuristic function for a search problem. h(n) is the estimated cost of reaching the goal from the state n. Let h*(n) be the optimal cost to reach a goal from n.

We say that h(n) is admissible if

$$h(n) \leq h^*(n) \text{ for all } n$$

## Remarks

1. There are no limitations on the function h(n). Any function of our choice is acceptable.
As an extreme case, the function h(n) defined by

**h(n) = c for all n**

where c is some constant can also be taken as a heuristic function.

The heuristic function defined by **h(n) = 0 for all n** is called the trivial heuristic function.

# Admissible heuristic functions

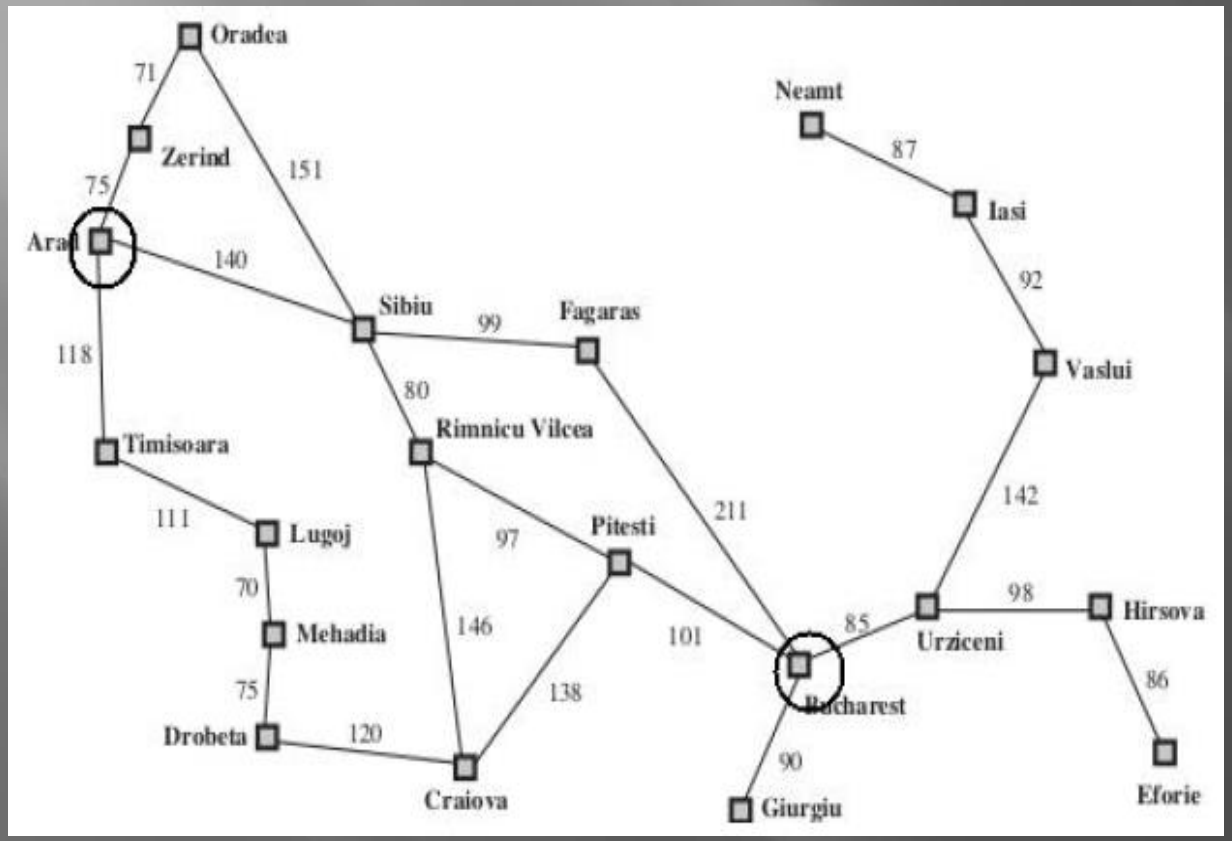
2. The heuristic function for a search problem would make use of information that are specific to the circumstances of the problem. Because of this, there is no one method for constructing heuristic functions that are applicable to all problems.

3. However, there is a standard way to construct a heuristic function: It is to find a solution to a simpler problem, which is one with fewer constraints. A problem with fewer constraints is often easier to solve. An optimal solution to the simpler problem cannot have a higher cost than an optimal solution to the full problem because any solution to the full problem is a solution to the simpler problem.

