Details of Seidel's Algorithm

Last time we presented the following algorithm to compute the shortest-path distance matrix of a connected, unweighted, undirected graph:

- Compute $M[G_2]$ in $O(\mu(n))$ time.
- If $M[G_2]$ is all ones off the diagonal, return the answer.
- Otherwise compute $D'[G] = D[G_2]$ recursively.
- Compute $P[G]$ from $D'[G]$ in time $O(\mu(n))$ by a method to be given later.
- Return $D[G] = 2D'[G] - P[G]$.

We prove correctness by induction on the **diameter** of G, the largest entry of the matrix $D[G]$ (this of course can be at most $n-1$). If the diameter is at most 2, then $M[G_2]$ must have ones off the diagonal, since G_2 is a complete graph. In that case $M[G]$ and $M[G_2]$ give us all the information we need to compute $D[G]$ in $O(n^2)$ further time: $D[G](i,j)$ is 0 if $i=j$, 1 if $M[G](i,j)=1$, and 2 otherwise.

If the diameter of G is greater than 2, note that the diameter of G_2 must be *smaller*. (In fact it is the ceiling of half the diameter of G, because every entry of $D[G_2]$ is the ceiling of half the corresponding entry of $D[G]$.) By the inductive hypothesis, then, we may assume that $D'[G]$ is computed correctly.

Every entry of $D[G]$ is either twice the corresponding entry of $D'[G]$, or twice that entry minus one. The matrix $P[G]$ tells us which case is true for each entry.

Timing of Seidel’s Algorithm

Let $T(n, d)$ be the time taken by Seidel’s algorithm on a graph with n vertices and diameter at most d.

First note that $T(n, 2) = O(\mu(n))$ because if the diameter is at most 2, we find this out by doing only one boolean matrix multiplication, and this takes time $O(\mu(n))$.

Then observe that for $d > 2$, we have the recurrence $T(n, d) \leq T(n, \lceil d/2 \rceil) + O(\mu(n))$. This is easy to solve: For d a power of 2, and hence for general d,

$$T(n, d) = O(\mu(n) \log d)$$

and hence for any graph of n vertices,

$$T(n) = O(\mu(n) \log n).$$

We now have to fill in details of the algorithm:

- Compute $M[G_2]$ in $O(\mu(n))$ time.
- If $M[G_2](i,j) = 1$ whenever $i \neq j$, return the answer.
- Otherwise compute $D'[G] = D[G_2]$ recursively.
- Compute $P[G]$ from $D'[G]$ in time $O(\mu(n))$ by a method we're about to present.
- Return $D[G] = 2D'[G] - P[G]$.

We showed that the algorithm is correct and runs in a time of $O(\mu(n) \log n)$, assuming that we can compute $P[G]$ from $D'[G]$ in $O(\mu(n))$ time. Recall that $P[G]$ is the matrix whose (i,j) entry is 1 iff the shortest-path distance from i to j is *odd*.

We will need one *integer* matrix multiplication for our computation. We take $D'[G]$, which records the distances in G_2, and multiply it by A, the adjacency matrix of G. Let's look at the result X and its entry for some pair of vertices i and j (with $i \neq j$).

The entry $X(i, j)$ is the sum, over all vertices k adjacent to j, of the distance in G_2 from i to k. Let c be the degree of j and let $d = D'[G](i, j)$. We will show:

- If $P[G](i, j) = 1$, then $X(i, j) < cd$, and
- If $P[G](i, j) = 0$, then $X(i, j) \geq cd$.

Since we have c and d available, then, computing X will give us $P[G]$ directly.

- If $P[G](i,j) = 1$, then $X(i,j) < cd$, and

Proof: The actual distance from i to j in G must be $2d-1$, since it is odd and half of it rounded up to d. The vertices k adjacent to j must have distances to i of either $2d-2$, $2d-1$, or $2d$, since these distances can only differ from $2d-1$ by one. Furthermore, at least one k, the one on the shortest path from i to j, must have $d(i,k) = 2d-2$. Thus $X(i,j)$ consists of the sum of c distances in G_2 that are either $d-1$ or d, with at least one $d-1$.

- If $P[G](i,j) = 0$, then $X(i,j) \geq cd$.

Proof: In this case the distance in G from i to j must be $2d$, since it is even and half of it rounded to d. For any k adjacent to j, then, the distance from i to k is either $2d-1$, $2d$, or $2d+1$. The corresponding distances in G_2 are thus all at least d, and thus $X(i,j)$ is at least cd.

