Traversing and Search

Lecture 19
CS 312

Objectives

• Use search and traversal to analyze
trees and graphs

Traversing Methods

Examine every vertex in the search tree

Search Methods

May not examine all vertices.

Traversing

• Produces a linear order for the information in
a tree
• L = left, D = print data, R = left
• Six possible combinations:
  – LDR, LRD, DLR, DRL, RDL, RLD
• Always traverse left before right, leaves us
with:
  – LDR = inorder
  – LRD = postorder
  – DLR = preorder

Inorder

Inorder
Inorder

```c
void InOrder(struct treenode *t)
{
    if (t) {
        InOrder(t->lchild);
        Visit(t);
        InOrder(t->rchild);
    }
}
```

Postorder

```c
void PostOrder(struct treenode *t)
{
    if (t) {
        PostOrder(t->lchild);
        PostOrder(t->rchild);
        Visit(t);
    }
}
```

Preorder

```c
void PreOrder(struct treenode *t)
{
    if (t) {
        Visit(t);
        PreOrder(t->lchild);
        PreOrder(t->rchild);
    }
}
```
**Preorder**

```c
void PreOrder(struct treenode *t)
{
    if (t) {
        Visit(t);
        PreOrder(t->lchild);
        PreOrder(t->rchild);
    }
}
```

**What’s the complexity of this algorithm?**

**Breadth First Search**

Visit node 1, then visit all of node 1’s children. Then visit all of node 2’s children. Basic idea: keep a queue of nodes to explore.

Visit node 1, then visit all of node 1’s neighbors. Then visit all of node 2’s neighbors. Basic idea: keep a queue of nodes to explore.

Pop the queue and explore that node’s children.

Pop the queue and explore that node’s neighbors. (except the neighbors that have already been visited)
Breadth First Search

void BFS(int v)
{
    int u = v; int w;
    Queue q(SIZE);
    visited[v] = 1;
    do {
        for all vertices w adjacent from u {
            if (visited[w]==0) {
                q.Enqueue(w); // w is unexplored
                visited[w] = 1;
            }
        }
        if (q.Empty()) return;
        q.Dequeue(u);
    } while (1);
}

Breadth First Search

What is the time complexity of this algorithm?

void BFS(int v)
{
    int u = v;
    Queue q(SIZE);
    visited[v] = 1;
    do {
        for all vertices w adjacent from u {
            if (visited[w]==0) {
                q.Enqueue(w); // w is unexplored
                visited[w] = 1;
            }
        }
        if (q.Empty()) return;
        q.Dequeue(u);
    } while (1);
}

Breadth First Search

What is the complexity of this algorithm?

void BFS(int v)
{
    int u = v;
    Queue q(SIZE);
    visited[v] = 1;
    do {
        for all vertices w adjacent from u {
            if (visited[w]==0) {
                q.Enqueue(w); // w is unexplored
            }
        }
        if (q.Empty()) return;
        q.Dequeue(u);
    } while (1);
}

Breadth First Search

Determining adjacency depends on the representation of the graph.

Represent this graph using adjacency lists or a matrix.
Breadth First Search

What is the complexity of this algorithm?

```c
void BFS(int v) {
    int u = v;
    Queue q(SIZE);
    visited[v] = 1;
    do {
        for all vertices w adjacent from u {
            if (visited[w]==0){
                q.AddQ(w); // w is unexplored
            }
        }
        if (q.empty()) return;
        q.DeleteQ(u);
    } while (!)
}
```

Determining adjacency depends on the representation of the graph.

Adjacency List vs. Adjacency Matrix

- BFS: \( \theta(n + e) \) time using adjacency lists
- BFS: \( \theta(n^2) \) time using adjacency matrix
- BFS: \( \theta(n) \) space either way
  - when does it take \( \theta(n) \) space?
  - why?

- Why is BFS slower than preorder?

Depth First Search

- Same a BFS, except in different order
- Exploration of a vertex is suspended as soon as a new vertex is reached

Depth First Search

Start at node 1, visit its first neighbor.

Visit node 2's first neighbor.

Visit node 4's first neighbor.
Questions

- Does BFS resemble postorder, preorder, or inorder?
- Does DFS resemble postorder, preorder, or inorder?
- What is the difference?
- When would you use each?

One more question

Give an example of an \( n \)-vertex graph for which the depth of recursion of DFS starting from a particular vertex \( v \) is \( n-1 \) whereas the queue of BFS has at most one vertex at any given time if BFS is started from the same vertex \( v \)?