LECTURE 7: Problems, Spaces and Search

A system to solve a particular problem in Artificial Intelligence:

The Following needed:

1)Building

(i) Define the problem precisely

(ii) Problem definition include:

(iv)Precise specification of initial(s) situation(s) as well what final situation(s) constitute acceptable solutions to the problem.

Building a system to solve a particular problem in Artificial Intelligence:

2)Analyze the problem. A few very important features can have an immerse impact on the appropriateness of various possible techniques for solving the problem

3) Isolate and represent the task knowledge that is necessary to solve the problem

4)Choose the best problem solving technique(s) and apply it(them)to the

Major Components of a production system

1) A set of rules. Each has a left side that determines the applicability of the rule and a right side that describes the operation to be performed if the rule is applied.

2) One or more knowledge/databases that contain whatever information is appropriate for the particular task

3)Control strategy that specifies the order in which the rules will be compared to the database and a way of resolving the conflicts that arise when several rules match at once.

Classical Travelling salesman problem

A salesman has a list of cities to visit. Each city must be visited exactly once. There are direct roads/routes between each pair of cities.

Find the route the salesman should follow so that he travels the shortest.

This type of problem is described as difficult to solve.

The easiest and most expensive solution is to simply try all possibilities.

Classical Travelling salesman problem

The problem is that for N cities, there are N-1

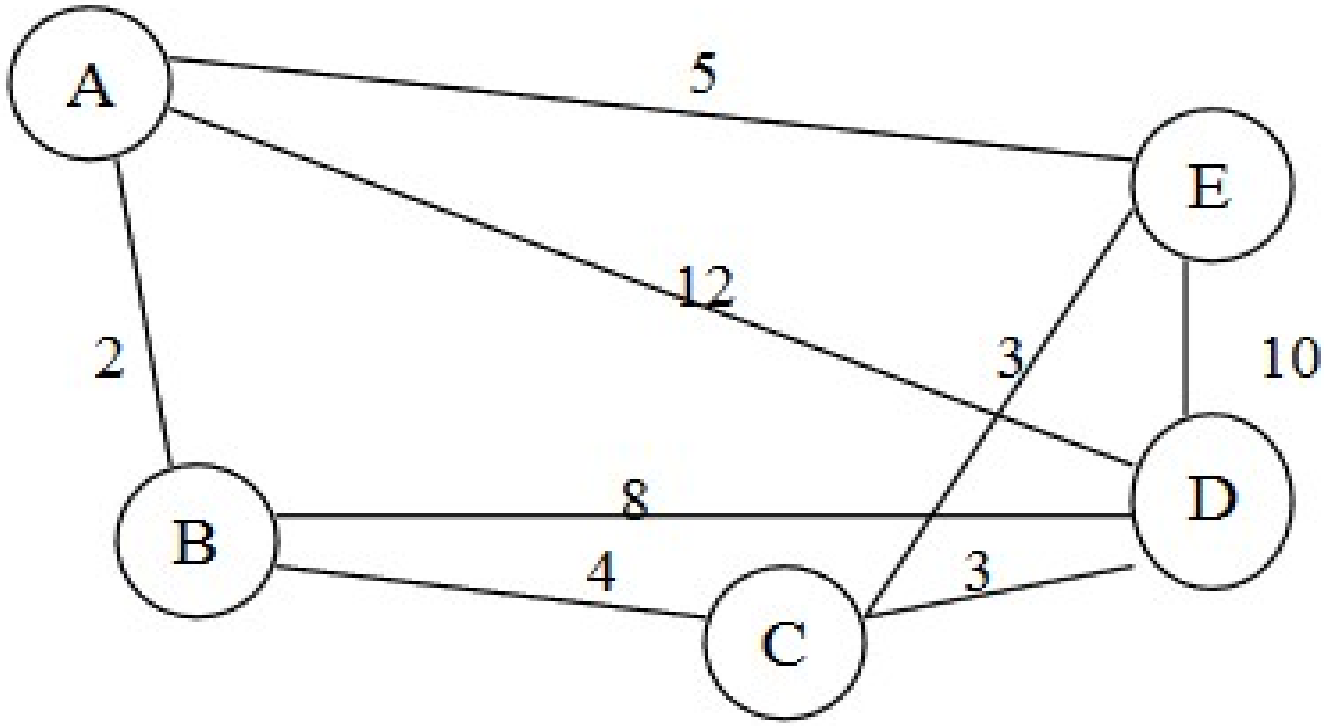
factorial, possibilities. So for 10 cities there are over 180,000 combinations to try. Since the start city is defined, there can be permutations on the remaining nine. We only count half since each route has an equal route in reverse with the same length or cost

9!/2 = 181,440

Example

The traveling salesman problem consists of a salesman and a set of cities. The salesman has to visit each one of the cities starting from a certain one (e.g. the hometown) and returning to the same city. The challenge of the problem is that the traveling salesman wants to minimize the total length of the trip.

Example: Consider 5 cities below:



Example of 5 cities

The problem lies in finding a minimal path passing all vertices once. Using the combinatorial method:

Calculate the costs of all the routes and select the minimal(smallest). Note that the cost can be in kilometers or dollars.

There can be routes with identical or same costs. In such instances the sales man can choose either.

