Discrete Mathematics

CS204: Spring, 2008

Jong C. Park

Computer Science Division, KAIST

Today's Topics

Introduction

Paths and Cycles

Hamiltonian Cycles and the Traveling Salesperson Problem

A Shortest-Path Algorithm

Representations of Graphs

Isomorphisms of Graphs

Planar Graphs

GRAPH THEORY

Planar Graphs

- Definition
  - A graph is planar if it can be drawn in the plane without its edges crossing.
  - If a connected, planar graph is drawn in the plane, the plane is divided into contiguous regions called faces. A face is characterized by the cycle that forms its boundary.
  - The equation below holds for any connected, planar graph.
    - $f = e - v + 2$, where f is the # of faces, e the # of edges, and v the # of vertices

Planar Graphs

- Examples
  - the graph of the Figure 8.7.2
  - Show that the graph $K_{3,3}$ of Figure 8.7.1 is not planar.
  - Show that the graph K_5 is not planar.
- Note
  - If a graph contains $K_{3,3}$ (or K_5) as a subgraph, it cannot be planar.

Planar Graphs

- Definition
  - If a graph G has a vertex v of degree 2 and edges (v, v_1) and (v, v_2) with $v_1 \neq v_2$, we say that the edges (v, v_1) and (v, v_2) are in series.
  - A series reduction consists of deleting the vertex v from the graph G and replacing the edges (v, v_1) and (v, v_2) by the edge (v_1, v_2).
  - The resulting graph G' is said to be obtained from G by a series reduction. By convention, G is said to be obtainable from itself by a series reduction.
- Example
  - Obtain a graph by a series reduction from the graph G of Figure 8.7.4.

Planar Graphs

- Definition
  - Graphs G_1 and G_2 are homeomorphic if G_1 and G_2 can be reduced to isomorphic graphs by performing a sequence of series reduction.
- Example
  - Determine if the graphs G_1 and G_2 of Figure 8.7.5 are homeomorphic.

Planar Graphs

- Theorem [Kuratowski's Theorem]
  - A graph G is planar if and only if G does not contain a subgraph homeomorphic to K_5 or $K_{3,3}$.
- Example
  - Show that the graph G of Figure 8.7.6 is not planar by using Kuratowski's Theorem.

Planar Graphs

- Theorem [Euler's Formula for Graphs]
  - If G is a connected, planar graph with e edges, v vertices, and f faces, then $f = e - v + 2$.
  - Proof.
    - Use induction on the number of edges.

Today's Topics

Introduction

Terminology and Characterization of Trees

Spanning Trees

Minimal Spanning Trees

Binary Trees

Tree Traversals

Decision Trees and the Minimum Time for Sorting

Isomorphisms of Trees

TREES

Introduction

- Definition
  - A (free) tree T is a simple graph satisfying the following:
    - If v and w are vertices in T, there is a unique simple path from v to w.
  - A rooted tree is a tree in which a particular vertex is designated the root.
- Example
  - the single-elimination tournament

Introduction

- Note
  - the level of a vertex *v* is the length of the simple path from the root to *v*.
  - The height of a rooted tree is the maximum level number that occurs.

Introduction

- Examples
  - the rooted tree of Figure 9.1.4
  - the trees of Figure 9.1.5
  - an administrative organizational chart
  - Computer File Systems
  - Hierarchical Definition Trees

Introduction

- Example
  - Huffman Codes
    - represent characters by variable-length bit strings
    - Use Figure 9.1.10 to decode the string 01010111.

Introduction

- Example
  - Construct an optimal Huffman code using Table 9.1.2.

Algorithm 9.1.9: Constructing an Optimal Huffman Code

Input: A sequence of n frequencies, $n \geq 2$

Output: A rooted tree that defines an optimal Huffman code

huffman(f, n) {
  if $(n == 2)$ {
    let f_1 and f_2 denote the frequencies
    let T be as in Figure 9.1.11
    return T
  }
  let f_i and f_j denote the smallest frequencies
  replace f_i and f_j in the list f by $f_i + f_j$
  $T' = huffman(f, n - 1)$
  replace a vertex in T' labeled $f_i + f_j$ by the tree shown
    in Figure 9.1.12 to obtain the tree T
  return T
}

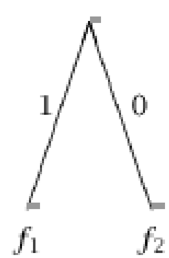

Figure 9.1.11

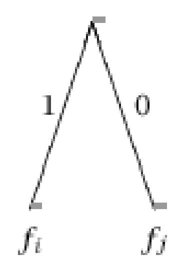

Figure 9.1.12

Terminology and Characterization of Trees

- Definition
  - Let T be a tree with root v_0.
  - Suppose that x, y, and z are vertices in T and that $(v_0, v_1, \ldots, v_n)$ is a simple path in T.
  - Then

    (a) v_{n-1} is the parent of v_n.

    (b) $v_0, \ldots, v_{n-1}$ are ancestors of v_n.

    (c) v_n is a child of v_{n-1}

    (d) If x is an ancestor of y, y is a descendant of x.

    (e) If x and y are children of z, x and y are siblings.

Terminology and Characterization of Trees

(f) If x has no children, x is a terminal vertex (or a leaf).

(g) If x is not a terminal vertex, x is an internal (or branch) vertex.

(h) The subtree of T rooted at x is the graph with vertex set V and edge set E, where V is x together with the descendants of x and

$E = \{e \mid e$ is an edge on a simple path from x to some vertex in $V\}$.

