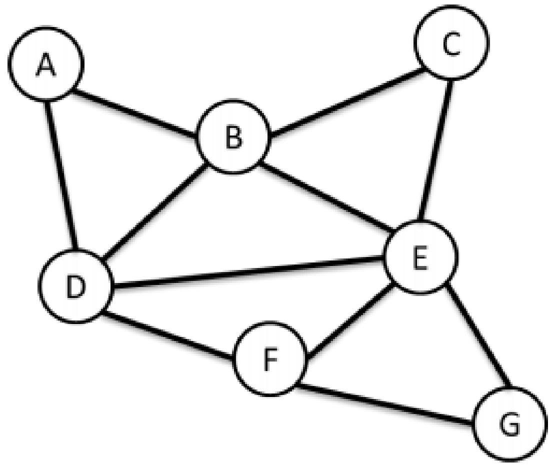
# Intermediate Graph Theory

Alex Chen

October 7, 2011

## 1 Introduction

**Graphs** are extremely useful in both computer science and math, so it is no doubt that a strong graph theory foundation will be useful for programming competitions such as USACO. A graph is defined as a collection of **nodes**, or vertices, interconnected with a set of **edges**. Each edge connects a pair of nodes, either unidirectionally (a **directed** edge) or bidirectionally (an **undirected** edge). Almost every problem has some sort of a graphical representation.



## 2 Shortest Path Algorithms

Within any graph, there exist **paths**. A path is a series of consecutive edges that connect a pair of nodes. A simple path is one that does not use any path more than once, and a cycle is a path that begins and ends at the same node. For example, a simple path from $A$ to $B$ in the above graph is $A \to D \to F \to E \to B$.

In a weighted graph, edges are **weighted**. A weighted edge has some "length" for traversal. The total length of a path is the sum of the lengths of its component edges. Thus, the **shortest path** between any two nodes is the path between the two nodes with the lowest total length. How do we calculate shortest paths?

## 3 Unweighted Graphs

In an unweighted graph, where all edges have the same length of 1, calculating a shortest path between any two pair of nodes is simple.

## 3.1 Breadth-First Search

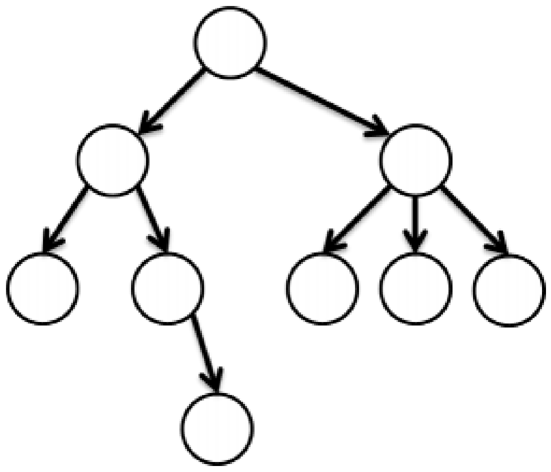
**Problem:** Given an unweighted graph and two nodes $A$ and $B$, find the length of the shortest path from $A$ to $B$.

**Solution:** We can use a breadth-first search from $A$. We start by pushing $A$ into the queue, and at each step we pop the next element and append all its neighbors that have not yet been visited. When $B$ is popped, we can stop our search. This works because a breadth-first search always reaches each node in the shortest number of edges. This method does not work in a directed graph because the path with the shortest number of edges is not necessarily the shortest path. In the example graph, the shortest path from $A$ to $B$ has length 1.

*Exercise:* What is the complexity of finding the shortest path between any pair of nodes $A$ and $B$? Then, show that the slightly harder problem of finding the shortest path from $A$ to every other node has the same complexity.

# 4 Trees

A **tree** is a connected graph with $N - 1$ edges, where $N$ is the number of nodes. (A connected graph has at least one path from each node to every other node.) Trees can be visualized by labeling one node as the root and all other nodes as children of their parent nodes. Each node except the root has one parent.



Trees are like unweighted graphs in that we can compute shortest paths rather easily, regardless of whether the tree's edges are weighted or unweighted.

## 4.1 Depth-First Search

**Problem:** Given a tree and two nodes $A$ and $B$, find the length of the shortest path from $A$ to $B$.

**Solution:** We can use a depth-first search from $A$, treating $A$ as the root of the tree. We recur starting at $A$, and then recur at each of its children until we reach $B$. At that point, we can stop our recursion. This works because in a tree, there is exactly one simple path between any two pairs of nodes. If there was more than one simple path, than there would be a cycle in the graph. That is impossible because trees contain no cycles. Thus, whenever the recursion reaches $B$ from $A$, we have found the shortest path from $A$ to $B$.

