CHAPTER 11 Trees

Problems:

- **How should items in a list be stored so that an item can be easily located?**

  Binary search trees

- **What series of decisions should be made to find an object with a certain property in a collection of objects of a certain type?**

  Decision trees

- **How should a set of characters be efficiently coded by bit strings?**

  Prefix codes

3. Prefix Codes

- The problem of using bit strings to encode the letters of the English alphabet.

  How to improve coding efficiency?

- Using bit strings of different lengths to encode letters can improve coding efficiency.

  How to ensure the code having the definite meaning?

  For example, e: 0 a: 1 t: 01

  0101: eat, tea, eaea, tt?

- When letters are encoded using varying numbers of bits, some method must be used to determine where the bits for each character start and end.

❁ the Concept of Prefix Codes

- To ensure that no bit string corresponds to more than one sequence of letters, the bit string for a letter must never occur as the first part of the bit string for another letter. Codes with this property are called *prefix codes*.

For example,

e: 0 a: 10 t: 110

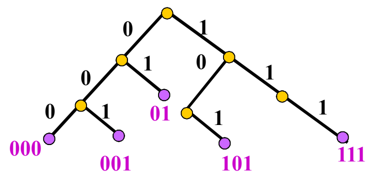
How to Construct Prefix Codes

- Using a binary tree.

  -- the left edge at each internal vertex is labeled by 0.

  -- the right edge at each internal vertex is labeled by 1.

  -- the leaves are labeled by characters which are encoded with the bit string constructed using the labels of the edges in the unique path from the root to the leaves.



Huffman Coding

Problem: How to produce efficient codes based on the frequencies of occurrences of characters?

For example,

Character	a	b	c	d	e	……
Frequences (w_i)	0.30	0.14	0.28	0.38	0.13	……

$$obj.\ \ \min\left(\sum_{i=1}^{26} l_i w_i\right)$$

where l_i is the length of prefix codes for characters i.

General problem: Tree T has t leaves, $w_1, w_2, \ldots, w_t$ are weights, $l_i = l(w_i)$. Let the weight of tree T be $w(T) = \sum_{i=1}^{t} l_i w_i$

$$obj.\ \ \min(w(T))$$

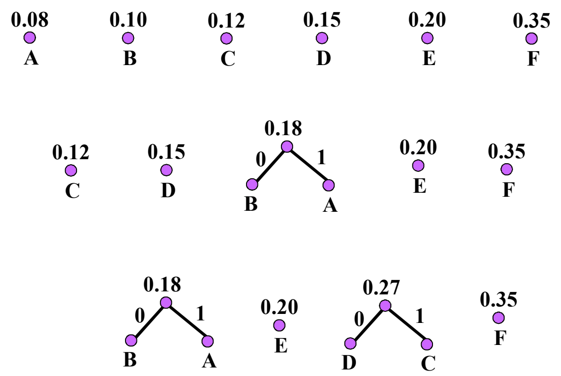
❁ Huffman Coding

Algorithm 2 Huffman Coding.

Procedure *Huffman* (C: symbols a_i with frequencies w_i, i=1, …, n)
F:=forest of n rooted trees, each consisting of the single vertex a_i and assigned weight w_i
While F is not a tree
begin
Replace the rooted trees T and T' of least weights from F with $w(T) \geq w(T')$ with a tree having a new root that has T as its left subtree and T' as its right subtree. Label the new edge to T with 0 and the new edge to T' with 1.
Assign $w(T)$ + $w(T')$ as the weight of the new tree.
end
{The Huffman coding for the symbol a_i is the concatenation of the labels of the edges in the unique path from the root to the vertex a_i}

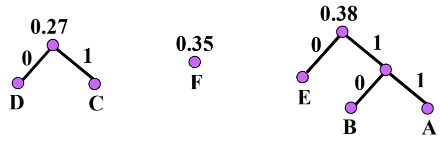
〖Example 1〗 Use Huffman coding to encode the following symbols with the frequencies listed: A:0.08, B:0.10, C:0.12, D:0.15, E:0.20, F:0.35. What is the average number of bits used to encode a character?