Example continued

In the diagram:

Path 1:{A,B,C,D,E,A} Cost = 24

Path 2:{A,B,C,E,D,A} Cost = 31

Path 3:{A,E,D,C,B,A} Cost = 24

Path 4:{A,B,D,C,E,A} Cost = 21

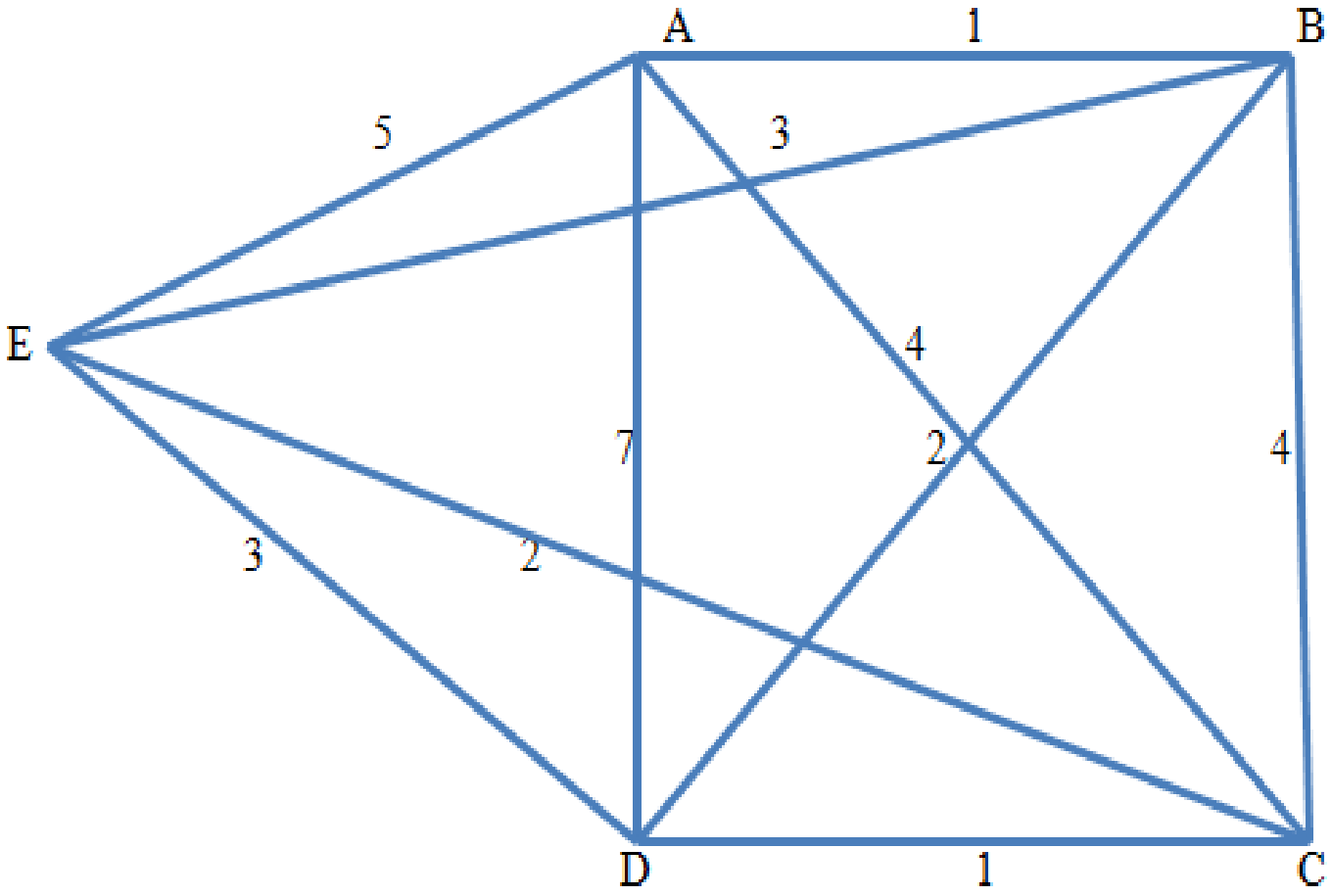
Path 5: {A,D,E,C,B,A} Cost = 31

The shortest route is path 4.

Home town in this case is A.

The shortest Route is Path 4

Heuristic method



Heuristic method

Problem statement

A salesman has a list of cities to visit. Each city must be visited exactly once. There are direct roads between each pair of the cities on the list.

Find the route the salesman should follow so that he travels the shortest possible on a round trip starting at any one of the cities and then returning home.

Solution

Using the heuristic search proceed to solve as follows:

1) Arbitrary select a starting city
2) To select next city, look at all cities not yet visited
3) Select the one closest to the current city and go to the next city
4) Repeat step 3 until all cities have been visited

Solution continued

Assume the home town is A:`

Nearest towns are E and B. B is shorter so proceed to B. At B nearest cities are E and D, E is nearer so choose E. At E nearer cities are C and D. C is nearer, choose C. At C the nearer city is D, so choose D.

So the route is: A-B-E-C-D-A with Cost = 12

But the route: A-E-C-D-B-A has a cost of 11 and is the minimum

Solution continued

This heuristic algorithm, clearly does not always find a minimum route. As emphasized earlier it can be seen that AI problems can be described as centering on search processes.

These AI problems can also be described more precisely as processes of heuristic searches.

Heuristic methods make solution of hard problems feasible.

