CS 461, Lecture 18

Jared Saia
University of New Mexico

Today's Outline

- Breadth First and Depth First Search
- Single Source Shortest Path

Traversing a Graph

- Suppose we want to visit every node in a connected graph (represented either explicitly or implicitly)
- The simplest way to do this is an algorithm called *depth-first search*
- We can write this algorithm recursively or iteratively - it's the same both ways, the iterative version just makes the stack explicit
- Both versions of the algorithm are initially passed a *source* vertex v

Recursive DFS

```
RecursiveDFS(v){
  if (v is unmarked){
    mark v;
    for each edge (v,w){
      RecursiveDFS(w);
    }
  }
}
```

Iterative DFS

```
IterativeDFS(s){
  Push(s);
  while (stack not empty){
    v = Pop();
    if (v is unmarked){
      mark v;
      for each edge (v,w){
        Push(w);
      }
    }
  }
}
```


Generic Traverse

- DFS is one instance of a general family of graph traversal algorithms
- This generic graph traversal algorithm stores a set of candidate edges in a data structure we'll call a "bag"
- A "bag" is just something we can put stuff into and later take stuff out of - stacks, queues and heaps are all examples of bags.

Generic Traverse

```
Traverse(s){
  put (nil,s) in bag;
  while (the bag is not empty){
    take some edge (p,v) from the bag
    if (v is unmarked)
      mark v;
      parent(v) = p;
      for each edge (v,w) incident to v{
        put (v,w) into the bag;
      }
    }
  }
}
```


Analysis

- Notice that we're keeping *edges* in the bag instead of vertices
- This is because we want to remember when we visit vertex v for teh first time, which previously-visited vertex p put v into the bag
- This vertex p is called the *parent* of v

Lemma

- Traverse(s) marks each vertex in a connected graph exactly once, and the set of edges $(v, parent(v))$, with parent(v) not nil, form a spanning tree of the graph.

Proof

- It's obvious that no node is marked more than once
- We next show that each vertex is marked at least once.
- Let $v \neq s$ be a vertex and let $s \to \cdots \to u \to v$ be the path from s to v with the minimum number of edges. (Since the graph is connected such a path always exists)
- If the algorithm marks u, then it must put (u, v) in the bag, so it must later take (u, v) out of the bag, at which point v must be marked
- Thus by induction on the shortest-path distance from s, the algorithm marks every vertex in the graph

Proof

- Call an edge $(v, parent(v))$ with $parent(v) \neq nil$ a *parent edge*.
- Note that since every node is marked, every node has a parent edge except for s.
- It now remains to be shown that the parent edges form a spanning tree of the graph

Proof

- For any node $v \neq s$, the path of parent edges $v \to parent(v) \to parent(parent(v)) \to \cdots$ eventually leads back to s, so the set of parent edges form a connected graph.
- Since every node except s has a unique parent edge, the total number of parent edges is exactly one less than the total number of vertices. (i.e. if there are n nodes, then there are $n - 1$ edges)
- Thus the parent edges form a spanning tree (we'll show this in the in-class exercise)

In Class Exercise

- Consider a connected graph $G = (V, E)$ that has n vertices and $n-1$ edges where $n > 1$. First we will prove that G has at least one vertex with degree 1
- Q: What is

$$\sum_{v \in V} deg(v)$$

- Q: Is it possible for each vertex to have degree ≥ 2? Why or why not?
- Q: Now show that there must be at least one vertex that has degree 1

In Class Exercise

Now we will prove by induction that any connected graph with n vertices and $n-1$ edges is a tree.

- Q: What is the base case?
- Q: What is the inductive hypothesis?
- Q: Now show the inductive step. Hint: Use the fact proved in the last slide.

DFS and BFS

- If we implement the "bag" by using a stack, we have *Depth First Search*
- If we implement the "bag" by using a queue, we have *Breadth First Search*

Analysis

- Note that if we use adjacency lists for the graph, the overhead for the "for" loop is only a constant per edge (no matter how we implement the bag)
- If we implement the bag using either stacks or queues, each operation on the bag takes constant time
- Hence the overall runtime is $O(|V| + |E|) = O(|E|)$

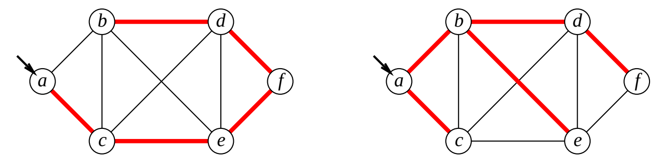
DFS vs BFS

- Note that DFS trees tend to be long and skinny while BFS trees are short and fat
- In addition, the BFS tree contains *shortest paths* from the start vertex s to every other vertex in its connected component. (here we define the length of a path to be the number of edges in the path)

Final Note

- Now assume the edges are weighted
- If we implement the "bag" using a *priority queue*, always extracting the minimum weight edge from the bag, then we have a version of Prim's algorithm
- Each extraction from the "bag" now takes $O(|E|)$ time so the total running time is $O(|V| + |E| \log |E|)$

Example



A depth-first spanning tree and a breadth-first spanning tree of one component of the example graph, with start vertex a.

