Lecture Two

Structure and Strategies for State Space Search

# Introduction

* Predicate calculus describes the objects and the relation between them.
* Inference rules are used to infer new knowledge from these descriptions.

*•* Predicate calculus + Inference rules are used to define the search space.

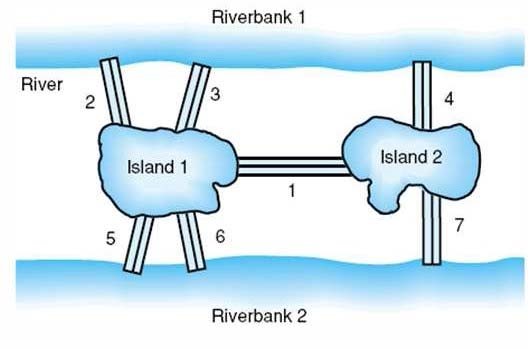
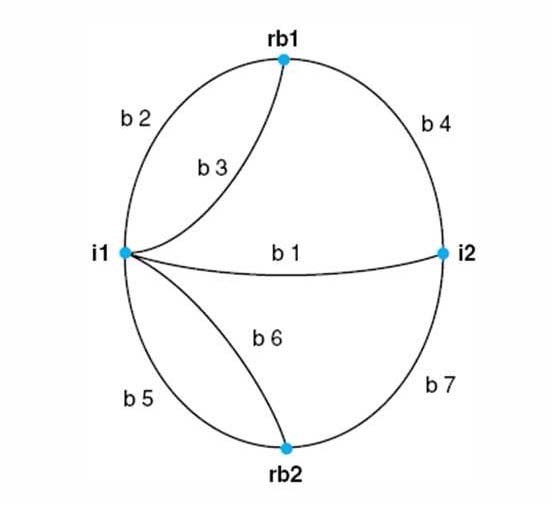
As a good programmer, you have to be able to predict and analyze the algorithm behavior such as

* Is there a solution?
* Will the algorithm always terminate?
* When the solution is found, Is it optimal?
* What is the complexity of the used algorithm? A graph **G** = (**V***,* **E***,* **W**) where,
* **V** = *{V*1*, V*2*, . . . , Vd}* is a finite set of vertices, which represents discrete state in the problem solving process.
* **E** is a finite set of edges *{(Vi, Vj )}*, where *(Vi, Vj )* represents an edge between the

vertices *Vi* and *Vj* . Each edge represents transition between states.

* **W** = *{wi,j }* is a finite set of weights, where *wi,j* is the weight assigned to the edge *(Vi, Vj )* in the graph.

***Example 1: Bridges of Königsberg***



Represent the system using first-order predicate logic:-

* connect(i1,i2,b1)

|  |  |
| --- | --- |
| *•* connect(i2,i1,b1) | Each bridge requires two connect predicates, |
| *•* connect(rb1,i1,b2) | one for each direction in which the bridge |
| *•* connect(i1,rb1,b2) | may be crossed. |

. .

* Most people were convinced that no path possible such that every bridge is crossed only once. But, no proof available.
* Inspired Euler to invent graph theory, Euler focused on the degree of the nodes of the graph, observing that a node could be either even or odd degree.
* Euler noted that unless a graph contained either exactly zero or two nodes of odd degree, the walk was impossible.

# The State Space Representation

The problem is represented by

1. Initial State: Starting state

1. Set of operators: Actions responsible for transition between states

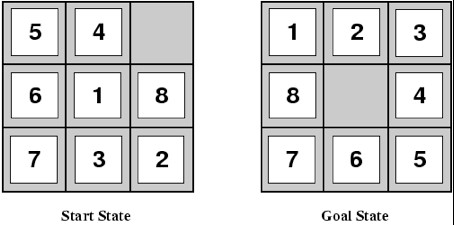
1. Goal test function: Applied to a measurable property of the state to deter- mine if it is a goal state, or measurable property to the solution path.

1. Path cost function: Assigns a cost to a path to tell if a path is preferable to another

* Search space: the set of all states that can be reached from the initial state by any sequence of action.
* Search algorithm: how the search space is visited. *•* Main problems in designing the graph search algorithm are:

* 1. States can sometimes be reached through different paths, therefore, the important thing is to choose the **best** path according to the needs of the problems.
  2. Multiple paths could lead to loops or cycles in a solution path that could prevent the algorithm from reaching a goal.

