Lecture three

Topics that must be covered in this lecture:

- Heuristic search (Hill-climbing).
- Methods of Heuristic search (Best-First-Search).

Informed Search (Heuristic Search):

Heuristic Search: A heuristic is a method that might not always find the best solution but is guaranteed to find a good solution in a reasonable time. Heuristic search is useful in solving problems that could not be solved any other way and where solutions take an infinite time or a very long time to compute. Heuristic is the information about the likelihood that a specific node is a better choice to try the next rater than another one. Most often heuristic search methods are based on maximizing or minimizing some aspect of the problem. The heuristic function (evaluation function) is a function that evaluates individual problem states and determines how desirable they are.

Fields of using the Heuristic in AI:

• Could not be solved any other way.

• Solution takes an infinite time or a very long time to compute.

Heuristic search methods generate and test algorithms, from these methods: -

1- Hill Climbing.

2- Best-first search (generic best-first, Dijkstra's algorithm, A^*).

Heuristic Search compared with another search:

The Heuristic search is compared with the Brute force or Blind search Techniques are as follows:

Uniformed Search	Informed Search
No information about the number of steps or path cost from the current state to the goal state	The path cost from the current state is calculated, to select the minimum path cost as the next state
More effective	Less effective in the search method
The problem to be solved with the given information	Additional information can be added as an assumption to solve the problem
Depth First Search, Breadth First Search	Hill Climbing, Best First Search, A* Search

1) Hill Climbing

The idea here is that you don't keep a big list of states around. You just keep track of the one state you are considering, and the path that got you there from the initial state. At every state you choose the state leads you closer to the goal (according to the heuristic estimate), and continues from there.

The name "Hill Climbing" comes from the idea that you are trying to find the top of a hill, and you go in the direction that is up from wherever you are. This technique often works, but since it only uses local information.

The algorithm for Hill Climbing is as follows:

1. Evaluate the initial state, if it is the goal state quit otherwise make the current state as the initial state.

2. Select a new operator that could be applied to this state and generate a new state.

3. Evaluate the new state If this new state is closer to the goal state than the current state make the new state the current state If it is not better ignore this state and proceed with the current state if the current state is the goal state or no new operators are available, quit. Otherwise, repeat from 2.

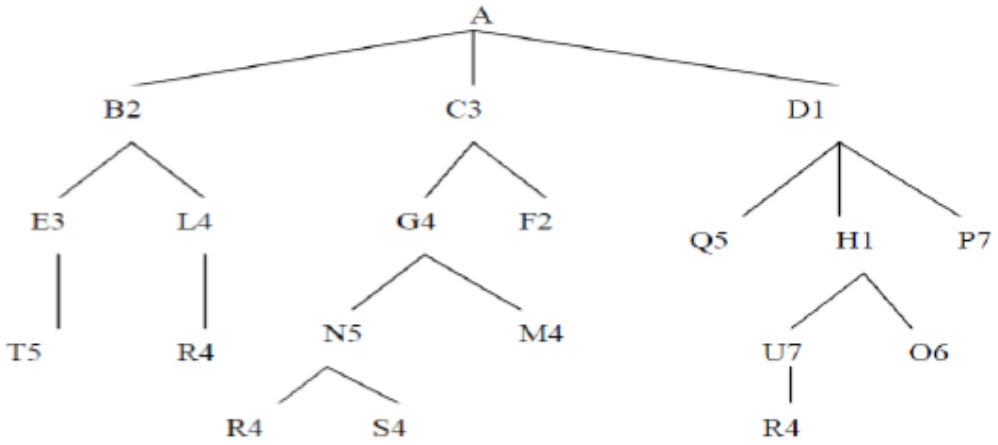
Hill Climbing Algorithm

```
Begin
Cs=start state;
Open=[start];
Stop=false;
Path=[start];
While (not stop) do
{
if (cs=goal) then
return (path);
generate all children of cs and put it into open
if (open=[]) then
stop=true
else
{
x:= cs;
for each state in open do
{
```

```
compute the heuristic value of y (h(y));
if y is better than x then
x=y
}
if x is better than cs then
cs=x
else
stop =true;
}
}
return failure;
}
```

For Example:

