Graph Searching

CSE 373
Data Structures
Lecture 20
Graph Searching

• Find Properties of Graphs
  › Connected components
  › Bipartite structure
  › Biconnected components

• Applications
  › Alternating paths for matching
  › Garbage collection – used in Java run time system
  › Finding dead code
  › Finding the web graph – used by Google and others
Depth First Search Algorithm

• Recursive marking algorithm
• Initially every vertex is unmarked

DFS(i: vertex)
mark i;
for each j adjacent to i do
  if j is unmarked then DFS(j)
end{DFS}

Marks all vertices reachable from i
Example of Depth First Search
Example Step 2

DFS(1)
DFS(2)
Example Step 3

DFS(1)
DFS(2)
DFS(7)
Example Step 4
Example Step 5
Example Step 6

DFS(1)  
DFS(2)  
DFS(7)  
DFS(5)  
DFS(4)  
DFS(3)
Example Step 7

DFS(1)
DFS(2)
DFS(7)
DFS(5)
DFS(4)
DFS(3)
Example Step 8

DFS(1)
DFS(2)
DFS(7)
DFS(5)
DFS(4)
Example Step 9

DFS(1)
DFS(2)
DFS(7)
DFS(5)
DFS(4)
Example Step 10
Example Step 11

DFS(1)
DFS(2)
DFS(7)
DFS(5)
DFS(6)
Example Step 12

DFS(1)
DFS(2)
DFS(7)
DFS(5)
DFS(6)
Example Step 13

DFS(1)
DFS(2)
DFS(3)
DFS(4)
Example Step 14

DFS(1)
DFS(2)
DFS(7)
Example Step 15

DFS(1)
DFS(2)
Example Step 16

DFS(1)
Adjacency List Implementation

• Adjacency lists

```
M: 0 0 0 0 0 0 0
G:
  1 - 2 - 5 - 6
  2 - 3 - 1 - 7
  3 - 2 - 4
  4 - 3 - 7 - 5
  5 - 6 - 1 - 7 - 4
  6 - 1 - 5
  7 - 4 - 5 - 2
```

Index  next
Connected Components

3 connected components
Connected Components

3 connected components are labeled
Depth-first Search for Labeling Connected components

Main {
  i : integer
  for i = 1 to n do M[i] := 0;
  label := 1;
  for i = 1 to n do
    if M[i] = 0 then DFS(G,M,i,label);
    label := label + 1;
}

DFS(G[]: node ptr array, M[]: int array, i,label: int) {
  v : node pointer;
  M[i] := label;
  v := G[i];
  while v ≠ null do
    if M[v.index] = 0 then DFS(G,M,v.index,label);
    v := v.next;
}
Spanning Tree

Spanning tree – no cycles and connects all vertices
Exercise

• Design a depth-first algorithm to output a spanning tree of a connected graph.

Main {
    i : integer
    for i = 1 to n M[i] := 0;
    T := EmptySet;
    STree(1);
}

STree(G[], node ptr array, M[], int array, i : int) : {
    ???
}

Performance DFS

- $n$ vertices and $m$ edges
- Storage complexity $O(n + m)$
- Time complexity $O(n + m)$
- Linear Time!
Breadth First Search 1

- Uses a queue to order search

```
Queue = 1
```
Breadth First Search 2

Queue = 2, 6, 5

Mark while on queue to avoid putting in queue more than once
Breadth First Search 3

Queue = 6,5,7,3
Breadth First Search 4

Queue = 5,7,3
Breadth First Search 5

Queue = 7, 3, 4
Breadth First Search 6

Queue = 3,4
Breadth First Search 7

Queue = 4
Breadth First Search 8

Queue =
Breadth-First Search

BFS
Initialize Q to be empty;
Enqueue(Q,1) and mark 1;
while Q is not empty do
  i := Dequeue(Q);
  for each j adjacent to i do
    if j is not marked then
      Enqueue(Q,j) and mark j;
  end {BFS}

Depth-First vs Breadth-First

- Depth-First
  - Stack or recursion
  - Many applications

- Breadth-First
  - Queue (recursion no help)
  - Can be used to find shortest paths from the start vertex
  - Can be used to find short alternating paths for matching
Bipartite Matching Algorithm

set M to be the empty set initially
repeat
    find an alternating path $x_1, x_2, \ldots, x_{2n}$
    // $(x_i, x_{i+1})$ in $E - M$ if $i$ is odd and $(x_i, x_{i+1})$ in $M$ if $i$ is even
    neither $x_1$ nor $x_{2n}$ are matched //
    delete $(x_i, x_{i+1})$ from $M$ if $i$ is even
    add $(x_i, x_{i+1})$ to $M$ if $i$ is odd
until no alternating path can be found

if $M$ has every vertex in $U$ then $M$ is a matching
if $M$ does not have some vertex then there is complete
matching, but $M$ is a maximum size matching
Partial Matching
Alternating Path
Finding an Alternating Path 1

- Direct the edges
  - U to V if edge in E – M
  - V to U if edge in M
Finding an Alternating Path 2

- For each $u$ in $U$ which is not matched do a breadth-first search until an unmatched $v$ in $V$ is found.
  - If no unmatched $v$ is found then no alternating path from $u$
  - Each visited node is marked and not visited again

Can also use depth-first search, but longer longer alternating paths would be found.
Running Time of Maximum Matching

- Parameters
  - Number of edges $m$
  - Number of vertices $n$
- Iterations of find alternating path – $n$
  - Each iteration increases the matching size by 1
- Time to find an alternating path
  - Direct the edges $O(m)$ (assuming $m \geq n$)
  - Breadth-first search $O(m)$ (assuming $m \geq n$)
- Total time $O(nm)$ (Comment: $nm \leq n^3$)
- Can be solved in $O(n^{2.5})$ by clever tricks.
Exercise Solution

• Design a depth-first algorithm to output a spanning tree of a connected graph.

STree(G[]: node ptr array, M[]: int array i: integer) : {
    v : node pointer;
    M[i] := 1;
    v := G[i];
    while v ≠ null do
        if M[v.index] = 0 then
            Insert({i,v.index}, T); STree(v.index);
            v := v.next
    }