The Network Flow Problem

We now turn to a graph problem somewhat similar to the ones we have seen, with a wide variety of applications.

A **flow network** is a directed graph where each edge (u, v) has a positive weight $c(u, v)$ called a **capacity**, and two distinct nodes are designated as the **source** s and the *sink* t. A **flow** on the network is a function f from $V \times V$ to the reals that satisfies three rules:

- for any distinct vertices u and v, $f(u, v) = -f(v, u)$
- for any v except s or t, the sum over all u of $f(u, v)$ is 0
- for any u and v, $f(u, v) \leq c(u, v)$ (where if $(u, v) \notin E$, $c(u, v) = 0$)

We can think of the flow as a movement of stuff through the network. The first condition says that if we have both an edge and its reversal, we consider only the net movement between the vertices, with the sign giving its direction. The second condition says that stuff cannot be created or destroyed at intermediate vertices - in fact some will normally be created at the source and an equal amount destroyed at the sink. The third condition explains the capacity – we cannot have more flow over an edge than the capacity allows.

Consider arbitrary vertices u and v. If there is only an edge (u, v) in one direction with capacity a, the flow $f(u, v)$ must be in the range from 0 through a. If there is also an edge (v, u) with capacity b, then the flow $f(u, v)$ must be in the range from $-b$ through a.

The **size** of the flow is the sum of the edge flows out of s, which must equal the sum of the edge flows into t. (Can you prove this quickly?) The **max-flow problem** is to input a flow network and find a flow whose size is as large as possible.

The central notion in our analysis of flow networks is the **cut**. A cut is a partition of V into two disjoint sets A and B with $s \in A$ and $t \in B$. Given a flow f, the **flow across** a cut (A, B) is the sum, for all $u \in A$ and all $v \in B$. of $f(u, v)$.

Lemma: For any cut (A, B) and any flow f, the flow across the cut equals $|f|$, the size of the flow.

Proof: We use induction on the size of A. If $|A| = 1$, the flow across the cut is exactly the flow out of s which is $|f|$ by definition. Assume the flow across the cut (A, B) is $|f|$, let v be any vertex in B except for t, and let $A' = A \cup \{v\}$ and $B' = B \setminus \{v\}$. The flow across (A', B') is the flow across (A, B), minus the flow on edges from A to v, plus the flow on edges from v to B'. This is $|f|$ (by the inductive hypothesis) plus the net flow into v (since every vertex except v is in either A or B'). By the conservation rule for v, the flow across (A', B') is thus $|f|$.

The **capacity** of a cut (A, B) is the sum, over all $u \in A$ and $v \in B$, of $c(u, v)$. It should be clear that the flow across a cut cannot exceed the capacity of a cut.

Corollary: The size of a flow cannot exceed the minimum of the capacities of all cuts.

Our fundamental result is the converse of this corollary.

The Max-Flow-Min-Cut Theorem

Theorem: In any flow network, the size of the maximum flow is exactly equal to the minimum of the capacities of all cuts.

Proof: We have already shown that the flow cannot exceed the minimum capacity – we must show that it can *achieve* the minimum capacity.

Given a flow network and a particular flow f, we define the **residual network** to be the flow network where the capacity of each edge (u, v) is $c_f(u, v) = c(u, v) - f(u, v)$ (since $f(u, v) \leq c(u, v)$, this quantity cannot be negative). (Note that an edge may occur in the residual network if it either it *or its reversal* is in the original network.) A legal flow in the residual network is thus exactly a flow that may be *added* to f resulting in another legal flow through the original network.

We claim that if f does not achieve the capacity of any of the possible cuts, then there is a path from s to t in the residual network consisting of edges with positive capacities. We prove this claim by contrapositive: If there is no path from s to t, let A be the set of nodes that are reachable from s in the residual network (by positive edges) and consider the cut $(A, V \setminus A)$. For every $u \in A$ and $v \in V \setminus A$, we must have that $f(u,v) = c(u,v)$ or else an edge across the cut would exist in the residual network. But then f has achieved the capacity of this cut, and hence of the minimum cut. We have proved our claim.

A path through the residual network is called an **augmenting path**, and the minimum of the capacities of its edges is called its **bottleneck capacity** b. If an augmenting path exists for a flow f, then f *cannot* be a maximum flow, because we can add a flow of size b along the augmenting path to f and get a strictly larger flow that is still legal.