Searches for R4



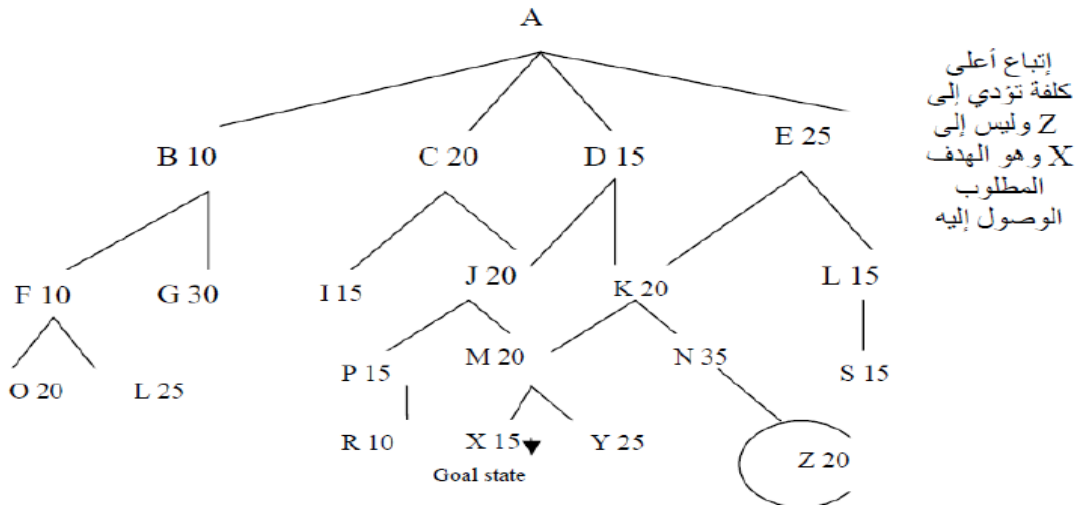
Open	Close	X
A	_	A
C3 B2 D1	A	C3
G4 F2	C3 A	G4
N5 M4	G4 C3 A	N5
R4 S4	N5 G4 C3 A	R4

Optimal path (A →C3 →G4 →N5 →R4)

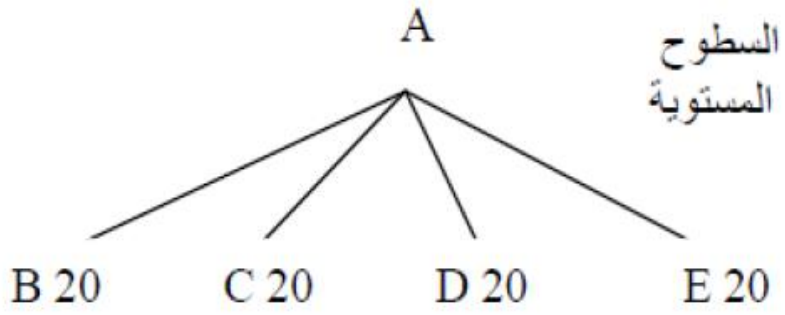
Hill climbing Problems:

Hill climbing may fail due to one or more of the following reasons: -

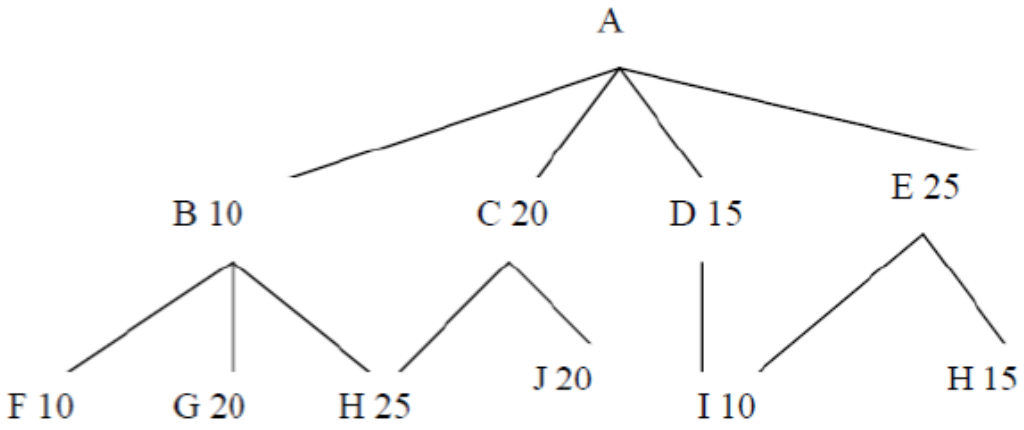
1- **A local maxima**: Is a state that is better than all of its neighbors but is not better than some other states.



2- **A Plateau:** Is a flat area of the search space in which a number of states have the same best value, on plateau it's not possible to determine the best direction in which to move.



3- **A ridge:** Is an area of the search space that is higher than surrounding areas, but that cannot be traversed by a single move in any one direction.