Solution:

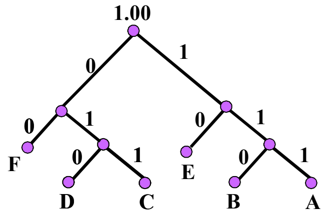


〖Example 1〗 Use Huffman coding to encode the following symbols with the frequencies listed: A:0.08, B:0.10, C:0.12, D:0.15, E:0.20, F:0.35. What is the average number of bits used to encode a character?

Solution:



......



Homework:

Sec.11.2 **20, 23**

CHAPTER 11 Trees

1. Traversal Algorithms

- ***A traversal algorithm*** **is a procedure for systematically visiting every vertex of an ordered rooted tree.**
- **Tree traversals are defined recursively.**
- **Three traversals are named:**
  - ✓ **preorder,**
  - ✓ **inorder,**
  - ✓ **postorder.**

☸ PREORDER Traversal Algorithm

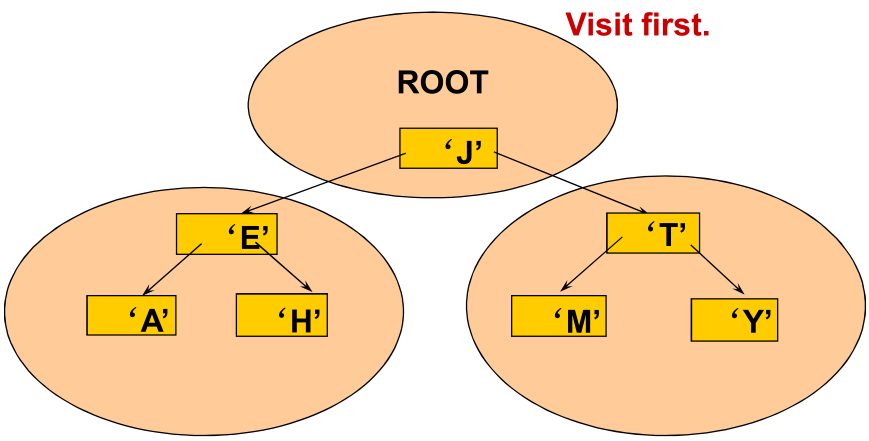
【Definition】 Let T be an ordered tree with root r. If T has only r, then r is the *preorder traversal* of T. Otherwise, suppose T_1, T_2 , …, T_n are the subtrees at r from left to right in T. The *preorder traversal* begins by visiting r. Then traverses T_1 in preorder, then traverses T_2 in preorder, and so on, until T_n is traversed in preorder.

Note:

Preorder traversal of an binary ordered tree

- Visit the root.
- Visit the left subtree, using preorder.
- Visit the right subtree, using preorder.

