Theory of Computing

Lecture 4

MAS 714

Hartmut Klauck

Traversing Graphs

- Common to BFS/DFS:
  - Use a datastructure with the following operations:
    - Insert a vertex
    - Remove a vertex
  - Maintain an active vertex (start with s)
  - Maintain an array of vertices already visited
  - Then:
    - Insert all (unvisited) neighbors of the active vertex, mark it as visited
    - Remove a vertex v and make it active

The Datastructure

- We distinguish by the rule that determines the next active vertex

- Alternative 1: queue
  - FIFO (first in first out)
- Alternative 2: stack
  - LIFO (last in first out)

Datastructures: Queue

- A queue is a linked list together with two operations
  - Insert: Insert an element at the rear of the queue
  - Remove: Remove the front element of the queue
- Implementations is as a linked list
  - We need a pointer to the rear and a pointer to the front

BFS

- Every time we put a vertex v into the queue, we also remember the predecessor of v, i.e., the vertex π(v) as who's neighbor v was queued
- And remember d(v), which will be the distance of v from s

BFS

- Procedure:
  - For all v:
    - visit(v)=0, d(v)=∞,π(v)=NIL
  - d(s)=0
  - Enter s into the queue Q
  - While Q is not empty
    - Remove v from Q
    - visit(v)=1, enter all neighbors w of v with visit(w)=0 into Q and set π(w)=v, d(w)=d(v)+1

BFS

- Clearly the running time of BFS is O(m+n)
  - n to go over all vertices
  - m to check all neighbors
  - Each queue operation takes constant time
- BFS runs in linear time

BFS tree

- Consider all edges (π(v), v)
- Claim: These edges form a tree

- This tree is called the BFS tree of G (from s)
  - vertices not reachable from s are not in the tree

BFS tree

- Proof (of Claim):
  - Each visited vertex has 1 predecessor (except s)
  - V_s is the set of visited vertices
  - Graph is directed
  - There are $|V_s-1|$ edges
  - Hence the edges form a tree

Shortest Paths

- BFS can be used to compute shortest paths
  - in unweighted graphs
- Definition:
  - Graph G, vertex s
  - δ_G(s,v) is the minimum number of edges in any path from s to v
    - No path: ∞

Shortest Paths

- Lemma:
  - Let (u,v) be an edge
  - Then: $\delta(s,v) \leq \delta(s,u)+1$

- Proof: v is reachable $\Rightarrow$ u is reachable
  - Shortest path from s to u cannot be longer than shortest path from s to v plus one edge
    - Triangle inequality

Shortest Paths

- Lemma:
  - The values d(v) computed by BFS are the δ(s,v)
- Proof:
  - First, show that d(v)$\geq\delta$(s,v)
  - Induction over the number of steps
    - Surely true in the beginning
    - Suppose true, when we queue a vertex
    - Then also true for the neighbors

Shortest Paths

- Now we show that $d(v) \leq \delta(s,v)$
- **Observation:** For all vertices in Q, d(v) is only different by 1 (and Q has increasing d(v) by position in Q)
- Now assume that $d(v) > \delta(s,v)$ for some v
  - Choose some v with minimum $\delta(s,v) < d(v)$
- v is reachable from s (otherwise $\delta(s,v) = \infty$)
- Consider the predecessor u of v on a shortest path s to v
  - $\delta(s,v) = \delta(s,u) + 1$

Shortest Paths

- $d(v)>\delta(s,v)=\delta(s,u)+1=d(u)+1$
  - Because v is minimal “violator”
- At some point u is removed from the queue
  - If v is unvisited and not in the queue, then $d(v)=d(u)+1$
  - If v is visited already then by our observation $d(v)\leq d(u)$
  - If v is unvisited, and in the queue, then $d(v)\leq d(u)+1$ (observation)
- Contradiction in any case

Shortest Paths

- Lemma: The BFS tree is a shortest path tree
- Proof:
  - We already saw it is a tree
  - (π(v),v) is always a graph edge
  - d(v) is the depth in the BFS tree
    - Induction: true for s
    - True for level d $\Rightarrow$ when v is added in level d+1 then d(v)=d+1
  - Hence a path from the root s following tree edges is a shortest path (has length d(v))

BFS

- Runs in time O(m+n) on adjacency lists
- Visits every vertex reachable from s
- Can be used to compute shortest paths from s to all other vertices in directed, unweighted graphs

Depth First Search

- If we use a *stack* as datastructure we get Depth First Search (DFS)
- Typically, DFS will maintain some extra information:
  - Time when v is put on the stack
  - Time, when all neighbors of v have been examined
- This information is useful for applications

Datastructure: Stack

- A stack is a linked list together with two operations
  - push(x,S): Insert element x at the front of the list S
  - pop(S): Remove the front element of the list S
- Implementation:
  - Need to maintain only the pointer to the front of the stack
  - Useful to also have
    - peek(S): Find the front element but don’t remove

Digression: Recursion and Stacks

- Our model of Random Access Machines does not directly allow recursion
  - Neither does any real hardware
- Compilers will “roll out” recursive calls
  - Put all local variables of the calling procedure in a safe place
  - Execute the call
  - Return the result and restore the local variables

Recursion

- The best datastructure for this is a stack
  - Push all local variables to the stack
  - LIFO functionality is exactly the right thing
- Example: Recursion tree of Quicksort

DFS

- Procedure:
  1. For all v:
     - $\pi(v)$=NIL, d(v)=0, f(v)=0
  2. Enter s into the stack S, set TIME=1, d(s)=TIME
  3. While S is not empty
     a) v=peek(S)
     b) Find the first neighbor w of v with d(w)=0:
        – push(w,S) , $\pi(w)$=v, TIME=TIME+1, d(w)=TIME
     c) If there is no such w: pop(S), TIME=TIME+1, f(v)=TIME

DFS

- The array d(v) holds the time we first visit a vertex.
- The array f(v) holds the time when all neighbors of v have been processed
- “discovery” and “finish”
- In particular, when d(v)=0 then v has not been found yet

Simple Observations

- Vertices are given d(v) numbers between 1 and 2n
- Each vertex is put on the stack once, and receives the f(v) number once all neighbors are visited
- Running time is O(n+m)

Edge labelleing

- We will classify edges
  - The edges in (π(v),v) form trees: tree edges
  - We can label all other edges as
    - back edges
    - cross edges
    - forward edges

Edge classification

- **Lemma:** the edges (π(v),v) form a tree
- **Definition:**
  - Edges going down along a path in a tree (but not tree edge) are *forward edges*
  - Edges going up along a path in a tree are *back edges*
  - Edges across paths/tree are *cross edges*
- A vertex v is a *descendant* of u if there is a path of tree edges from u to v
- Observation: descendants are discovered after their "ancestors" but finish before them

Example: edge labeling

- Tree edges, Back edges, Forward edges, Cross edges