سلسلة الانخفاضات و الارتفاعات

These Problems could be solved using methods:

1. Backtracking to some earlier node and try going in a different direction.
2. Making big jumps in some direction to try to get new section of the search.
3. Apply two or more rules before doing the test.

Methods of Heuristic Search:

2-Best-First-Search

Best-First-search is a way of combining the advantages of both depth-first and breadth-first search into a single method.

The actual operation of the algorithm is very simple. It proceeds in steps, expanding one node at each step until it generates a node that corresponds to a goal state. At each step, it picks the most promising of the nodes that have so far been generated but not expanded. It generates the successors of the chosen node, applies the heuristic function to them, and adds them

to the list of open nodes, after checking to see if any of them have been generated before. By doing this check, we can guarantee that each node only appears once in the graph, although many nodes may point to it as a successor. Then the next step begins.

In the Best-First search, the search space is evaluated according to a heuristic function. Nodes yet to be evaluated are kept on an OPEN list and those that have already been evaluated are stored on a CLOSED list. The OPEN list is represented as a priority queue, such that unvisited nodes can be queued in order of their evaluation function. The evaluation function f(n) is made from only the heuristic function (h(n)) as f (n) = h(n).

Best-First Search Method:

1. Use two ordered lists open and close.
2. Start with the initial node n0 and put it on the ordered list open.
3. Create a list close. This is initially an empty list.
4. If open is empty exit with failure.
5. Select the first node on open. Remove it from the open and put it on close. Call this node n. 6. If n is the goal node exit. The solution is obtained by tracing a path backward along the arcs in the tree from n0 to n.
7. Expand node n. This will generate success.

Let the set of successors generated be m. create arcs from n to each member of m.

8. Reorder the list open according to the heuristic and go back to step 4.

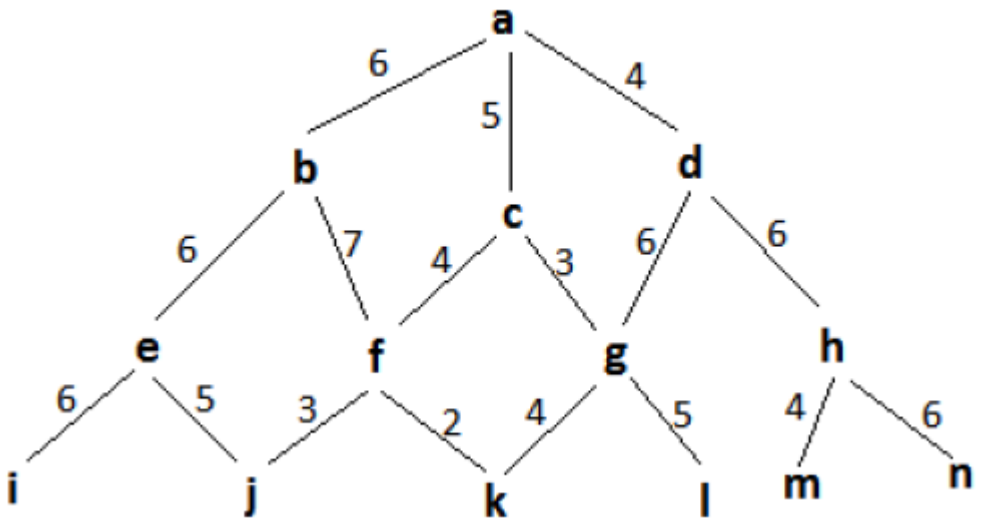
Best-First-Search Algorithm

```
{
Open:=[start];
Closed:=[];
While open ≠[] do
{
Remove the leftmost from open, call it x;
If x= goal then
Return the path from start to x
Else
{
Generate children of x;
For each child of x do
Do case
The child is not already on open or closed;
{ assign a heuristic value to the child state ;
Add the child state to open;
}
The child is already on open:
If the child was reached along a shorter path than the state currently on open then
give the state on open this shorter path value.
The child is already on closed:
If the child was reached along a shorter path than the state currently on open then
{
Give the state on closed this shorter path value
Move this state from closed to open
}
}
Put x on closed;
```

Re-order state on open according to heuristic (best value first)
}
Return (failure);
}