Preorder Traversal: J E A H T M Y



Visit left subtree second

Visit right subtree last

❁ INORDER Traversal Algorithm

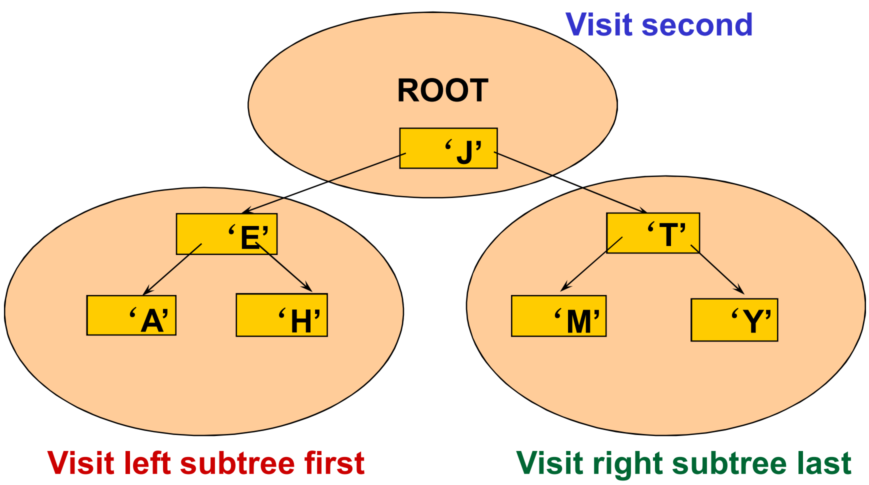
【Definition】 Let T be an ordered tree with root r. If T has only r, then r is the *inorder traversal* of T. Otherwise, suppose T_1, T_2,…, T_n are the left to right subtrees at r. The *inorder traversal* begins by traversing T_1 in inorder. Then visits r, then traverses T_2 in inorder, and so on, until T_n is traversed in inorder.

Note:

Inorder traversal of an binary ordered tree

- Visit the left subtree, using inorder.
- Visit the root.
- Visit the right subtree, using inorder.

Inorder Traversal: A E H J M T Y



☸ POSTORDER Traversal Algorithm

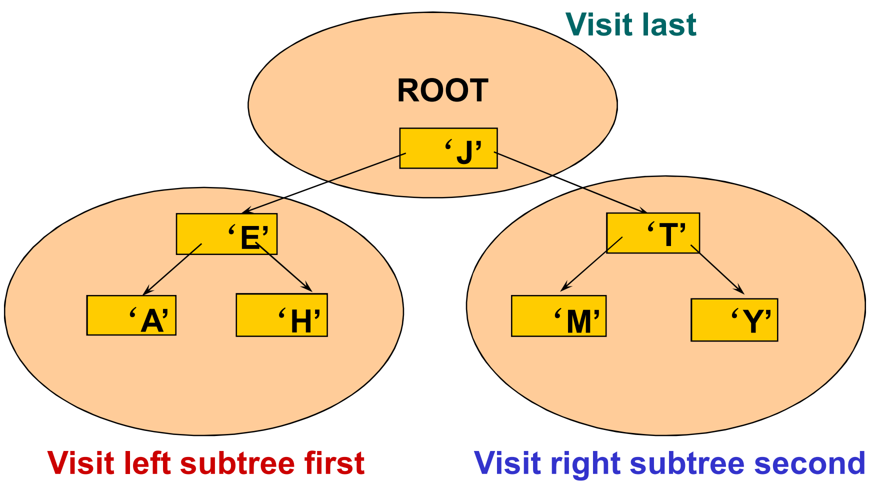
【Definition】 Let T be an ordered tree with root r. If T has only r, then r is the postorder traversal of T. Otherwise, suppose $T_1, T_2, \ldots, T_n$ are the left to right subtrees at r. The postorder traversal begins by traversing T_1 in postorder. Then traverses T_2 in postorder, until T_n is traversed in postorder, finally ends by visiting r.

Note:

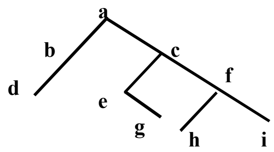
Postorder traversal of an binary ordered tree

- Visit the left subtree, using postorder.
- Visit the right subtree, using postorder.
- Visit the root.

Postorder Traversal: A H E M Y T J



〖Example 1〗 In which order does a preorder, inorder or postorder traversal visit the vertices in the ordered rooted tree shown in the following figure?



- Preorder traversal : a, b, d, c, e, g, f, h, i
- Inorder traversal : d, b, a, e, g, c, h, f, i
- Postorder traversal : d, b, g, e, h, i, f, c, a

2. Infix, prefix, and postfix notation

Complicated expressions can be represented using ordered rooted trees, such as

- ✓ Compound propositions
- ✓ Combinations of sets
- ✓ Arithmetic expressions

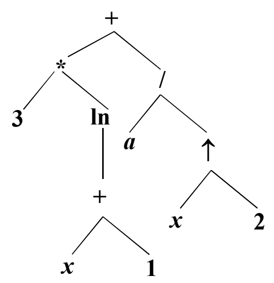
A Binary Expression Tree is . . .