# Examples

**1. h(n) in path search in maps**

Consider the problem of finding the shortest path from Arid to Bucharest in Romania. In the problem, the traveler is required to travel by roads shown in the map. Let us consider a problem obtained by relaxing this condition.




Let the traveler be allowed to travel by any method. In this relaxed problem, the shortest path from from a city n to Bucharest is the air distance between n and Bucharest. We may use this solution as the definition of a heuristic function for the original problem.

# Examples

If n is any of the cities in Romania, we may define the heuristic function h(n) as follows:
**h(n) = The straight line distance (air distance) from n to Bucharest**.

Table : 1 shows the values of h(n).

Note that h(n)≥0 for all n and h(Bucharest)= 0

Now h*(n) is the length of the shortest road-path from n to Bucharest. Since, the air distance never exceeds the road distance we have

$$h(n) \leq h^*(n) \quad \text{for all } n.$$

Thus, h(n) as defined above is an admissible heuristic function for the problem

| City ($n$) | $h(n)$ |
|---|---|
| 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 | 100 |
| Rimnicu Vilcea | 193 |
| Sibiu | 253 |
| Timisoara | 329 |
| Urziceni | 80 |
| Vaslui | 199 |
| Zerind | 374 |

Table : 1

# Examples

## 2. h(n) in the 8-puzzle problem

There are several restrictions, though not explicitly stated, on the movements of the tiles in the 8-puzzle problem like the following :

1. A tile can be moved only to a nearby vacant slot.
2. Tiles can only be moved in the horizontal or vertical direction.
3. One tile cannot slide over on another tile.

We can get different admissible heuristic functions for the 8-puzzle problem by solving simpler problems by relaxing or removing some of these restrictions. We define two different heuristic functions h1(n) and h2(n) for the 8-puzzle problem.

h1(n) is obtained by solving the problem without all the three restrictions given above. h2(n) is obtained by solving the problem obtained by relaxing first restriction to the restriction that “a tile may be moved to any nearby slot and deleting the third restriction”.

**h1(n) = Number of tiles out of position.**

# Examples

## 2. h(n) in the 8-puzzle problem

Eg. 1.

| | | |
|---|---|---|
| 5 | | 8 |
| 4 | 2 | 1 |
| 7 | 3 | 6 |

| | | |
|---|---|---|
| 1 | 2 | 3 |
| 4 | 5 | 6 |
| 7 | 8 | |

(a) Given state ($n$) (b) Goal state

Consider the state shown in Eg. 1 and compare the positions of the tiles with their positions in the goal state. It can be seen that only two tiles, namely 4 and 7 (ignoring the empty cell), are in their positions. Hence, for the state we have **h1(n) = 6:**

- **Manhattan distance heuristic**

We first consider the Manhattan distance between two cells in a grid of cells. The Manhattan distance heuristic function is defined as follows:

**h2(n) = Sum of the Manhattan distances of every numbered tile to its goal position**.

Eg: For the initial state n shown in Fig 1, we have

h2(n) =Manhattan distance of 1 from goal position+

Manhattan distance of 2 from goal position+

+……

Manhattan distance of 8 from goal position

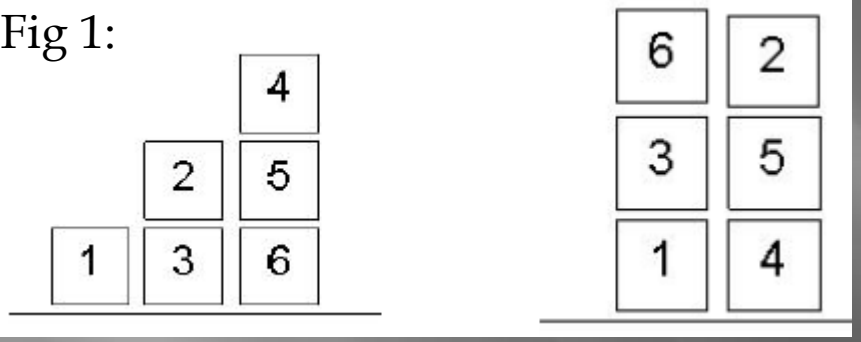
=3 + 1 + 3 + 0 + 2 + 1 + 0 + 3 = 13:

# Examples

**3.** h(n) in blocks world problem

We define a heuristic function for the blocks world problem

Fig 1:




(a) Given state ($n$) (b) Goal state

In the blocks world problem with the initial and goal states as in Fig 1. In the initial state, only block numbered "1" is resting on the thing it is supposed to be resting on. All other blocks are sitting on the wrong things. Hence we have:

h(Initial state) = +1 - 1 - 1 - 1 - 1 - 1 = -4:

Since in the goal state every block is sitting on the right thing, we have

h(Goal state) = 1 + 1 + 1 + 1 + 1 + 1

= 6 ≠ 0

h(n) = Add one point for every block that is resting on the thing it is supposed to be resting on. Subtract one point for every block that is sitting on the wrong thing.