For Example

Searches for l



Open	closed
[a]	[ ]
[b6,c5,d4]	[a]
[d4,c5,b6] ‘ sort ‘	[a]
[g6,h6,c5,b6]	[d4,a]
[c5,g6,h6,b6] ‘ sort ‘	[d4,a]
[f4,g3,g6,h6,b6] ‘delete g6’	[c5,d4,a]
[g3,f4,h6,b6] ‘ sort ‘	[c5,d4,a]
[k4,l5, f4,h6,b6]	[g3, c5,d4,a]
[k4,f4,l5,h6,b6] ‘ sort ‘	[g3, c5,d4,a]
[f4, l5,h6,b6]	[k4, g3, c5,d4,a]
[j3,k2,l5,h6,b6]	[f4, k4, g3, c5,d4,a]
[k2,J3, l5,h6,b6] ‘ sort ‘	[f4, k4, g3, c5,d4,a]
[k2,J3, l5,h6,b6]	[f4, g3, c5,d4,a] ‘ delete k4’

[J3, l5,h6,b6]	[k2, f4, g3, c5,d4,a]
[l5,h6,b6]	[j3, k2, f4, g3, c5,d4,a]

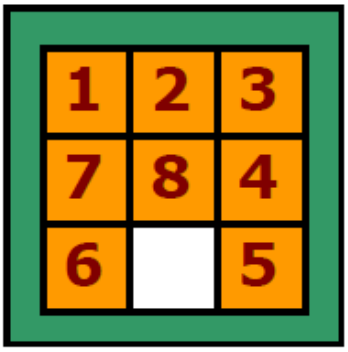
Optimal path (a → d4 → c5 → g3 → f4 → k2 → j3 → l5)

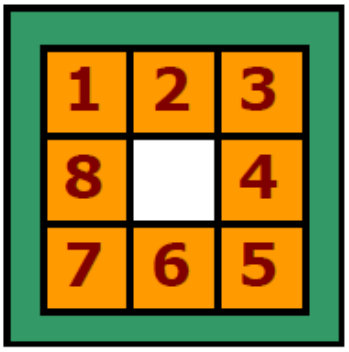
Example : 8 – Puzzle:

◊ State space: Configuration of 8- tiles on the board

◊ state **Initial**: any configuration

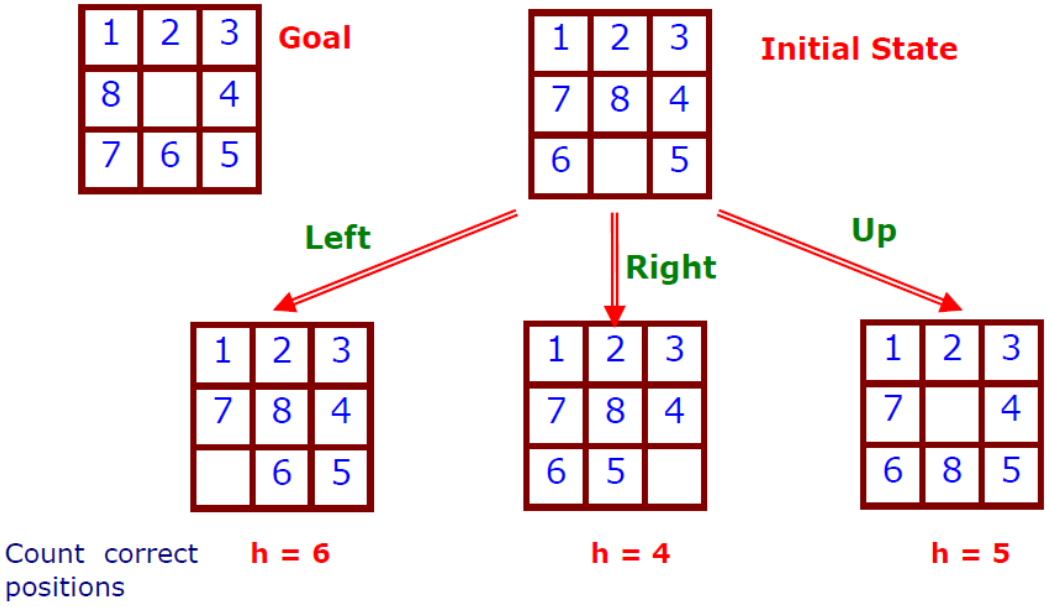
◊ **Goal**: tiles in a specific order





◊ **Actions**

Figure below shows: three possible moves - left , up, right



Apply the Heuristic : Three different approaches

- Count correct position of each tile, compare to goal state

- Count incorrect position of each tile, compare to goal state
- Count how far away each tile is from it is correct position.

Approaches	Left	Right	Up
1. Count correct position	6	4	5
2. Count incorrect position	2	4	3
3. Count how far away	2	4	4