*Exercise:* What is the complexity of finding the shortest path between any pair of nodes $A$ and $B$? Then, show that finding the shortest path from $A$ to every other node has the same complexity.

## 5 Dijkstra's Algorithm

What happens if we have a weighted graph that is not a tree? The problem is much harder, but fortunately, smart people have discovered and proved *Dijkstra's algorithm* (pronounced *DIKE-STRA*).

Dijkstra's algorithm finds a shortest path from any node. The algorithm will work for any graph that contains only nonnegative edge weights. This algorithm is a **greedy** algorithm, one that solves problems by choosing the locally best choice at any stage.

### 5.1 Pseudo-code

The following pseudo-code implements Dijkstra's algorithm for a single source, $A$, and a single target, $B$. It returns the shortest possible path length from $A$ to $B$.

```
def Dijkstra(graph, A, B):
    let N = number of vertices
    let dist = integer array of length N
    let processed = boolean array of length N
    for each vertex v:
        dist[v] = infinity
        processed[v] = false
    dist[A] = 0                     // distance from node to itself is 0

    while there are unprocessed vertices:
        let next_v = unprocessed vertex with the smallest distance in dist
        processed[next_v] = true

        if dist[u] == infinity:
            break // we are done

        pop next_v from queue
        for each edge e (a -> b) starting from next_v:
            new_d = dist[next_v] + e.length
            if new_d < dist[b]:     // if this distance is better
                dist[b] = new_d     // update it

    return dist[B]
```

We use *dist* to hold the current "best" distances to each node, with "best" being the shortest. Through each iteration of the loop, we try to improve upon previous bests by trying new paths. The values in "dist" are initially infinity to note that we have not yet found a path to these nodes. At every step of the loop, we choose the next node to process. To find this next node, we consider all nodes that have not been processed and find the one with the current shortest distance to $A$.

After we have found the node to process next, we consider all the edges from that node and **relax** them. Edge relaxation refers to the process of seeing if a new shortest path can be generated by using that node. Suppose that an edge goes from $a$ to $b$. We check if we can find a better path to $b$ than the current path by first going to $a$, and then taking this edge. The total distance is ($dist[a]+$edge length). If this edge improves the shortest path, then we can update the distance to $b$.

Like with BFS and DFS, calculating the shortest path to one node is the same complexity as calculating the shortest path to all nodes, as long as we are using a single source. If we wanted to find the all the shortest paths (the "all shortest paths problem"), we can simply run Dijkstra's algorithm from every source.

*Exercise:* How do we keep track of the path from $A$ to $B$ (the sequence of nodes) while using Dijkstra's algorithm?

### 5.2 Priority Queue Dijkstra

Consider this line from the pseudo-code.

```
let next_v = unprocessed vertex with the smallest distance in dist
```

How do we perform this step? The naive way is to check all the nodes in the queue and to choose the one with the smallest distance. However, there are $N$ nodes to check, so this step would take $O(N)$ time. The main loop runs $N$ times, once for each node, so the entire algorithm would be at least $O(N^2)$. We can do better than that by improving the way we choose the next node to process.

We can use a **priority queue**. A priority queue is like a queue in that one can push elements, but differs in that the element popped is always the minimum (or the maximum) element. If the priority queue is implemented with a binary heap, then both the push and pop operations take $O(\log N)$ time. There are faster ways to implement a priority queue, but a binary heap is simple and efficient enough for most applications.

*Exercise:* What is the runtime complexity of Dijkstra's algorithm? Let $E$ be the number of edges and $V$ be the number of nodes. Find the complexity for both regular Dijkstra and priority queue Dijkstra.

## 6 Problems

1. Farmer John and Bessie live on a farm that is a collection of pastures with weighted edges between some pairs of pastures. Given the locations of Bessie and Farmer John, find the length of the shortest path between Farmer John and Bessie.

2. Farmer John owns a collection of pastures with weighted edges between some pairs of locations. Each pasture is inhabited by a cow, and the cows wish to all congregate at one of the pastures. Find the pasture at which the cows should meet in order to minimize combined travel distance.

   Bonus: Suppose you know that the given graph is a tree. Can you find the pasture in O(n) time?

3. You are given a set of locations, lengths of roads connecting them, and an ordered list of package dropoff locations. Find the length of the shortest route that visits each of the package dropoff locations in order.

4. Bessie and her two friends inhabit a farm made up of pastures and weighted cowpaths. The three cows live at three different pastures. Bessie wants to place food at a pasture such that the total distance that all three cows must travel to reach the food is minimized. Help Bessie choose this optimal pasture.

## 7 Have Fun!

Try implementing Dijkstra sometime. There's a problem on the grader ("gogogo") to test your program with.