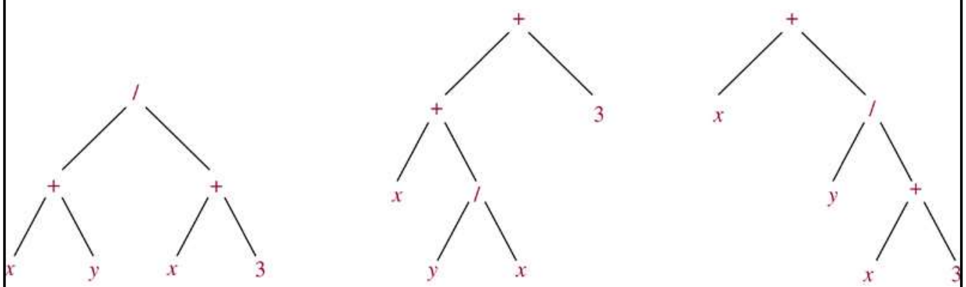
A special kind of binary tree in which:

1. **Each leaf node contains a single operand,**
2. **Each nonleaf node contains a single operator, and**
3. **The left and right subtrees of an operator node represent subexpressions that must be evaluated before applying the operator at the root of the subtree.**

〖Example 2〗 What is the ordered tree that represents the expression $3 * \ln(x+1) + a / x^2$?

Solution:

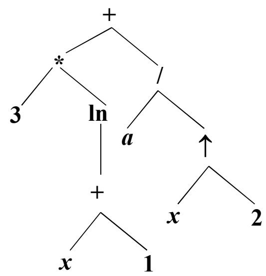




☸ Infix Form

The fully parenthesized expression obtained by an inorder traversal of the binary tree is said to be in *infix form*.

For example,

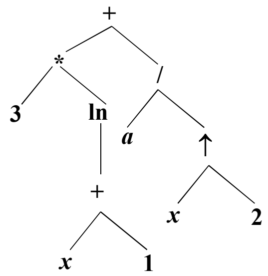


Infix form: $(3 * \ln(x+1)) + (a/(x \uparrow 2))$

❁ Prefix Form

The expression obtained by an preorder traversal of the binary tree is said to be in *prefix form (Polish notation).*

For example,

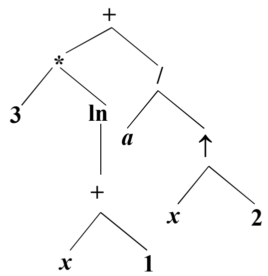


Prefix form: $+*3\ln + x1/a\uparrow x2$

☸ Postfix Form

The expression obtained by an postorder traversal of the binary tree is said to be in *postfix form (reverse Polish notation).*

For example,

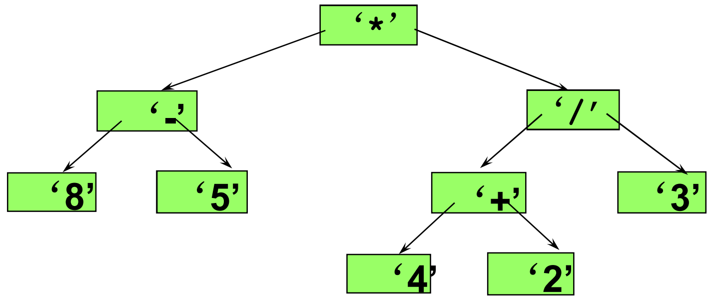


Postfix form: $3x1+\ln * ax2\uparrow /+$

❁ Evaluate the binary expression tree

- When a binary expression tree is used to represent an expression, the levels of the nodes in the tree indicate their relative precedence of evaluation.
- Operations at higher levels of the tree are evaluated later than those below them. The operation at the root is always the last operation performed.