# Homework 1:

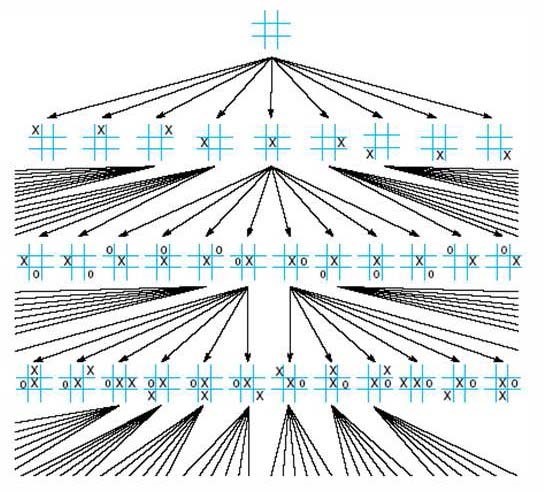


*•* Set of operators: blank moves up, blank moves down, blank moves left, blank moves right. *•* Goal test function: state matches the goal state. *•* Path cost function:

each movement costs 1, so the path cost is the length of the path (the number of moves). *•* Complexity of the problem:

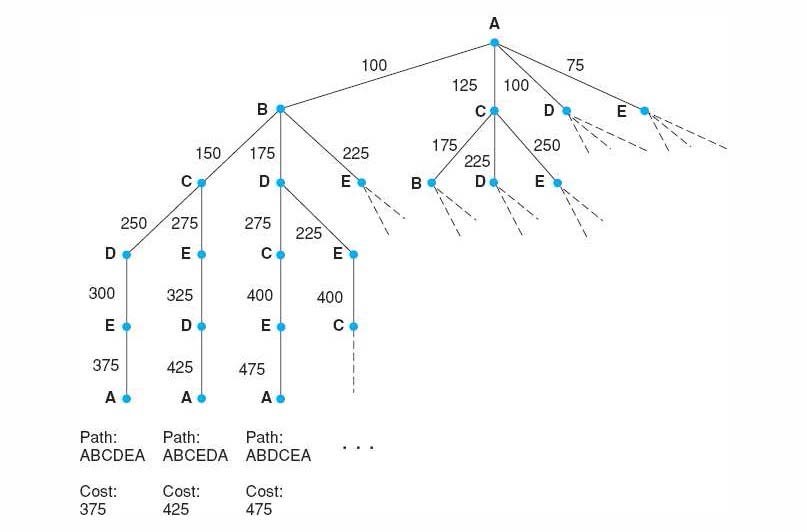
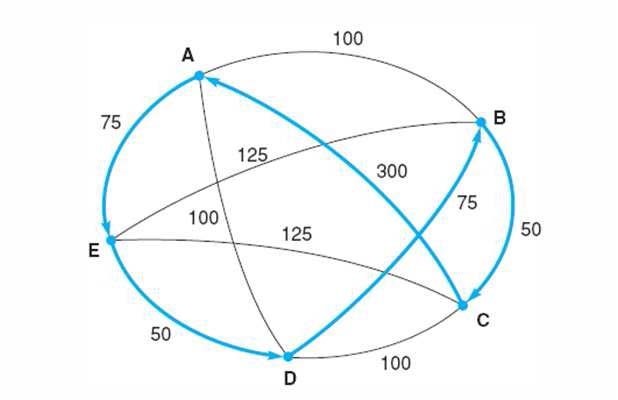
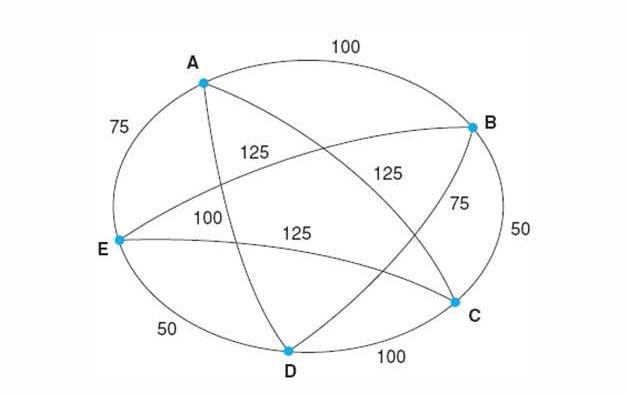
1. 8-puzzle: 8! = 40*,* 320 different states.
2. 16-puzzle: 16! = 20*,* 922*,* 789*,* 888*,* 000 *""* 1013 different states.