This is an example of a heuristic function not satisfying the conditions h(n) ≥ 0 and h(Goal node) = 0.

# Examples

## 4. h(n) in water jug problem

For the water jug problem, we may choose the following as a heuristic function. For a state specified by the ordered pair (x , y), we define

$$h(x, y) = |x - 2| + |y - 2|$$

This function satisfies the condition h(n) ≥ 0 for all n. It does not satisfy the condition h(Goal state) = 0, because as a goal state is specified by a pair of the form (2 , y), we have

h(Goal state) = h(2 , y) = |y – 2|

## 5. h(n) in missionaries and cannibals problem.

To define a heuristic function for the missionaries and cannibals problem, let us consider a simpler problem where in we ignore the condition that at any location (on either bank of the river or in the boat) the number of cannibals should not exceed the number of missionaries.
Let k be the number of persons at the initial side of the river. If k = 2 we need only 1 river crossing to take both persons to the other side of the river.

**k = 3** ⇒ 3
**k = 4** ⇒ 5 ?
**k = 5** ⇒ 7
**K= 6** ⇒ 9

# Greedy best first search

## The algorithm

The greedy best first search algorithm uses the following notations and conventions:

OPEN : A list which keeps track of the current "immediate" nodes available for traversal.

CLOSED : A list that keeps track of the nodes already traversed.

h(n) : The heuristic function used as the evaluation function f(n), that is, f(n) = h(n).

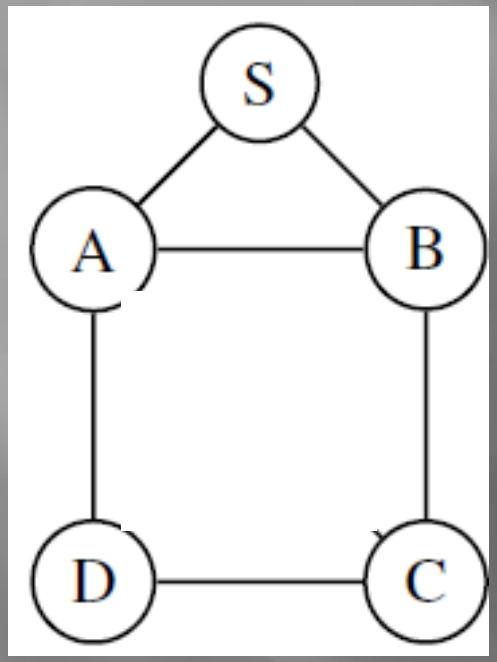
## Algorithm

1. Create two empty lists: OPEN and CLOSED.
2. Start from the initial node (say N) and put it in the ordered OPEN list.
3. Repeat the next steps until goal node is reached.
   (a) If OPEN list is empty, then exit the loop returning FALSE.
   (b) Select the first/top node (say N) in the OPEN list and move it to the CLOSED list. Also record the information of the parent node of N.
   (c) If N is a goal node, then move the node to the CLOSED list and exit the loop returning TRUE. The solution can be found by backtracking the path.
   (d) If N is not the goal node, generate the immediate successors of N, that is, the immediate next nodes linked to N and add all those to the OPEN list.
   (e) Reorder the nodes in the OPEN list in ascending order according to the values of the evaluation function f(n) at the nodes.

