Today’s Outline

- BFS and DFS Wrapup
- Midterm Review

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

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**

- 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:

```plaintext
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 $G$ yields 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 $s$

**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)

**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

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

**SSSP**

- Every algorithm 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 shortest 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).
**Midterm Info**

- Midterm will be Thursday, Nov. 13th at regular class time and place
- You can bring 2 pages of “cheat sheets” to use during the exam. You can also bring a calculator. Otherwise the exam is closed book and closed note.
- The web page contains new links to prior classes and their midterms. *Many of the questions on my midterm will be similar in flavor to these past midterms!*

**Midterm Review Session**

- I will have a review session Weds, Nov. 12th at 6:00pm in FEC 141 (the conference room on the first floor of FEC)
- Maxwell will also have a review session
- Please come with questions

**Midterm**

- 5 questions, about 20 points each
- Hard but fair
- There will be some time pressure, so make sure you can e.g. solve recurrences both quickly and correctly.
- I expect a class mean of between 60 :( and 70 :) points

**New Topics**

- Amortized Analysis: Aggregate Method, Accounting Method, Potential Method, Dynamic Array
- Disjoint Sets: Disjoint Set Operations, Representation as Forest, Union by Rank and Path Compression, Amortized Costs
- Graph Theory: Graph Representations, BFS, DFS
- MST: Definition, Kruskall’s Algorithm, Prim’s Algorithm, Safe Edge Theorem and Corollary
- Single-Source Shortest Paths: Definition and Algorithm
Problem 1

- Collection of true/false, multiple choice and short answer on topics we've covered
- Make sure you know resource bounds for all the algorithms we've covered so far
- Link on website to MIT's algorithms class gives some good example problems

Problem 2 - Amortized Analysis

- I will give you a data structure and code for operations over that data structure. It will be a simple data structure, but not a stack, queue or bit counter
- You show the amortized cost per operation using both the accounting method and potential method
- Accounting method - you will give the charge for each operation and show how you can use these charges to pay for all operations
- Potential Method - I will give you a potential function and you will show that it's valid and will use it to calculate the amortized costs
- Like hw problems from Chapter 17 of text on Stacks and Bit Counters and Exercise 17.3-7

Problem 3 - Union Find

- A question about disjoint sets.
  - Possibility 1: Simulate a disjoint set data structure as in Exercise 21.2-2
  - Possibility 2: question about using the disjoint set data structure, similar to Exercise 21.1-3
  - Possibility 3: ???

Problem 4 - Graph Theory

- Possibility 1: Computing the BFS and DFS trees of a graph
- Possibility 2: Questions about properties of BFS and DFS on certain types of graphs
- Possibility 3: Graph Theory proof, similar to in-class exercise
- Possibility 1: I give you an algorithm and ask you to either show that it always finds an MST or provide a counterexample where it doesn’t, similar to Exercise 23.2-8
- Possibility 2: I give you a graph $G$ and an edge set $A$ and ask you to give me all the safe edges in $G$ along with a cut for each edge which shows that it is safe.
- Possibility 3: A general question about MSTs, similar to Exercise 23.1-1, 23.1-3, 23.1-6
- Possibility 4: Simulation of Kruskal’s and Prim’s, questions about properties of these algorithms on certain types of graphs