Searching Disconnected Graphs

If the graph is disconnected, then Traverse only visits nodes in the connected component of the start vertex s. If we want to visit all vertices, we can use the following "wrapper" around Traverse

```
TraverseAll(){
  for all vertices v{
    if (v is unmarked){
      Traverse(v);
    }
  }
}
```

DFS and BFS

- Note that we can do DFS and BFS equally well on undirected and directed graphs
- If the graph is undirected, there are two types of edges in G: edges that are in the DFS or BFS tree and edges that are not in this tree
- If the graph is directed, there are several types of edges

DFS in Directed Graphs

- *Tree edges* are edges that are in the tree itself
- *Back edges* are those edges (u, v) connecting a vertex u to an ancestor v in the DFS tree
- *Forward edges* are nontree edges (u, v) that connect a vertex u to a descendant in a DFS tree
- *Cross edges* are all other edges. They go between two vertices where neither vertex is a descendant of the other

Acyclic graphs

- Useful Fact: A directed graph G is acyclic if and only if a DFS of G yeilds no back edges
- Challenge: Try to prove this fact.

Take Away

- BFS and DFS are two useful algorithms for exploring graphs
- Each of these algorithms is an instantiation of the Traverse algorithm. BFS uses a queue to hold the edges and DFS uses a stack
- Each of these algorithms constructs a spanning tree of all the nodes which are reachable from the start node s

Shortest Paths Problem

- Another interesting problem for graphs is that of finding shortest paths
- Assume we are given a weighted *directed* graph $G = (V, E)$ with two special vertices, a source s and a target t
- We want to find the shortest directed path from s to t
- In other words, we want to find the path p starting at s and ending at t minimizing the function

$$w(p) = \sum_{e \in p} w(e)$$

Example

- Imagine we want to find the fastest way to drive from Albuquerque,NM to Seattle,WA
- We might use a graph whose vertices are cities, edges are roads, weights are driving times, s is Albuquerque and t is Seattle
- The graph is directed since driving times along the same road might be different in different directions (e.g. because of construction, speed traps, etc)

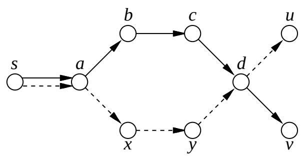
SSSP

- Every algorith known for solving this problem actually solves the following more general *single source shortest paths* or SSSP problem:
- Find the shortest path from the source vertex s to *every* other vertex in the graph
- This problem is usually solved by finding a *shortest path tree* rooted at s that contains all the desired shortest paths

Shortest Path Tree

- It's not hard to see that if the shortest paths are unique, then they form a tree
- To prove this, we need only observe that the sub-paths of shortest paths are themselves shortest paths
- If there are multiple shotest paths to the same vertex, we can always choose just one of them, so that the union of the paths is a tree
- If there are shortest paths to two vertices u and v which diverge, then meet, then diverge again, we can modify one of the paths so that the two paths diverge once only.

Example

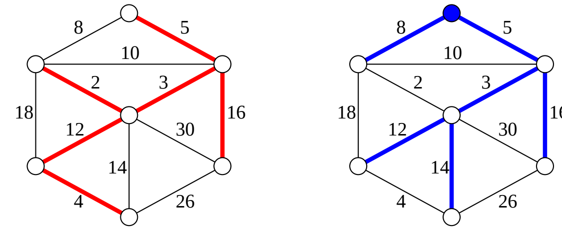


If $s \to a \to b \to c \to d \to v$ and $s \to a \to x \to y \to d \to u$ are both shortest paths,
then $s \to a \to b \to c \to d \to u$ is also a shortest path.

MST vs SPT

- Note that the minimum spanning tree and shortest path tree can be different
- For one thing there may be only one MST but there can be multiple shortest path trees (one for every source vertex)

Example



A minimum spanning tree (left) and a shortest path tree rooted at the topmost vertex (right).