# Example 2



* Set of operators: Draw *X* in the blank state.
* Goal test function: Having three *X*’s in a row, column and diagonal.
* The path from the start state to a goal state gives the series of moves in a winning game.
* Complexity of the problem: 9! = 362*,* 880 different states.
* This graph is a Directed Acyclic Graph *DAG*, because it is impossible to go back up the structure once a state has been reached.

# Example 3



* Salesperson has to visit 5 cities and he must return home afterwards.
* Goal: find shortest path for travel *""* minimize cost and/or time of travel.
* Nodes represent cities and the Weighted arcs represent cost of travel.
* Simplification: salesperson lives in city A and will return there.
* Complexity of the problem: (N - 1)! with N the number of cities

# Strategies for state Space Search

* Data-driven vs. goal-driven search

Data driven (forward chaining) Start from available data, search for goal.

*•* Goal driven (backward chaining)

Start from goal, generate sub-goals until arriving at initial state.

*•* Best strategy depends on problem.

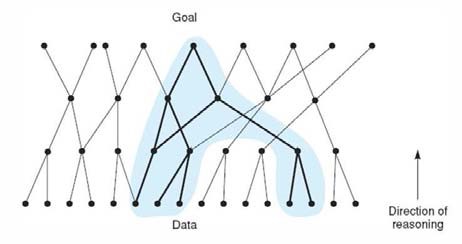
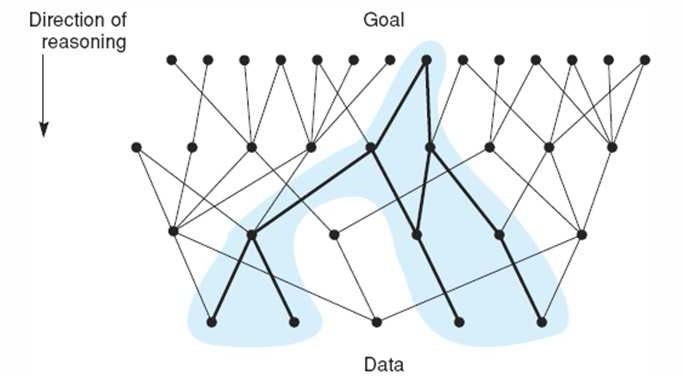


Figure 3: (a) Data driven approach.(b) Goal driven approach

**Example:**

Prove: “I am a descendant of Thomas Jefferson”

1. Start with yourself (goal) until Jefferson (data) is reached.

1. Start with Jefferson (data) until you reach yourself (goal).

Assume the following:

*•* Jefferson was born 250 years ago.

*•* 25 years per generation: length of path is 10.

1. Goal-driven search space

If each person has 2 parents, then the search space: order of 210 ancestors.

1. Data-driven search space

If average of 3 children per family, then the search space: order of 310descendent.

So goal driven (backward chaining) is better, but both directions yield exponential complexity

*•* **Use the goal driven approach when:**

* 1. A goal or hypothesis is given in the problem statement or can easily be formulated.
  2. There are a large number of rules that match the facts of the problem and thus produce an increasing number of conclusions or goals.
  3. Problem data are not given but must be acquired by the problem solver.

*•* **Use the data driven approach when:**

1. All or most of the data are given in the initial problem statement.

1. There are a large number of potential goals, but there are only a few ways to use the facts and given information of a particular problem instance.
2. It is difficult to form a goal or hypothesis.