Heuristic Search Techniques

- Used in most cases when problems are too complex to be solvable by direct methods or techniques
- Complex problems have to be solved only by suitable heuristic search techniques
- Heuristic techniques can be described independently and are domain specific
- They are called “weak methods” since they are vulnerable to combinational explosion

Heuristic Search Techniques

- The heuristic search techniques provide a framework into which domain specific knowledge can be placed either by hand or as a result of learning

The following are some general purpose control strategies used in heuristic search methods:

Depth First Search

Breath First Search

Hill Climbing

Best First Search

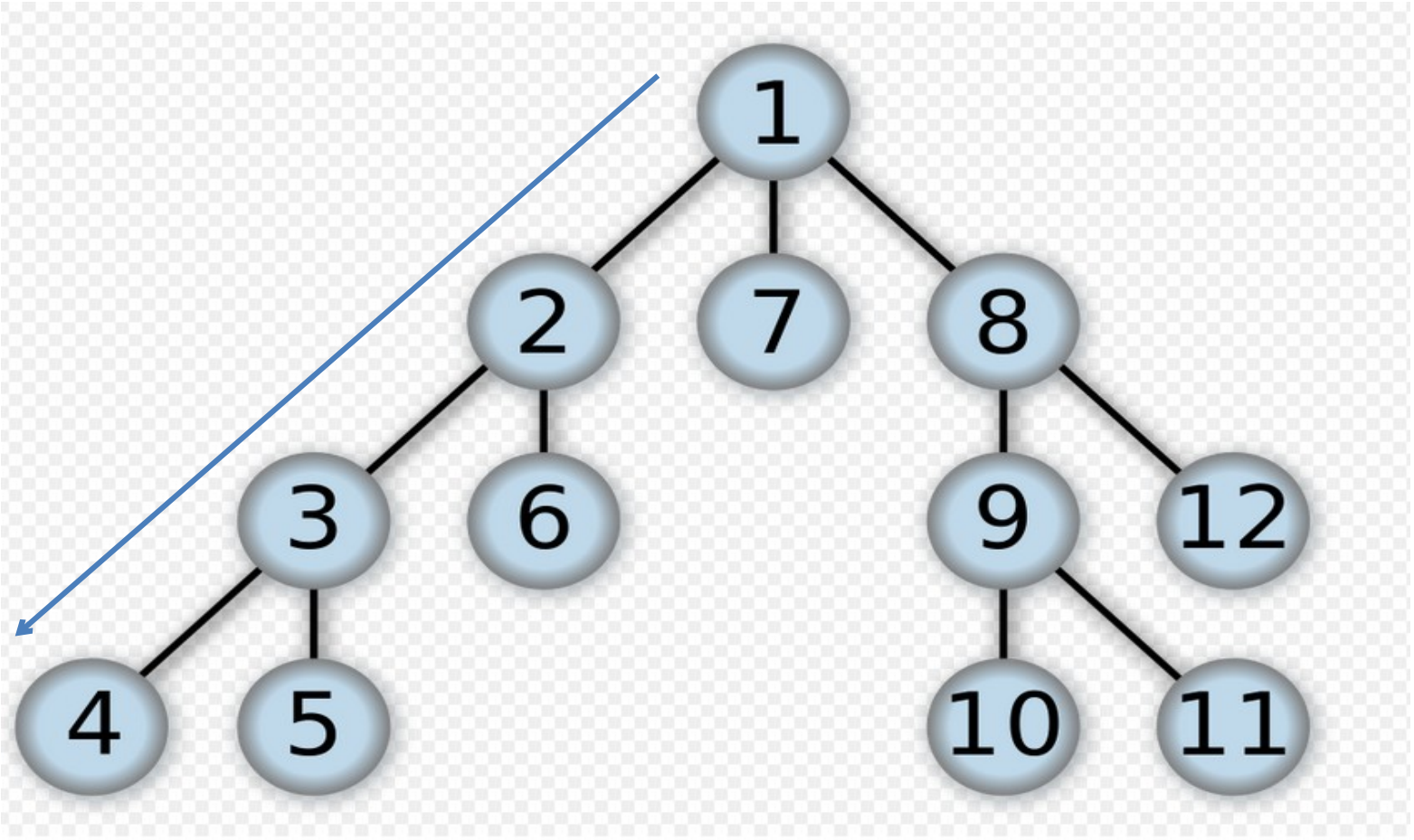
Depth First Search(DFS)

In DFS search starts from an arbitrary node which is taken as the root and explores as far as possible along each branch before backtracking

From the way DFS operates, it clear that it is a search algorithm for transversing or searching a tree or graph data structures.

Consider the graph/tree structure below:

Example of DFS operation showing the order in which the nodes are visited



Example of DFS operation showing the order in which the nodes are visited

- Node 1 is the root, so searching starts from this node until it reaches node 4, then backtracks to 3 and then proceeds to 5, back tracks to 2, then moves to 6 etc.

Time and space analysis

Differs according to application area of DFS

In AI computer Science DFS is typically used to transverse entire graph and

Time taken to transverse entire graph

Θ(V)+E, (V are vertices, E are edges, linear in the size of the graph

In these applications it also uses space O(V) in the worst case to store the stack of vertices on the current search path as well as the set of already visited vertices.

Applications of DFS in specific domains such as searching for solutions in AI or web crawling, the graph to be transversed is often either too large