- Note
  - A graph with no cycles is called an acyclic graph.

Terminology and Characterization of Trees

- Theorem
  - Let T be a graph with n vertices. The following are equivalent.
    (a) T is a tree.
    (b) T is connected and acyclic.
    (c) T is connected and has $n - 1$ edges.
    (d) T is acyclic and has $n - 1$ edges.
  - Proof.
    - Exercise.

Spanning Trees

- Definition
  - A tree T is a spanning tree of a graph G if T is a subgraph of G that contains all of the vertices of G.
- Examples
  - Find a spanning tree of the graph G of Figure 9.3.1.
  - Find an alternative spanning tree of the graph G of Figure 9.3.1.

Spanning Trees

- Theorem
  - A graph G has a spanning tree if and only if G is connected.
  - Proof sketch.
    - ➔: Use the notion of a path.
    - 🡐: Use the notion of cyclicity

Spanning Trees

Algorithm 9.3.6: Breadth-First Search for a Spanning Tree

Input: A connected graph G with vertices ordered $v_1, v_2, \ldots v_n$

Output: A spanning tree T

$bfs(V, E)$ {
  // V = vertices ordered $v_1, \ldots, v_n$; E = edges
  // V' = vertices of spanning tree T;
  // E' = edges of spanning tree T
  // v_1 is the root of the spanning tree
  // S is an ordered list
  $S = (v_1)$
  $V' = \{v_1\}$
  $E' = \varnothing$
  while (true) {
    for each $x \in S$, in order,
      for each $y \in V - V'$, in order,
        if $((x, y)$ is an edge)
          add edge (x, y) to E' and y to V'
    if (no edges were added)
      return T
    S = children of S ordered consistently with the
        original vertex ordering
  }
}

Spanning Trees

Algorithm 9.3.7: Depth-First Search for a Spanning Tree

Input: A connected graph G with vertices ordered $v_1, v_2, \ldots v_n$

Output: A spanning tree T

```
dfs(V, E) {
  // V' = vertices of spanning tree T;
  // E' = edges of spanning tree T
  // v1 is the root of the spanning tree
  V' = {v1}
  E' = ∅
  w = v1
  while (true) {
    while (there is an edge (w, v) that when added to T
           does not create a cycle in T) {
      choose the edge (w, vk) with minimum k that when
        added to T does not create a cycle in T
      add (w, vk) to E'
      add vk to V'
      w = vk
    }
    if (w == v1)
      return T
    w = parent of w in T // backtrack
  }
}
```

Spanning Trees

Algorithm 9.3.10: Solving the Four-Queens Problem Using Backtracking

Input: An array *row* of size 4

Output: true, if there is a solution
false, if there is no solution
[If there is a solution, the kth queen is in column k, row $row(k)$.]

```
four_queens(row) {
  k = 1 // start in column 1

  // start in row 1
  // since row(k) is incremented prior to use,
  // set row(1) to 0
  row(1) = 0
  while (k > 0) {
    row(k) = row(k) + 1
    // look for a legal move in column k
    while (row(k) ≤ 4 ∧ column k, row(k) conflicts)
      // try next row
      row(k) = row(k) + 1
```

Algorithm 9.3.10 (continued)

```
    if (row(k) ≤ 4)
      if (k == 4)
        return true
      else { // next column
        k = k + 1
        row(k) = 0
      }
    else // backtrack to previous column
      k = k − 1
  }
  return false // no solution
}
```

Minimal Spanning Trees

- Definition
  - Let G be a weighted graph. A minimal spanning tree of G is a spanning tree of G with minimum weight.
- Example
  - Find two spanning trees for graph G of Figure 9.4.1 and compare their weights.

Minimal Spanning Trees

Algorithm 9.4.3: Prim's Algorithm

Input: A connected, weighted graph G with vertices $1, \ldots, n$ and start vertex s. If (i, j) is an edge, $w(i, j)$ is equal to the weight of (i, j); if (i, j) is not an edge, $w(i, j)$ is equal to ∞ (a value greater than any actual weight).

Output: The set of edges E is a minimal spanning tree (mst)

```
prim(w, n, s) {
   // v(i) = 1 if vertex i has been added to mst
   // v(i) = 0 if vertex i has not been added to mst
   for i = 1 to n
      v(i) = 0
   // add start vertex to mst
   v(s) = 1
   // begin with an empty edge set
   E = ∅
```

Algorithm 9.4.3 (continued)

```
   // put n − 1 edges in the minimal spanning tree
   for i = 1 to n − 1 {
      // add edge of minimum weight with one
      // vertex in mst and one vertex not in mst
      min = ∞
      for j = 1 to n
         if (v(j) == 1) // if j is a vertex in mst
            for k = 1 to n
               if (v(k) == 0 ∧ w(j, k) < min) {
                  add_vertex = k
                  e = (j, k)
                  min = w(j, k)
               }
      // put vertex and edge in mst
      v(add_vertex) = 1
      E = E ∪ {e}
   }
   return E
}
```

Minimal Spanning Trees

- Theorem
  - Prim's Algorithm is correct; that is, at the termination of Algorithm 9.4.3, T is a minimal spanning tree.
- Kruskal's algorithm
  - Sort edges.

Summary

- Terminology and Characterization of Trees
- Spanning Trees
- Minimal Spanning Trees
- Binary Trees
- Tree Traversals
- Decision Trees and the Minimum Time for Sorting
- Isomorphisms of Trees