# State Space Description of a Logic System

**Example 1:**

1. *q → p*
2. *r → p*
3. *v → q*
4. *s → r*
5. *t → r*
6. *s → u*

1. *s*

1. *t*

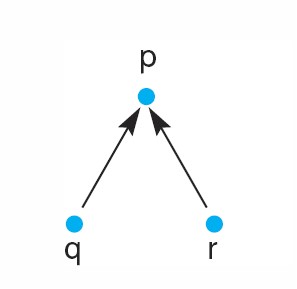
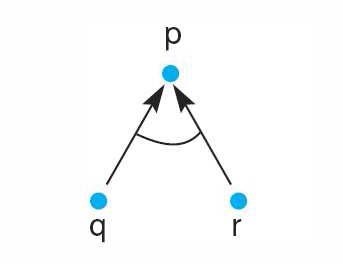
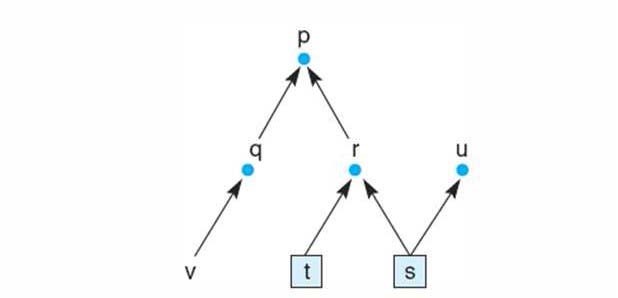
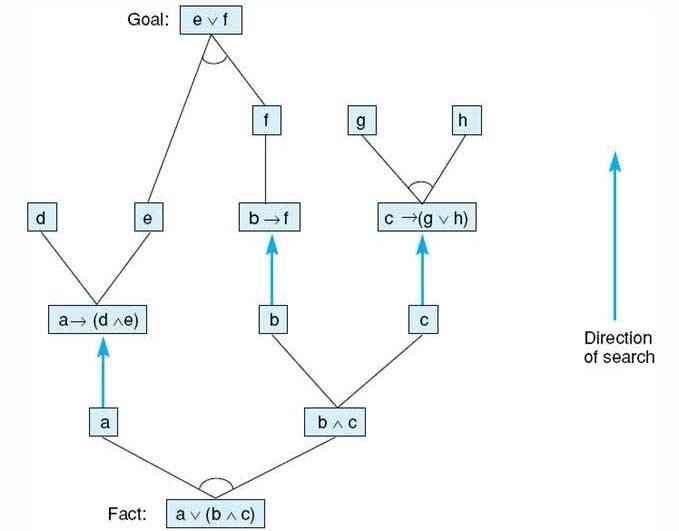
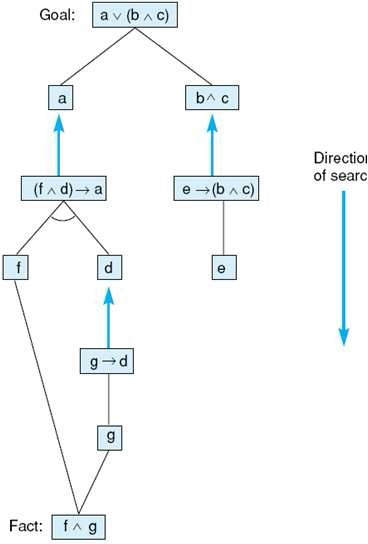


Figure 4: (a) *r* ∨ *q → p* (b) *r* ∧ *q → p*



**Example 3:**



# Search Algorithms

The search space algorithms can be divided into:

* Uninformed search (systematic/exhaustive/blind/brute force search)

* 1. Backtracking technique.

* 1. Breath-first search.

* 1. Depth-first search.

* Informed search (heuristic search)

* 1. Hill climbing search.

* 1. Best-First search.
  2. *A*∗algorithm.

# Backtracking Technique

* It is a techniques for systematically trying *all* pathes through a state space.

In addition to not get stuck in cycles.

* ***Idea:*** keep track of visited nodes and apply recursion to get out of dead ends.

* ***Termination:***

If it finds a goal, it quits and returns the solution path, or state space is exhausted. *•* ***Backtracking:***

* + If it reaches a dead end, it backtracks to the most recent node on the path having unexamined siblings and continues down one of these branches.
  + It requires stack oriented recursive environment.

* ***Details of backtracking:***

* + *SL* (State List):

∗States in current path being tried.

∗If goal is found, SL contains ordered list of states on solution path.