# Greedy best first search

## Example 1

Use the greedy best first search algorithm to find a path from S to C in the graph




Use the heuristic function:

| Node $n$ | S | A | B | C | D |
|---|---|---|---|---|---|
| $h(n)$ | 6 | 2 | 3 | 0 | 2 |

From the list of nodes in the CLOSED list, the goal state has been reached , the path from S to C is **S →A →D →C**

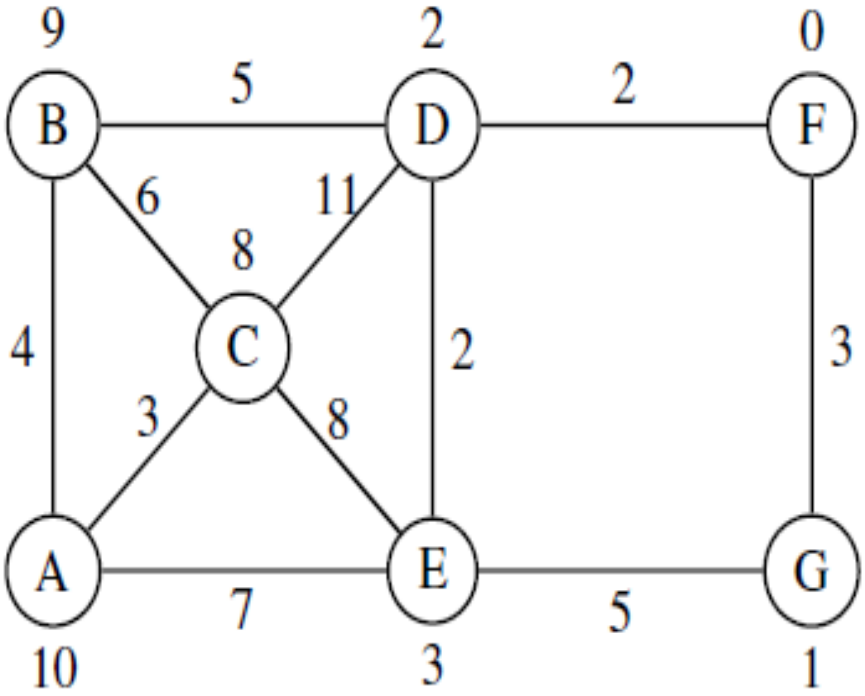
| Current (N) | Children of N | OPEN | CLOSED (node, parent) | Remarks |
|---|---|---|---|---|
| S | – | – | – | Initial state |
| – | – | S | – | Initial state added to OPEN. |
| S | – | – | (S, –) | S made current node and added to CLOSED. |
| S | A, B | – | (S, –) | Children of S |
| S | – | A, B | (S, –) | Add children to OPEN and order according to values of heuristic function. |
| A | – | B | (S, –), (A, S) | A made current node and added to CLOSED. |
| A | B, D | B | (S, –) (A, S) | Children of A |
| A | – | D, B | (S, –) (A, S) | Add children to OPEN and order according to values of heuristic function. |
| D | – | B | (S, –), (A, S), (D, A) | D made current node and added to CLOSED. |
| D | A, C | B | (S, –), (A, S), (D, A) | Children of D |
| D | – | C, B | (S, –), (A, S), (D, A) | Add children to OPEN and order according to values of heuristic function. |
| C | – | B | (S, –), (A, S), (D, A), (C, D) | C made current node and added to CLOSED. |
| C | – | – | – | C is goal state. |

"C ← D ← A ← S",

HOMEWORK

# EXAMPLE 2

Using the greedy best first search algorithm, find an optimal path from A to F in the search graph shown in Figure . In the figure, the numbers written alongside the nodes are the values of the heuristic function and the numbers written alongside the edges are the costs associated with the edges.



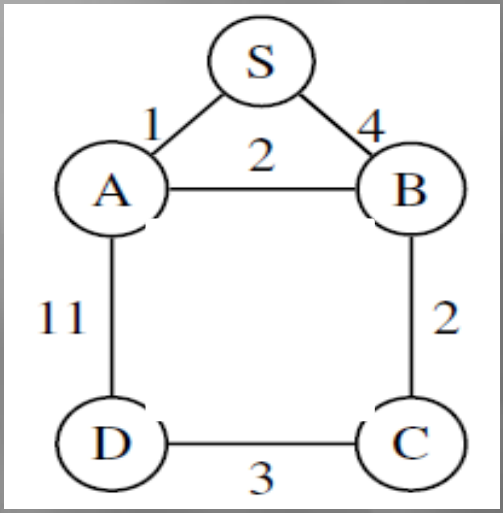
# A* best first search

Let h(n) be a heuristic function associated with the search problem and g(n) be the cost of traversing the path from the start node to the node n. In the A* best first search, we use the following evaluation function:

$$f(n) = g(n) + h(n)$$

## Example 1

Use the A best first search algorithm to find a path from A to C in the graph.