Evaluate this binary expression tree

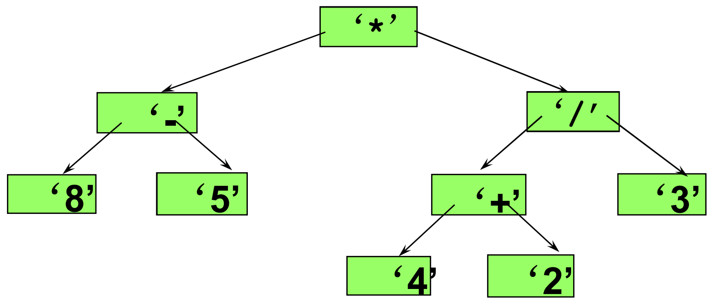


Prefix form: * - 8 5 / + 4 2 3

Note:

We can evaluate this expression from right to left.

Evaluate this binary expression tree



Postfix form: **8 5 - 4 2 + 3 / ***

Note:

We can evaluate this expression from left to right.

Homework:

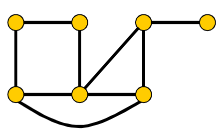
Sec. 11.3 8, 16

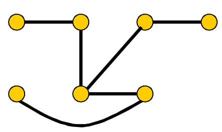
CHAPTER 11 Trees

❁ The definition of spanning tree

【Definition 1】 Let G be a simple graph. A spanning tree of G is a subgraph of G that is a tree containing every vertex of G.

For example,

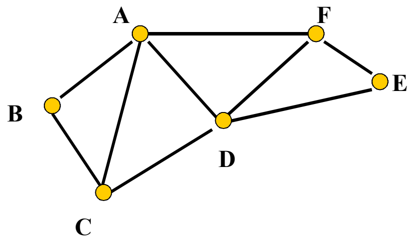


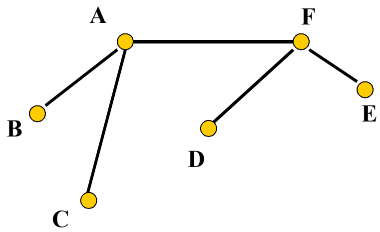


Problem:

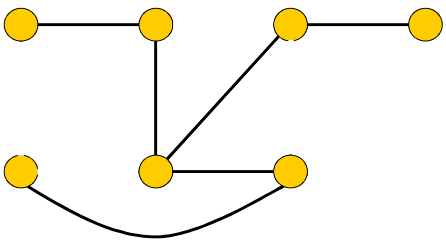
Why should we study the problem of spanning tree?

-- Consider the system of roads





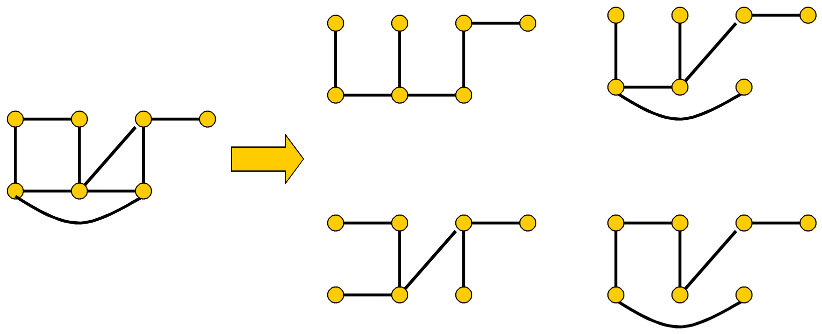
☸ Find A Spanning Tree of The Simple Graph



Method:

Find spanning trees by removing edges from simple circuits.

More than one spanning tree for a simple graph



【 Theorem 1 】 **A simple graph is connected if and only if it has a spanning tree.**

Proof:

First, suppose that a simple graph *G* has a spanning tree tree *T*.