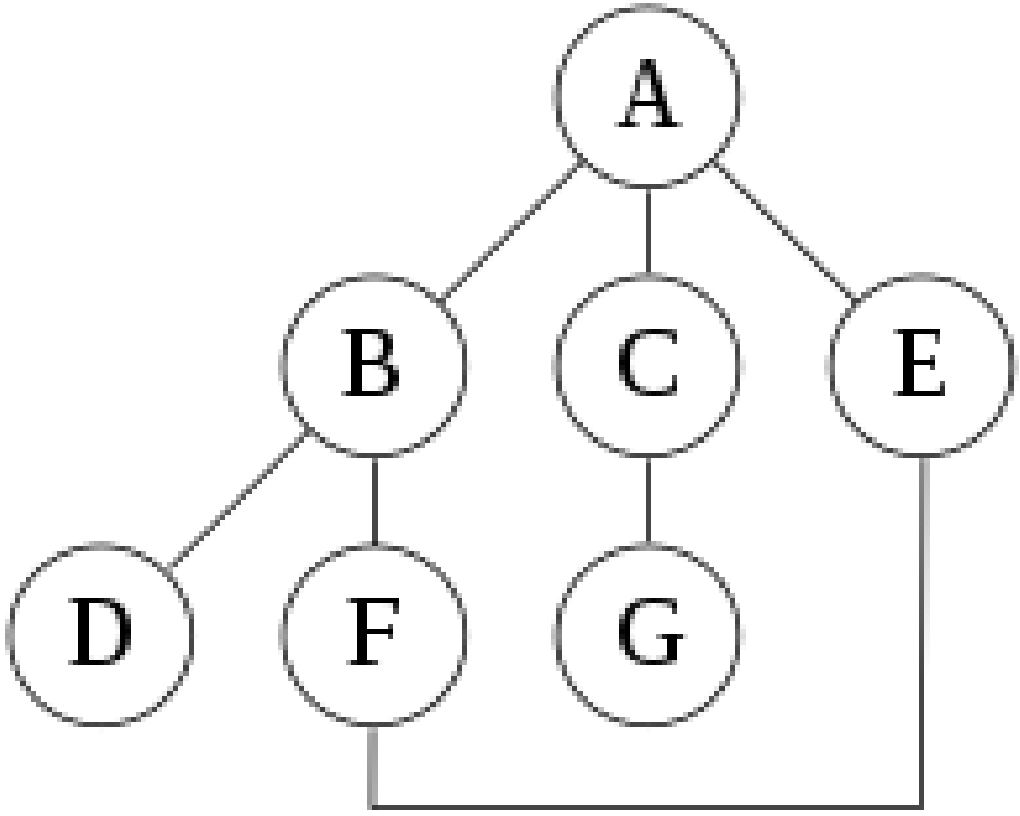
Time taken to transverse entire graph

to visit in its entirety or infinite. In such cases DFS may suffer from non-termination.

In these cases(graph too large or infinite), search is only performed to a limited depth due to limited resources such as memory or disk space.

Even when search is performed to a limited depth, the time is still linear in terms of expanded vertices and edges.

Example of DFS remembering previously visited nodes



Example explanation

Assuming that the left edges are visited before the right edges and assuming that the search remembers previously visited nodes and will not repeat them, the nodes in the graph will be transversed in the order:

A,B,D,F,E,C,G

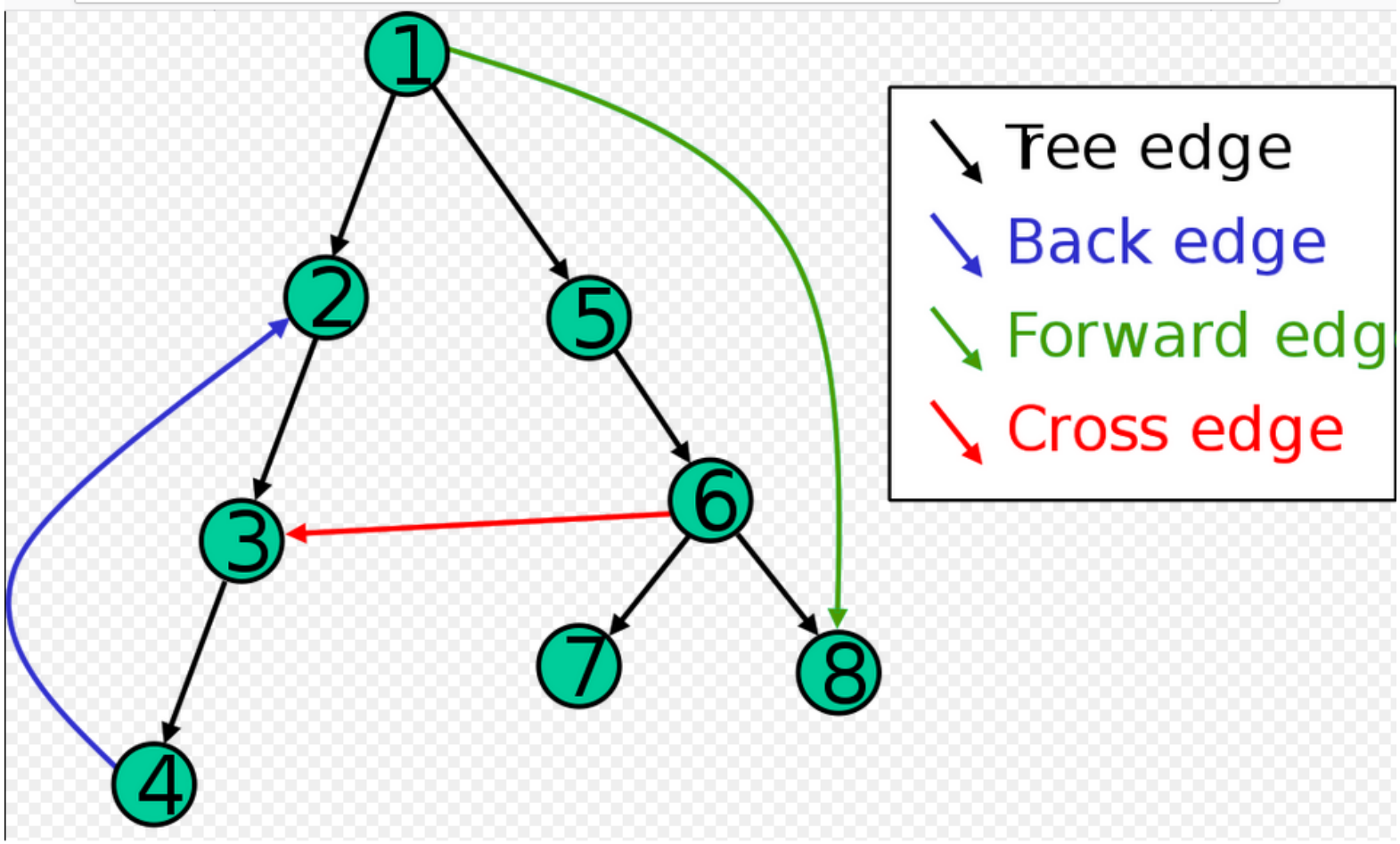
Edges transversed in this search form the Tremaux tree, a tree structure with important applications in graph theory.