* + *N SL* (New State List)

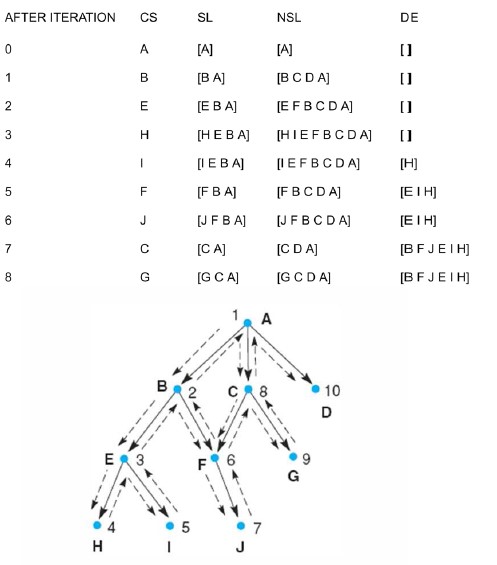
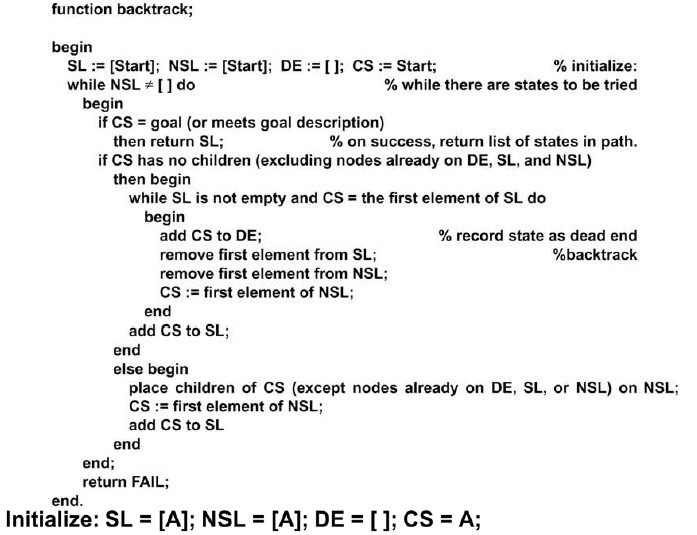
∗nodes awaiting evaluation.

∗Nodes whose descendants have not been generated and searched.

* + *DE* (Dead Ends)

∗States whose descendants failed to contain a goal node.

∗If encountered again, they are recognized and eliminated from search.



15

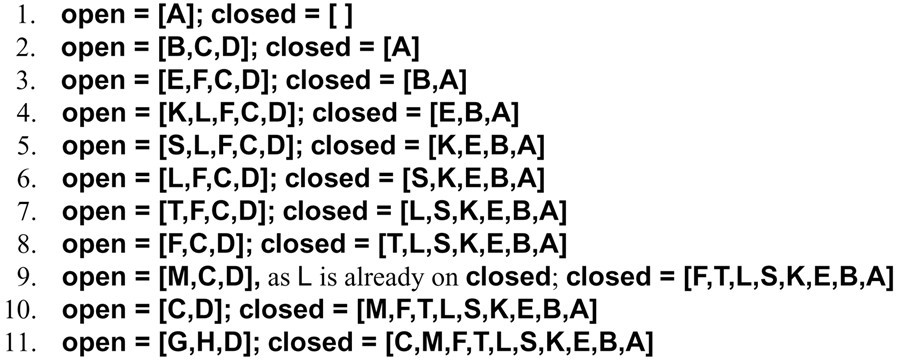
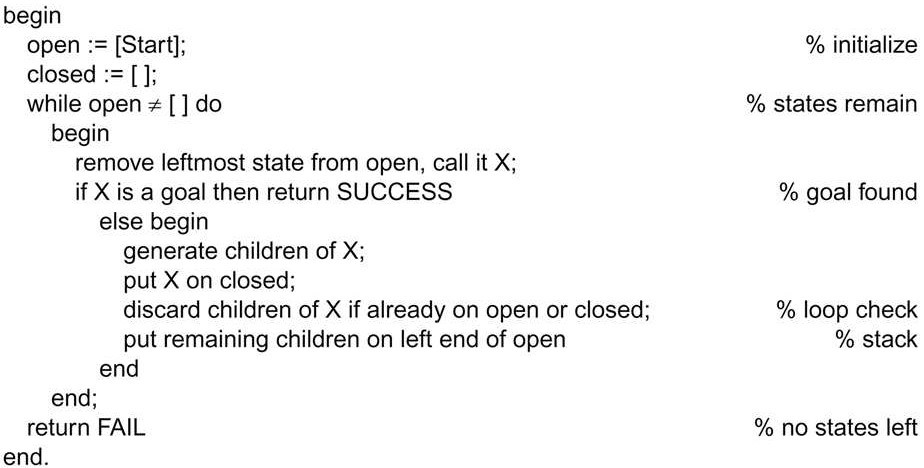
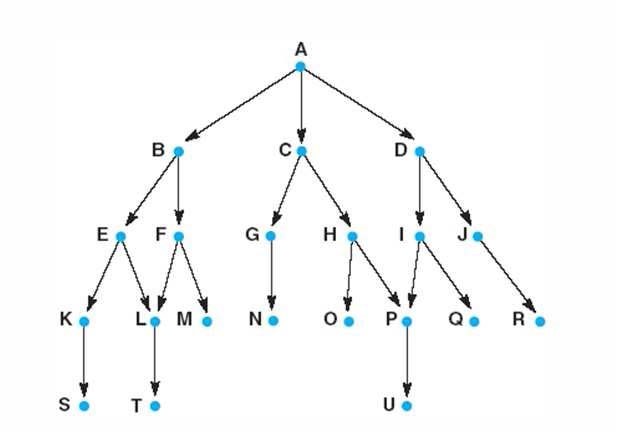
1. Backtrack is a data driven search because it starts from the roots then it evaluates its descendent children to search for the goal.