Use the heuristic function:

| Node $n$ | S | A | B | C | D |
|---|---|---|---|---|---|
| $h(n)$ | 6 | 2 | 3 | 0 | 2 |

**Solution**

**Step 1**. We start with S.

Nodes A and B are immediate successors of S. We have

f(A) = g(A) + h(A) = 1 + 2 = 3

f(B) = g(B) + h(B) = 4 + 3 = 7

Min {f(A), f(B)}= min{3,7} = 3 = f(A)

So we move to A and the path is **S →A.**

**Step 2.** Nodes B and D can be reached from A. We have

f(B) = g(B) + h(B) = (1 + 2) + 3 = 6

f(D) = g(D) + h(D) = (1 + 11) + 2 = 14

Min { f(B) , f(D) } = min { 6 , 14 } = 6 = f(B)

So we move to B and the path so far is **S→A → B.**

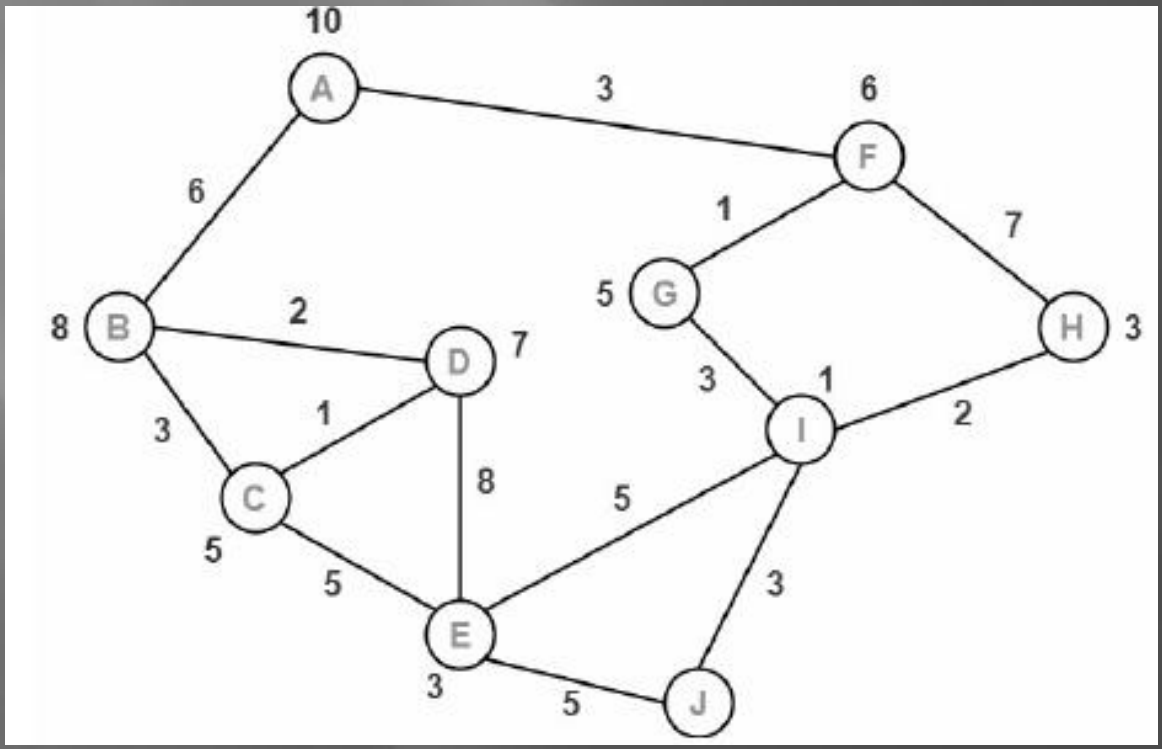
**Step 3.** C (Goal state) can be reached from B.

Hence the required path is **S→A →B→ C.**

# Example 2

HOMEWORK

Consider the graph shown in Figure. The numbers written on edges represent the distance between the nodes. The numbers written on nodes represent the heuristic value. Find the most cost-effective path to reach from start state A to final state J using A* Algorithm.



# Greedy best first vs A* best first

| Sl. No. | Greedy best first | $A^*$ best first |
|---|---|---|
| 1 | The greedy best first is not complete, that is, the algorithm may not find the goal always. | If the heuristic function is admissible, the $A^*$ best first is complete. |
| 2 | In general, the greedy best first may not find the optimal path from the initial to the goal state. | If the heuristic function is admissible, the $A^*$ best first is always yields the optimal path from the initial state to goal state. |
| 3 | In general, greedy best first uses less memory than $A^*$ best first. | In general, A* best first uses more memory than greedy best first. |