Performing the same search without remembering previously visited nodes

In this case graph will be transversed as follows:

A,B,D,F,E,A,B,D,F,E etc. forever caught in the A,B,D,F,E cycle and never reaching C or G.

Spanning tree



Depth First Search Algorithm

Begin

Open: = [start]

Closed: =[ ];

While open ≠ [ ] Do,

Begin

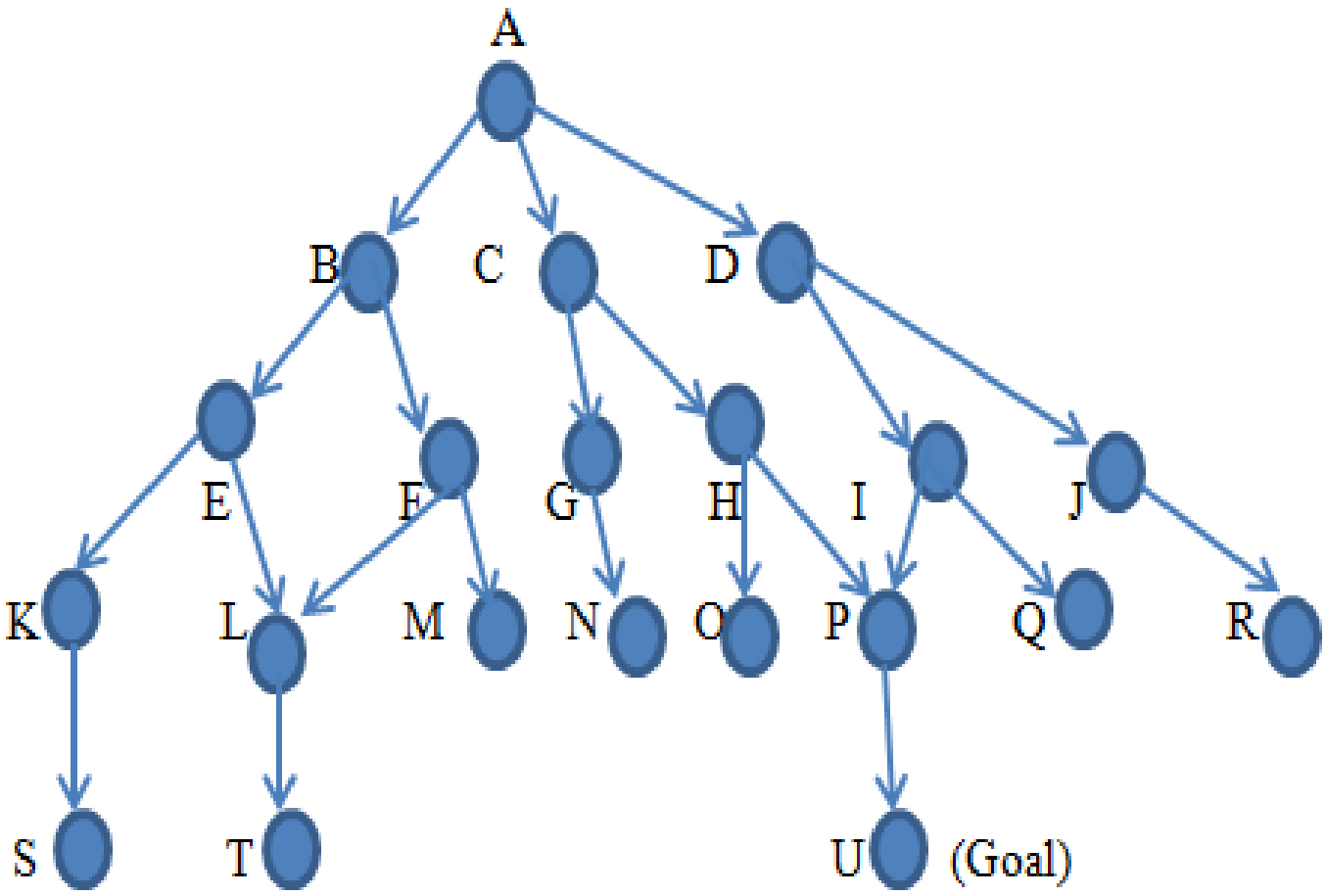
Remove leftmost state from open call it X;

IF X is a goal then return SUCCESS

Depth First Search Algorithm

```
else begin
generate children of X;
put x on closed;
discard children of X if already
on open or closed;
put remaining children on left
 end on open
end
   end;
Return FAIL
end
```

Example of DFS algorithm operation



Example of DFS algorithm operation

Assume U is the Goal. A trace of the DFS on the above graph can be shown as follows:

