G205 Fundamentals of Computer Engineering

CLASSES 14/15, Wed. Oct. 27 03

Stefano Basagni

Fall 2003

M-W, 9:50am-11:30am, 410 Ell

Elementary Graph Algorithms

- Graph G=(V,E)
  - Finite set of vertices V, |V|=n
  - Finite set of edges E joining pairs of nodes, |E|=m
- G can be
  - Directed: $E \subseteq V \times V$, $(a,b) \neq (b,a)$, $a,b \in V$
  - Undirected: $E=\{\{a,b\}: a,b \in V\}$
- Allows natural graphical representation

Graph Representation

- Two common ways to represent a graph
  - Adjacency list
  - Adjacency matrix
- Running time is expressed in term of both |V|=n and |E|=m
- In asymptotic notation we will drop the cardinality: O(V+E)=O(n+m)

Adjacency Lists

- Array Adj of n lists, one per vertex
- u's list = all vertices v such that $(u, v) \in E$
- u and v are said to be neighbors
- Works for directed and undirected graphs
- Edge weights $w: E \rightarrow \mathbf{R}$ can be listed
- **Space:** $\Theta(V + E)$
- **Time:** to list all neighbors of u: $\Theta(deg(u))$
- **Time:** to check if $(u,v) \in E$: $O(deg(u))$

Adjacency Matrix

- G is represented by a n x n matrix $A = (a_{i,j})$
  - $a_{i,j} = 1$ if $(i,j) \in E$
  - $a_{i,j} = 0$ if $(i,j) \in E$
- **Space:** $\Theta(n^2)$
- **Time:** to list all vertices adjacent to u: $\Theta(V)$
- **Time:** to determine if $(u,v) \in E$: $\Theta(1)$
- Can store weights instead of bits for weighted graph

Breadth-First Search, BFS 1

- **Input**: Graph $G = (V, E)$, directed or undirected, and **source vertex** $s \in V$
- **Output:** $d[v]$ = distance (smallest # of edges) from s to v, for all $v \in V$
- Also $\pi[v]=u$ such that (u,v) is last edge on shortest path $s \rightsquigarrow v$
  - u is v's **predecessor**
  - Set of edges $\{(\pi[v], v) : v = s\}$ forms a tree

BFS 2

- Compute only d[v], not π[v]
- Omitting colors of vertices
- **Idea:** Send a wave out from s
  - First hits all vertices 1 edge from s
  - From there, hits all vertices 2 edges from s
  - Etc.
- Use FIFO queue Q to maintain "wavefront"
  - $v \in Q$ if and only if wave has hit v but has not come out of v yet

BFS 3

```
BFS(V,E,s)
  for each u∈V\{s} do d[u]=∞
  d[s]=0; Q=0
  ENQUEUE(Q, s)
  while Q≠0 do
    u=DEQUEUE(Q)
    for each v ∈ Adj[u] do
      if d[v]=∞ then
                    d[v]=d[u]+1
                    ENQUEUE(Q, v)
```

BFS, Analysis

- Time = O(V + E)
  - O(V) because every vertex enqueued at most once
  - O(E) because every vertex dequeued at most once and we examine (u,v) only when u is dequeued
  - Every edge examined at most once if directed, at most twice if undirected

Depth-First Search, DFS 1

- **Input:** G=(V,E), directed or undirected. No source vertex given!
- **Output:** 2 **timestamps** on each vertex:
  - d[v] = **discovery time**
  - f[v] = **finishing time**
  - (These will be useful for other algorithms later on)
- Can also compute π[v]
- Will methodically explore every edge
  - Start over from different vertices as necessary

DFS 2

- As soon as a vertex is discovered, explore from it (no queue like BFS)
- As DFS progresses, every vertex has a **color**:
  - WHITE = undiscovered
  - GRAY = discovered, not finished
  - BLACK = finished (found everything reachable)
- Discovery and finish times:
  - Unique integers from 1 to 2|V|
  - For all v, $d[v] < f[v]$
  - In other words, $1 \le d[v] < f[v] \le 2|V|$

DFS 3

```
DFS(V,E)
  for each u ∈ V
  do color[u] = WHITE
  time = 0
  for each u ∈ V do
    if color[u] = WHITE
      then DFS-VISIT(u)
```

DFS 4

```
DFS-VISIT(u)
  color[u]=GRAY                          // discover u
  time=time+1
  d[u]=time
  for each v ∈ Adj[u] do                 // explore (u,v)
    if color[v] = WHITE
      then DFS-VISIT(v)
  color[u]=BLACK
  time=time+1
  f[u]=time                              // finish u
```

DFS Analysis

- Time = $\Theta(V + E)$
- Similar to BFS analysis
- Θ, not just O, since guaranteed to examine every vertex and edge
- DFS forms a **depth-first forest** comprised of > 1 **depth-first trees**.
- Each tree is made of edges (u,v) such that u is gray and v is white when (u,v) is explored

DFS, Parenthesis Theorem

- For all u and v exactly one of the following holds:
  1. $d[u]<f[u]<d[v]<f[v]$ or $d[v]<f[v]<d[u]<f[u]$ and u and v are not descendant of each other
  2. $d[u]<d[v]<f[v]<f[u]$ and v is a descendant of u
  3. $d[v]<d[u]<f[u]<f[v]$ and u is a descendant of v
- So $d[u]<d[v]<f[u]<f[v]$ cannot happen
- **Corollary:** v is a proper descendant of u if and only if $d[u]<d[v]<f[v]<f[u]$

DFS, White Path Theorem

- v is a descendant of u if and only if at time d[u], there is a path u $\rightsquigarrow$ v consisting of only white vertices

  (Except for u, which was just colored gray)

Classification of Edges

- **Tree edge:** in the depth-first forest. Found by exploring (u,v)
- **Back edge:** (u,v), u is a descendant of v
- **Forward edge:** (u,v), where v is a descendant of u, but not a tree edge
- **Cross edge:** any other edge
- **Theorem:** In DFS of an undirected graph, we get only tree and back edges. No forward or cross edges

Assignments

- Textbook, Chapter 22, pages 524—549
- Updated information on the class web page:

  www.ece.neu.edu/courses/eceg205/2003fa