Network Flow

Senior Computer Team

January 16, 2004

1 Introduction

Network Flow is an algorithm to solve what may seem to be a narrow range of problems, but can actually be applied to a few other situations. The basic setup for network flow is exactly what it sounds like: pushing as much "flow" as possible through a "network". More specifically, given a graph with a maximum capacity for each edge, find the maximum amount of flow from the *sink* to the *source*.

2 The Algorithm

While a simple greedy algorithm might be your first thought, it is fairly simple to show that it will not always work; you might push some flow the "wrong way". So to get around this, you actually keep track of *two* capacities for each node: one in each direction. When you put some flow through an edge, you subtract that amount from the direction the flow goes, and add it in the other direction. This lets you "cancel out" your mistakes. But how do you find a path in the first place? While BFS is one possibility (i.e., just find a path from the source to the sink), a modified Dijkstra is *nearly* always faster. This Dijkstra searches by maximum flow rather than minimum distance, so it will find the path that allows maximum flow to be pushed through. It also keeps track of the previous node, so that the edges can be updated. Pseudo-code is shown on the final page. However, if all the edges have equal length, the Dijkstra can be replaced by a much faster BFS. And, of course, if the edges are all very short, they can be replaced by multiple length 1 edges, allowing BFS to be used.

3 Applications of Network Flow

3.1 Bipartite Matching

Problem: Farmer John wants to pair cows with different stalls. However, each cow only likes some of the stalls. Find the pairing that satisfies the most cows (for, as we know, discontent bovines can lead to trouble).

This problem has no graphs, yet is still network flow! The reasoning is simple: give each cow and each stall a node, and connect each cow to all of the stalls she likes with an edge of weight 1. Then create a new source node that has a connection of weight 1 to all of the cows, and a sink connected to each stall. If the network flow algorithm is applied to the resulting graph, the edges with flow will represent the pairings of cows to stalls. Thus, network flow often applies to problems that involve matching up two or more groups as efficiently as possible.

3.2 Min Cut

Problem: The cows have two camps in two separate fields which are sending secret messages through other fields. Help Farmer John (who has a complete list of fields and paths between the fields, as well as the cost of blocking each path) who wants to cut the two fields off from each other by spending as little as possible blocking each field.

This problem does involve graphs, but still seems unrelated to network flow. Yet it, too, can be solved using those methods. The trick is that, if you find the maximum flow, the nodes that you want to cut will have zero flow through them. So if you floodfill from the source and the sink along edges that have non-zero capacity after running network flow, the edges connected the two components will be the edges to cut.

4 Problems

1. Code up network flow; this is often the hardest part of a problem. It is a fairly long one, with plenty of places to make mistakes. The more you do it, the faster you will get at it.

2. Farmer John is giving his cows gifts in an attempt to improve moral. He has a set of gifts to give out, and knows how many gifts he can give each cow (different for each cow) before more gifts cease to have an effect. Help him distribute the gifts among the cows to maximize the total happiness of his herd.

3. You are working for quality control for a milk distribution company, and you just learned a shipment of bad milk was sent out. You know where it's going, but not how it's going to get there. The distribution system consists of several warehouses, with trucks running between warehouses. You want to prevent the milk from being delivered by shutting down certain trucks; each truck will result in a certain amount of lost money. Find the trucks that should be stopped to minimize lost money while ensuring that the milk is not delivered.

4. The cows are building a rail system between the barns and pastures, represented as points on a 32×32 lattice. The rail lines must run along the lattice (i.e, the must connect orthogonally adjacent lattice points), and may not intersect or overlap (except and pastures and barns). There are also several large boulders lying around that make

a lattice point unusable for railroading. Help them find the maximum total number of railroads they can build connecting pastures and barns (the same pasture and barn may be connected by several different railroads).

5. The cows are setting up umbrellas on the beach for shade. The beach can be represented as a 50×50 grid, with cows sitting in certain squares, and an umbrella covering two orthogonally adjacent squares. The cows want to shade as many of themselves as possible, but are firmly against waste and do not want to shade any squares that do not have cows. Find out the maximum number of cows that can be covered without covering any cow-less area.

6. Farmer John has a number of cows capable of milking themselves. Whenever a cow enters the barn, she instantaneously hooks herself up to the milking machine and produces milk at a constant rate (potentially different for each cow) for the duration of her stay in the barn, and unhooks herself immediately before leaving the barn. Farmer John can only measure the aggregate output of all cows in the barn at any given time; given a set of these measurements and the times when each cow entered and left the barn, find the milk production rate of each cow.

```
1  if (source = sink)
2      totalflow = Infinity
3      DONE
4  totalflow = 0
5  while (True)
6      // find path with highest capacity from source to sink
7      // uses a modified djikstra's algorithm
8      for all nodes i
9          prevnode(i) = nil
10         flow(i) = 0
11         visited(i) = False
12     flow(source) = infinity
13     while (True)
14         maxflow = 0
15         maxloc = nil
16         // find the unvisited node with the highest capacity to it
17         for all nodes i
18             if (flow(i) > flow(maxloc) AND not visited(i))
19                 maxloc = i
20         if (maxloc = nil)
21             break inner while loop
22         if (maxloc = sink)
23             break inner while loop
24         // update its neighbors
25         for all neighbors i of maxloc
26             if (flow(i) < min(maxflow,capacity(maxloc,i)))
27                 prevnode(i) = maxflow
28                 flow(i) = min(maxflow, capacity(maxloc,i))
29     if (maxloc = nil) // no path
30         break outer while loop
31     pathcapacity = flow(sink)
32     // add that flow to the network, update capacity appropriately
33     totalflow = totalflow + pathcapacity
34     curnode = sink
35     while (curnode != source)
36         nextnode = prevnode(curnode)
37         capacity(nextnode,curnode) = capacity(nextnode,curnode) - pathcapacity
38         capacity(curnode,nextnode) = capacity(curnode,nextnode) + pathcapacity
39         curnode = nextnode
```