1) Open = [A]; closed = [ ]
2) Open =[B,C,D]; closed = [A]
3) Open =[E,F,C,D]; closed = [B,A]
4) Open = [K,L,F,C,D];closed = [E,B,A]
5) Open = [S,L,F,C,D]; closed = [K,E,B,A]
6) Open = [L,F,C,D]; closed = [S,K,E,B,A]
7) Open = [T,F,C,D]; closed=[L,S,K,E,B,A]

Example of DFS algorithm operation(continued)

8 open = [F,C,D]; closed = [T,L,S,K,E,B,A]

9 open =[M,C,D];closed = [F,T,L,S,K,E,B,A]

L is already closed

10 open =[C,D];closed = [M,F,T,L,S,K,E,B,A]

11 open=[G,H,D];closed = [C,M,F,T,L,S,K,E,B,A]

This process will continue until either U is discovered on open or open = [ ]

Each successive iteration of the “while” loop is indicated by a single line (2,3,4,5,6,. . .).

Example of DFS algorithm operation(continued

The initial states of open and closed are given on line 1

DFS is not guaranteed to find the shortest path to a state the first time that state is encountered

If path length matters in a problem solver, when the algorithm encounters duplicate state , the algorithm should save the version with the shortest path.

Example of DFS algorithm operation(continued

This can be done by storing each state as a triple (state, parent, length of path). If a child is reached along multiple paths, this information can be used retain the best(shortest) path.

Example 2

A standard DFS implementation puts each vertex of the graph into one of two categories:

- Visited
- Not Visited

The ides of the algorithm is to mark each vertex as visited while avoiding cycles.

Example 2

The DFS algorithm works as follows:

1)Start by putting any one of the graph's vertices on top of a stack.

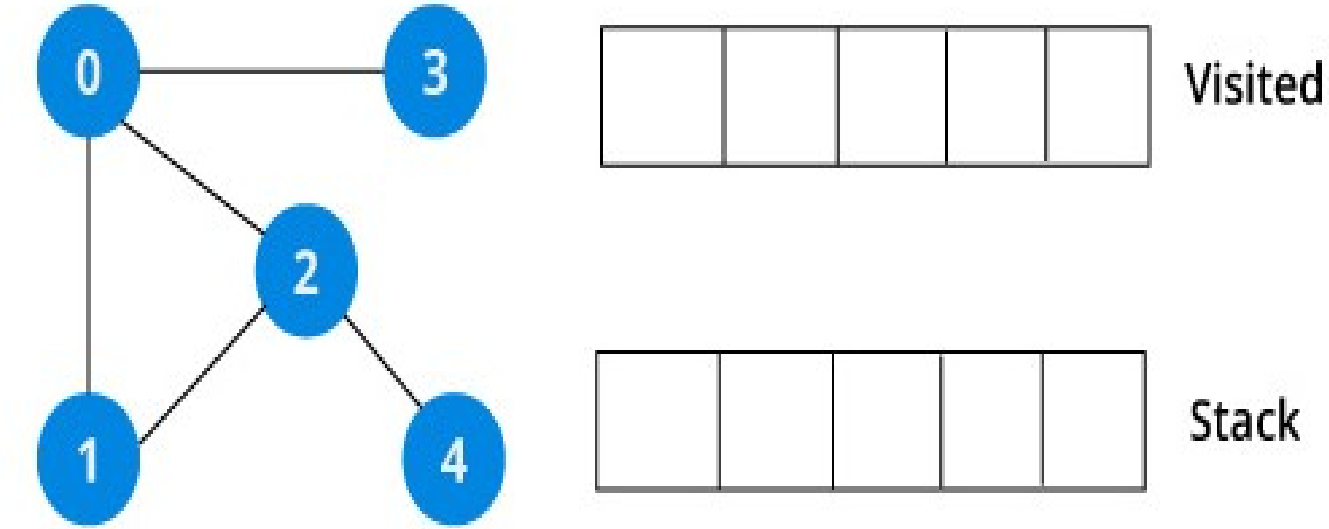
2)Take the top item of the stack and add it to the visited list.

3)Create a list of that vertex's adjacent nodes. Add the ones which aren't in the visited list to the top of the stack.

4)Keep repeating steps 2 and 3 until the stack is empty.