1. Backtrack can be viewed as a goal driven by letting the goal be a root of the graph and evaluating descendent back in attempting to find the a start state (root).

1. Backtrack prevents looping by explicit checking for membership of *N S*.

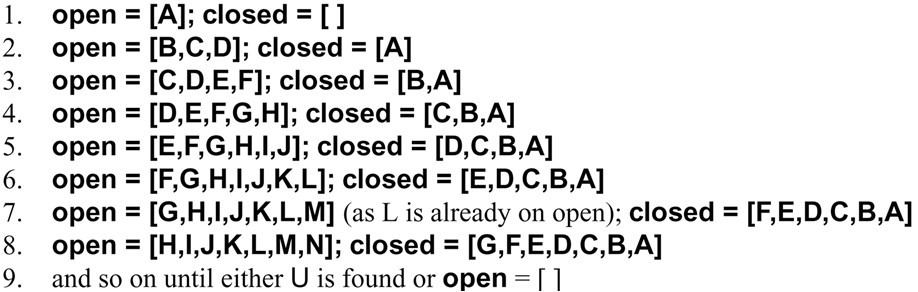
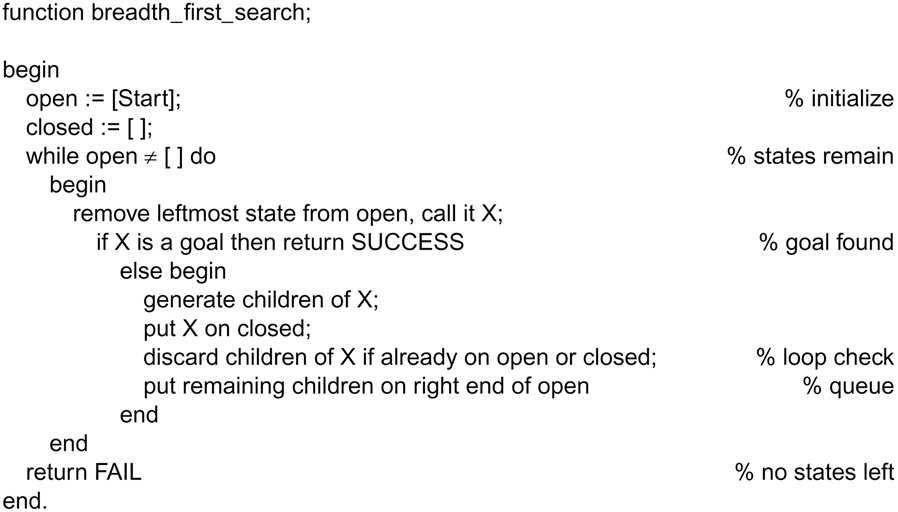
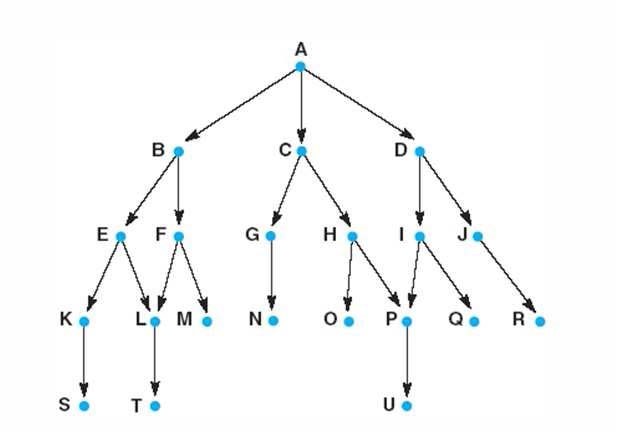
# Depth-First Search Algorithm