T **contains every vertex of *G*.**

There is a path in *T* between any two of its vertices.

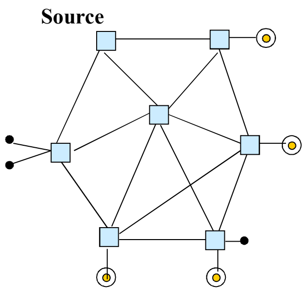
Since T is a subgraph of *G*, there is a path in *G* between any two of its vertices. Hence *G* is connected.

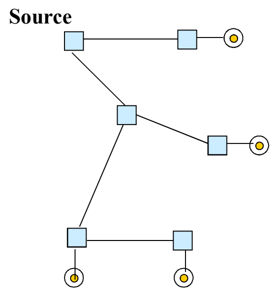
Second, suppose that *G* is connected.

We can find a spanning trees by removing edges from simple circuits of G.

❁ The Applications of Spanning Trees

〖Example 1〗 **IP Multicasting**.





☸ Algorithms for constructing spanning trees

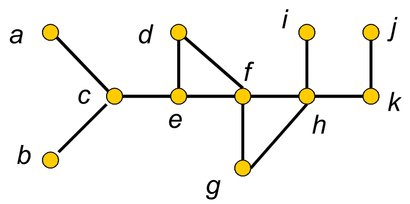
- **Theorem 1 gives an algorithm for finding spanning trees by removing edges from simple circuits.**
- **Instead of constructing spanning trees by removing edges, spanning trees can be built up by successively adding edges.**
- **Two algorithm:**
  - ✓ **Depth-first search**
  - ✓ **Breadth-first search**

❁ Depth-first search

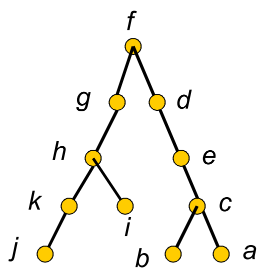
Depth-first search (also called *backtracking*) -- this procedure forms a rooted tree, and the underlying undirected graph is a spanning tree.

1. Arbitrarily choose a vertex of the graph as root.
2. Form a path starting at this vertex by successively adding edges, where each new edge is incident with the last vertex in the path and a vertex not already in the path.
3. Continue adding edges to this path as long as possible.
4. If the path goes through all vertices of the graph, the tree consisting of this path is a spanning tree.
5. If the path does not go through all vertices, more edges must be added. Move back to the next to last vertex in the path, if possible, form a new path starting at this vertex passing through vertices that were not already visited. If this cannot be done, move back another vertex in the path. Repeat this process.

〖Example 2〗 Use a depth-first search to find a spanning tree for the following graph.



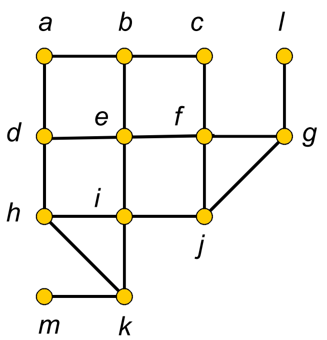
Solution:

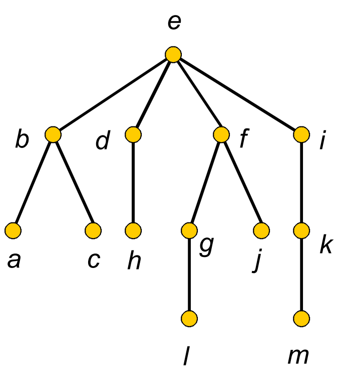


☸ Breadth-first search

1. Arbitrarily choose a vertex of the graph as a root, and add all edges incident to this vertex.
2. The new vertices added at this stage become the vertices at level 1 in the spanning tree. Arbitrarily order them.
3. For each vertex at level 1, visited in order, add each edge incident to this vertex to the tree as long as it does not produce a simple circuit. Arbitrarily order the children of each vertex at level 1. This produces the vertices at level 2 in the tree.
4. Follow the same procedure until all the vertices in the tree have been added.