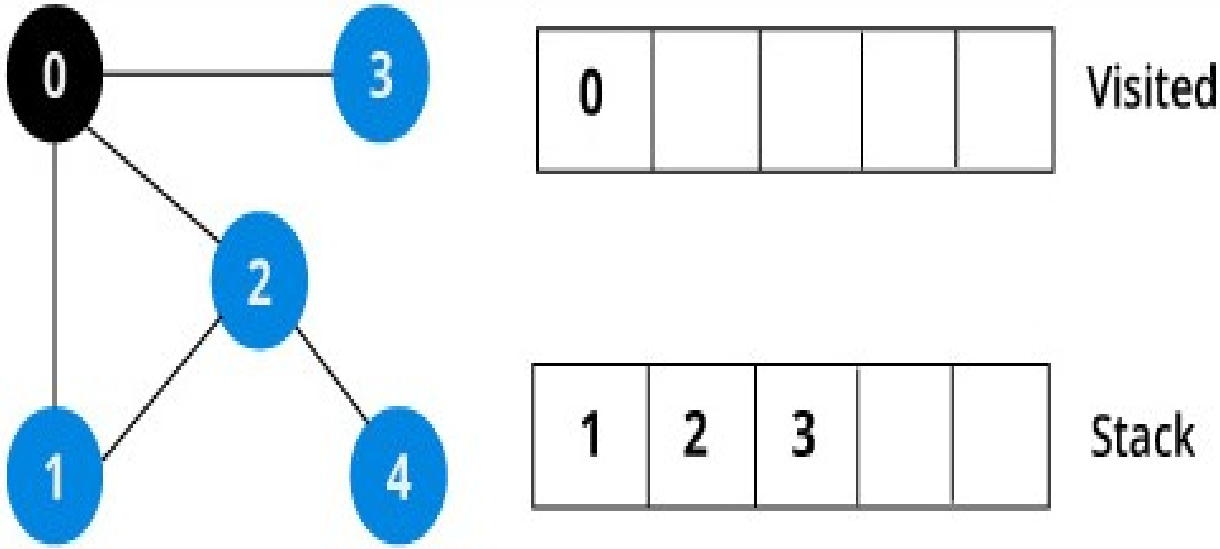
Example 2

Consider an undirected graph with 5 vertices



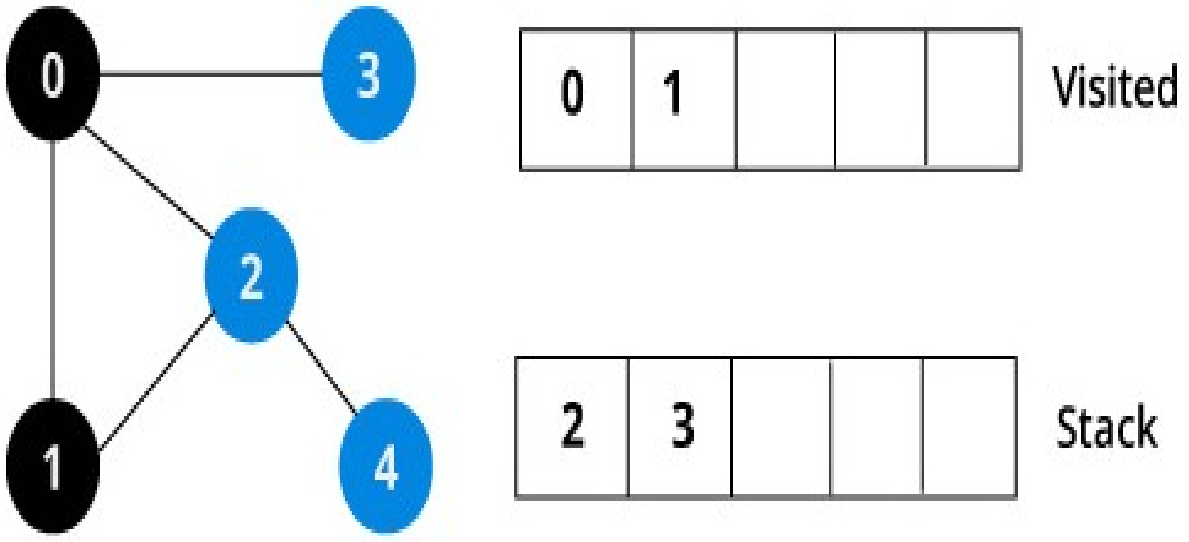
Example 2(continued)

We start from vertex 0, the DFS algorithm starts by putting it in the Visited list and putting all its adjacent ve



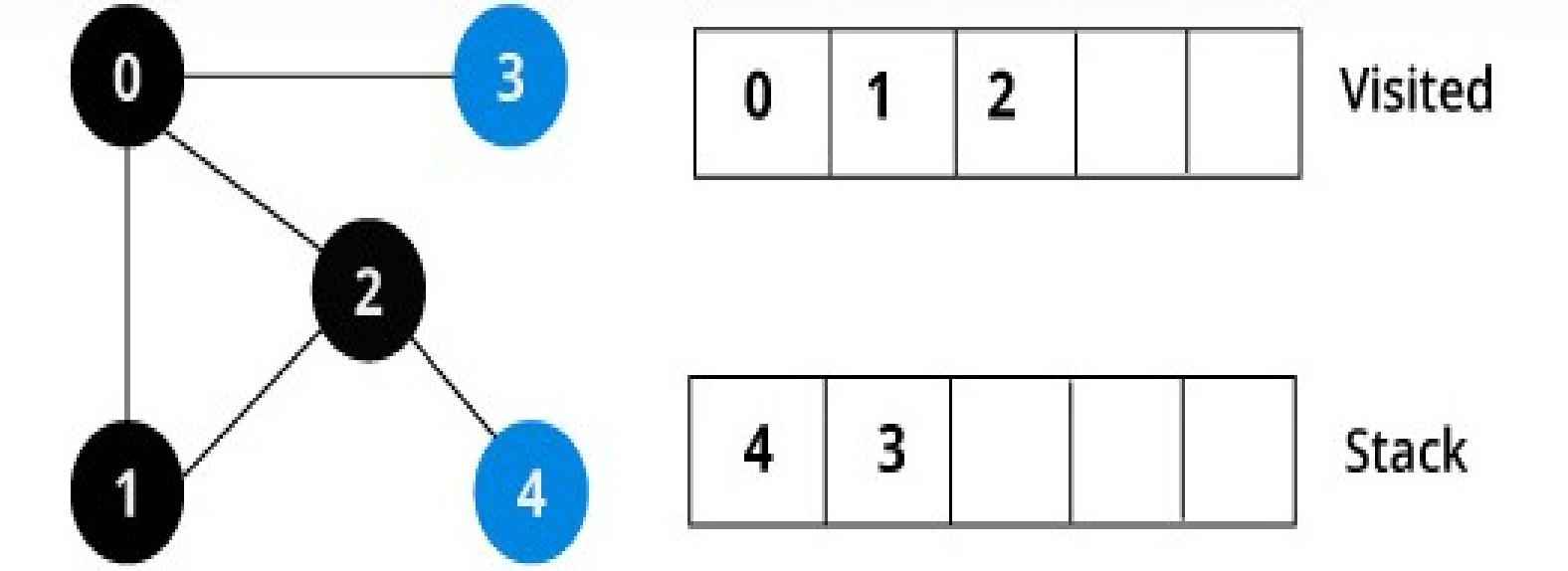
Example 2(continued)