The Ford-Fulkerson Algorithm

Our proof of the Max-Flow-Min-Cut Theorem immediately gives us an algorithm to *compute* a maximum flow, known as the **Ford-Fulkerson algorithm**:

- Set f to be a zero flow.
- While the residual graph of f contains an augmenting path, find such a path, create a flow of the bottleneck capacity along this path, and add that flow to f.
- Once there is no augmenting path, return f.

Let's immediately note an important consequence of this algorithm. If the capacities of all the edges are integers, and the edge flows are all integers, then the capacities in the residual network are all integers and hence the bottleneck capacity of any augmenting path is an integer. Since the flow begins as a zero flow, if the capacities are all integers the edge flows will *remain* integers through the entire course of the Ford-Fulkerson algorithm. We have proved:

Theorem: Any flow network whose capacities are all integers has a maximum flow whose edge flows are all integers.

One phase of this algorithm takes only polynomial time – we need $O(e)$ time to search (depth-first or breadth-first) for the augmenting path, then $O(n)$ to update the residual graph to reflect the new f. But it's not immediately clear how many phases we might need to reach a maximum flow. The only obvious bound is $|f|$, in the special case where the capacities are integers, since we know that the flow will increase by at least one each time (the bottle-neck capacity must be a positive integer). Of course $|f|$ could be much larger than n. Could the performance of Ford-Fulkerson be that bad? In fact it could:

```
       (a)
       ^|\
1000  / | \ 1000
     /  |  \
    /  1|   \
   /    |    V
(s)     |    (t)
   \    |    ^
    \   |   /
1000 \  |  / 1000
      \ | /
       VV/
       (b)
```

If we pick a first augmenting path from s to a to b to t, the resulting residual network has a path from s to b to a to t. We could then take a path through a and b, then one through b and a, and so on until we find the maximal flow of size 2000 only after 2000 phases.

It turns out that we can avoid this bad behavior by using the **Edmonds-Karp heuristic**: When we pick an augmenting path, we always pick one that is as short as possible in terms of the number of edges – so, for example, we could just pick one by breadth-first search.

Theorem: If the Edmonds-Karp heuristic is used, then the Ford-Fulkerson algorithm terminates with a maximum flow after at most ne phases.

We'll prove this theorem in the next lecture.

Network-Flow Applications

The Max-Flow-Min-Cut Theorem and the existance of integer-size flows give easy proofs of some classic results in graph theory:

Note that we did not get to the proofs of these two results in the live Lecture #10, so they will be repeated in Lecture #11.)

Hall's Theorem: (early 1900's) A bipartite graph $G = (U, V, E)$, with $|U| = |V| = n$, has a perfect matching iff there does *not* exist a set $A \subseteq U$ such that the set $\Gamma(A) = \{v \in V : \exists u \in A : (u, v) \in E\}$ is smaller than A.

Proof: Set up a flow network with a node s that has edges of weight 1 to every node in U and a node t that has edges of weight 1 *from* every node in V. Give every edge in E weight 1. If this network has a flow of size n, then it has an integer flow, which must have edge values of 0 or 1, and the edges of E used in this flow form a perfect matching. If the network does not have such a flow, there must be a cut (X, Y) of capacity $n-1$ or smaller. The set X contains s plus some nodes of U and V. (Why must it contain at least one node of U?). Let A be $X \cap U$. We will show that $|\Gamma(A)| < |A|$.

We first modify the cut by taking any nodes in $\Gamma(A) \cap Y$ and moving them to X. This doesn't *increase* the capacity of the cut – if y is such a vertex then the edge (y, t) is the only one that now crosses the cut but didn't before, and at least one edge from A to y no longer crosses the cut.

Let $|A| = k$. The $n - k$ edges from s to $U \setminus A$ cross the cut, as do the edges from $\Gamma(A)$ to t. This gives us at least $n - k + |\Gamma(k)|$ edges crossing the cut, so if $\Gamma(A) \geq k$ we have a contradiction because the capacity of the cut was assumed to be at most $n - 1$.

Here is another classic result in graph theory:

Menger's Theorem: (1927) In any directed graph with nodes s and t, the maximum number of edge-disjoint paths from s to t is equal to the minimum number of edges whose removal separates s from t.

Proof: This is simply the special case of the Max-Flow-Min-Cut Theorem when all the edges have weight 1. If the size of the minimum cut is k, there must exist a flow of size k, and hence an integer flow of size k. We can easily divide this flow into k edge-disjoint paths from s to t.