〖Example 3〗 Use a breadth-first search to find a spanning tree for the following graph.

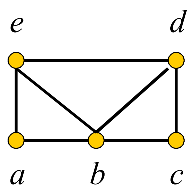


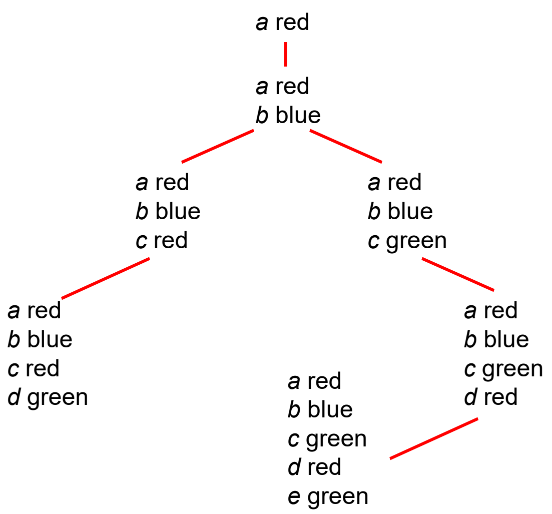


❁ Backtracking scheme

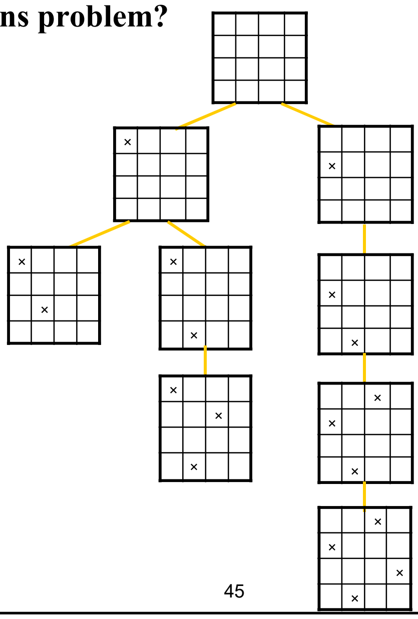
- There are problems that can be solved only by performing an exhaustive search of all possible solutions.
- One way to search systematically for a solution is to use a decision tree, where each internal vertex represents a decision and each leaf a possible solution.
- The method to find a solution via backtracking
- The applications of backtracking scheme
  - ✓ Graph Coloring
  - ✓ The n-Queens Problem
  - ✓ Sums of Subsets

〖Example 4〗 How can backtracking be used to decide whether the following graph can be colored using 3 colors?



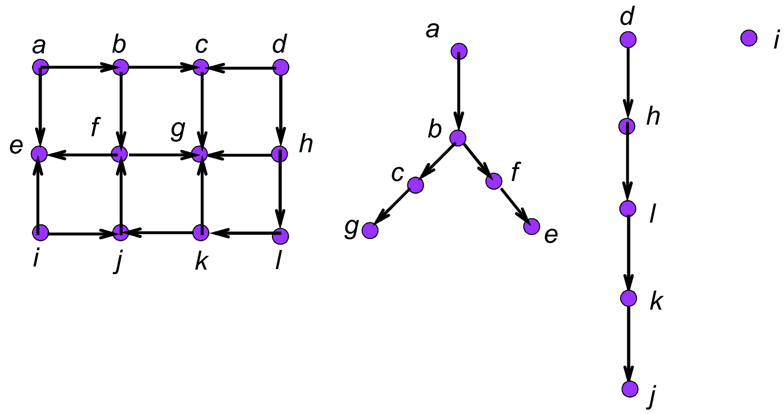


〖Example 5〗 How can backtracking be used to solve the *4*-queens problem?