When the state is examined, all of its children and their descendants are exam- ined before any of its siblings.



# Breadth-First Search Algorithm

It explores the space in a level by level fashion. Only when there are no more states to be explored at a given level, the algorithm move to the next level.



# Difference Between Depth and Breadth Algorithms

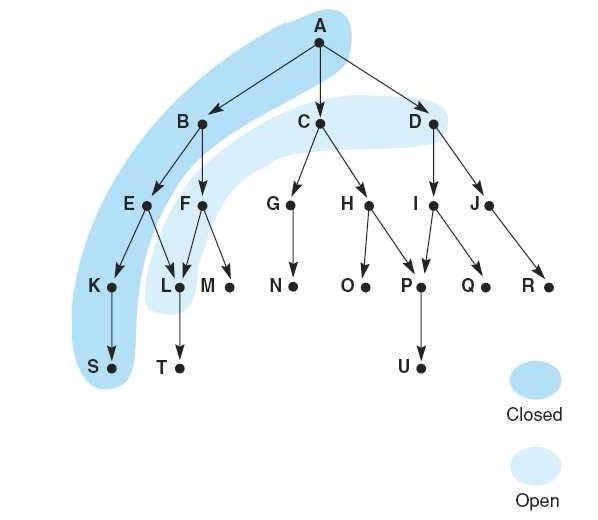
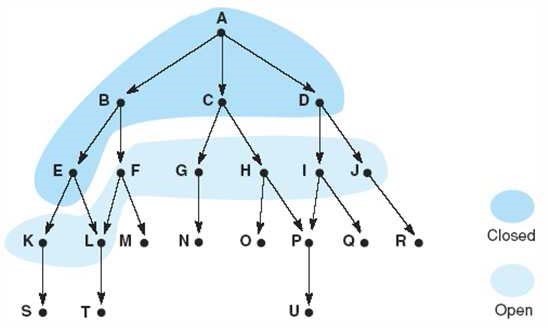


Figure 5: (A) Breadth-First Search (b) Depth-First Search

* *DF S* and *BF S* with respect to ordering nodes in the open list:

* + *DF S* uses a stack (nodes are added on the top of the list)

* + *BF S* uses a queue (nodes are added at the end of the list)

* *DF S* and *BF S* with respect to examination process:

* + *DF S* examines all the node’s children and their descendent before the node’s siblings.
  + *BF S* examines all the node’s siblings and their children.

* *DF S* and *BF S* with respect to completeness:

* + *DF S* is not complete (it may be stuck in an infinite brunch)

* + *BF S* is complete (it always finds a solution if it exists)

* *DF S* and *BF S* with respect to optimality:

* + *DF S* is not optimal (it will not find the shortest path)

* + *BF S* is optimal (it always finds shortest path)

* *DF S* and *BF S* with respect to memory:

* + *DF S* requires less memory (only memory for states of one path needed:

*B × n*)

* + *BF S* requires less memory (exponential space for states required: *Bn*)

* *DF S* and *BF S* with respect to efficiency:

* + *DF S* is efficient if solution path is known to be long.

* + *BF S* is inefficient if branching factor B is very high.

*DF S* and *BSF* do not maintain a list of states which helps us in retrieving the solution path.

We can found the solution path by storing ancestor information along with each state. Whenever the goal is found, the algorithm can construct the solution path by tracing back along parents from the goal to the start state. (How could we recognize the root?)

The choice of the *DF S* or *BF S* depends on the problem being solved:

* Importance of finding the shortest path.
* The branching factor of the space.
* The available compute time and space resources.
* The average length of paths to a goal node.
* Whether we are looking for all solutions of the first one.