Next, we visit the element at the top of stack i.e. 1 and go to its adjacent nodes. Since 0 has already been visi



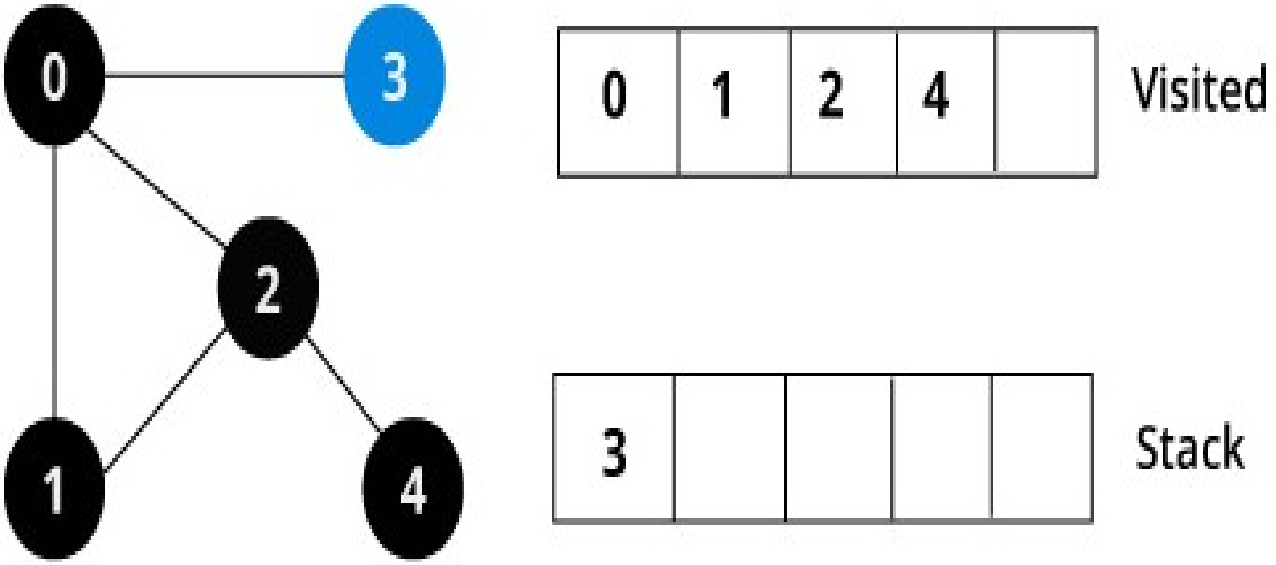
Example 2(continued)

Vertex 2 has an unvisited adjacent vertex in 4, so we add that to the top of the stack and visit it.

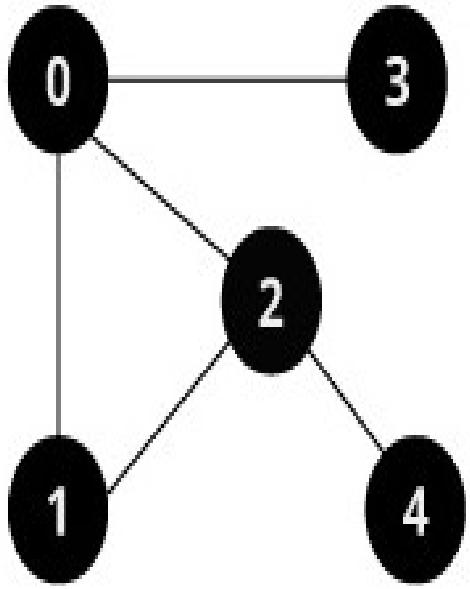


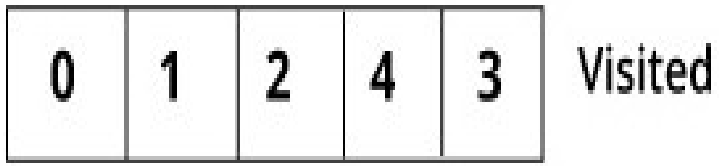
Example 2(continued)

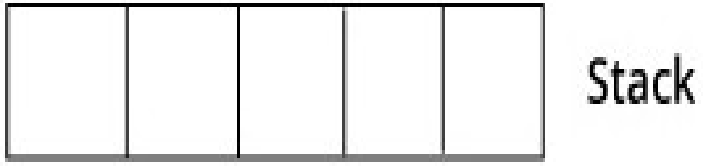
After we visit the last element 3, it doesn't have any unvisited adjacent nodes, so we have completed the Depth First Traversal of the graph.



Example 2(continued)







Advantages and Disadvantage of DFS

Depth-first search 3

- advantages of the method:
  - easy implementation
  - modest memory requirement (b*d, b: branching factor, d: dept
- disadvantages of the method:
  - can get stuck going down the wrong path (very deep or infinite search tree) → incomplete
  - can find solution that is more expensive than the optimal ones not optimal

Class activity

Consider the 8-puzzle below:
Use DFS to reach the goal

Initial State

2	8	3
1	6	4
7		5

Goal State

1	2	3
8		4
7	6	5