❁ Depth-first Search in Directed Graphs

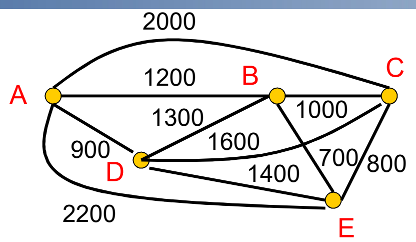
〖Example 6〗 What is the output of depth-first search given the following graph as input?



Homework:

Sec. 11.4 4, 14, 16(14), 29

CHAPTER 11 Trees



A weighted graph showing monthly lease costs for lines in a computer network.

Problem:

Which links should be made to ensure that there is a path between any two computer centers so that the total cost of the network is minimized?

We can solve this problem by finding a spanning tree so that the sum of the weights of the edges of the tree is minimized. Such a spanning tree is called a minimum spanning tree.

❁ the Concept of Minimum Spanning Trees

【Definition 1】 A *minimum spanning tree* in a connected weighted graph is a spanning tree that has the smallest possible sum of weights of its edges.

☸ Algorithms for minimum spanning trees

Two algorithms for constructing minimum spanning trees.

- ✓ Prim's algorithm
- ✓ Kruskal's algorithm

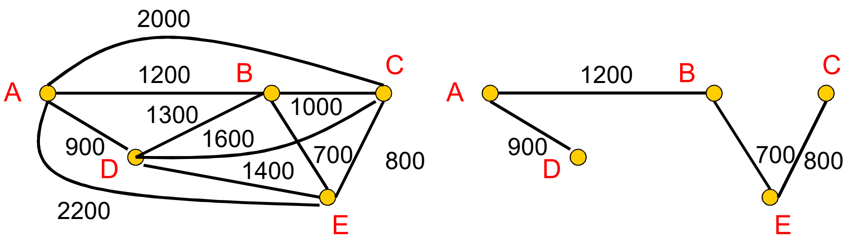
Both proceed by successively adding edges of smallest weight from those edges with a specified property that have not already been used.

These two algorithms are examples of *greedy algorithms*.

Prim's algorithm

Procedure *Prim* (G: weighted connected undirected graph with n vertices)
T:= a minimum-weight edge
for i:= 1 to n-2
begin
  e:= an edge of minimum weight incident to a vertex in
    T and not forming a simple circuit in T if added to T.
  T:= T with e added
end {T is a minimum spanning tree of G}

〖Example 1〗 Find a minimum spanning tree in the weighted graph.

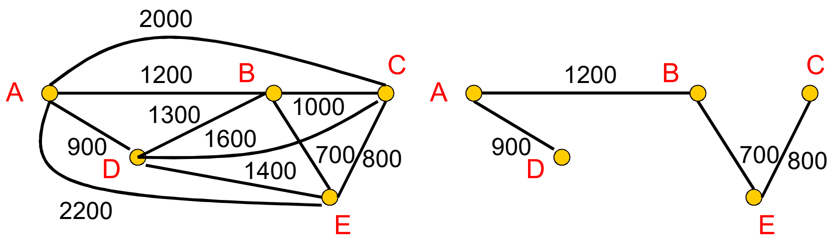


Choice	Edge	Cost
1	BE	700
2	EC	800
3	BA	1200
4	AD	900
	Total:	3600

☸ Kruskal's algorithm

procedure Kruskal (*G*: weighted connected undirected graph with n vertices)
T:= empty graph
for i:= 1 to *n*-1
begin
  e:= any edge in *G* with smallest weight that does not form a simple circuit when added to *T*
  T:= *T* with e added
end {*T* is a minimum spanning tree of *G*}

〖Example 2〗 **Find a minimum spanning tree in the weighted graph.**



Choice	Edge	Cost
1	BE	700
2	EC	800
3	AD	900
4	AB	1200
	Total:	3600

Homework:

Sec. 11.5 **3,7,12**