CS 573: Algorithms, Fall 2012

Homework 3, due Monday, October 29, 23:59:59, 2012

Version 1.1

Name:	
Net ID:	

Neatly print your name(s), NetID(s). If you are off campus, please submit the homework on moodle, otherwise submit the homework in SC 3306 (or sliding it under the door). Please solve each problem on a separate page.

> "Napoleon has not been conquered by man. He was greater than all of us. But god punished him because he relied on his own intelligence alone, until that prodigious instrument was strained to breaking point. Everything breaks in the end."
>
> – Carl XIV Johan, King of Sweden.

Required Problems

1. Independence Matrix

[30 Points]

(A) **[15 Points]** Consider a $0-1$ matrix H with n_1 rows and n_2 columns. We refer to a row or a column of the matrix H as a line. We say that a set of 1's in the matrix H is ***independent*** if no two of them appear in the same line. We also say that a set of lines in the matrix is a ***cover*** of H if they include (i.e., "cover") all the 1's in the matrix. Using the max-flow min-cut theorem on an appropriately defined network, show that the maximum number of independent 1's equals the minimum number of lines in the cover.

(B) **[15 Points]** Let M be an $n \times n$ matrix with each entry equal to either 0 or 1. Let m_{ij} denote the entry in row i and column j. A *diagonal entry* is one of the form m_{ii} for some i.

Swapping rows i and j of the matrix M denotes the following action: we swap the values of m_{ik} and m_{jk}, for $k = 1, \ldots, n$. Swapping two columns is defined analogously.

We say that M is ***rearrangeable*** if it is possible to swap some of the pairs of rows and some of the pairs of columns (in nay sequence) so that after all the swapping, all the diagonal entries of M are equal to 1.

(B.I) **[5 Points]** Give an example of a matrix M that is not rearrangeable, but for which at least one entry in each row and each column is equal to 1.

(B.II) **[10 Points]** Give a polynomial-time algorithm that determines whether a matrix M with 0-1 entries is rearrangeable.

2. Unique Cut

[30 Points]

(A) **[12 Points]** Let $G = (V, E)$ be a directed graph, with source $s \in V$, sink $t \in V$, and nonnegative edge capacities $\{c_e\}$. Give a polynomial-time algorithm to decide whether G has a *unique* minimum s-t cut (i.e., an s-t of capacity strictly less than that of all other s-t cuts).

(B) **[18 Points]** The good, the bad, and the middle.

Suppose you're looking at a flow network G with source s and sink t, and you want to be able to express something like the following intuitive notion: Some nodes are clearly on the "source side" of the main bottlenecks; some nodes are clearly on the "sink side" of the main bottlenecks; and some nodes are in the middle. However, G can have many minimum cuts, so we have to be careful in how we try making this idea precise.

Here's one way to divide the nodes of G into three categories of this sort.

- We say a node v is ***upstream*** if, for all minimum s-t cuts (A, B), we have $v \in A$ – that is, v lies on the source side of every minimum cut.
- We say a node v is ***downstream*** if, for all minimum s-t cuts (A, B), we have $v \in B$ – that is, v lies on the sink side of every minimum cut.
- We say a node v is ***central*** if it is neither upstream nor downstream; there is at least one minimum s-t cut (A, B) for which $v \in A$, and at least one minimum s-t cut (A', B') for which $v \in B'$.

Give an algorithm that takes a flow network G and classifies each of its nodes as being upstream, downstream, or central. The running time of your algorithm should be within a constant factor of the time required to compute a *single* maximum flow.

3. Maximum Flow By Scaling

[40 Points]

Let $G = (V, E)$ be a flow network with source s, sink t, and an integer capacity $c(u, v)$ on each edge $(u, v) \in E$. Let $C = max_{(u,v)\in E} c(u, v)$.

(A) **[4 Points]** Argue that a minimum cut of G has capacity at most $C|E|$.

(B) **[10 Points]** For a given number K, show that an augmenting path of capacity at least K can be found in $O(E)$ time, if such a path exists.

(C) **[6 Points]** The following modification of Ford-Fulkerson-Method can be used to compute a maximum flow in G.

```
MaxFlowByScaling(G, s, t)
1    C ← max_{(u,v)∈E} c(u, v)
2    initialize flow f to 0
3    K ← 2^⌊lg C⌋
4    while K ≥ 1 do {
5        while (there exists an augmenting path p of
                        capacity at least K) do {
6            augment flow f along p
         }
7        K ← K/2
     }

8    return f
```

Argue that **MaxFlowByScaling** returns a maximum flow.

(D) **[8 Points]** Show that the capacity of a minimum cut of the residual graph G_f is at most $2K|E|$ each time line 4 is executed.

(E) **[8 Points]** Argue that the inner **while** loop of lines 5-6 is executed $O(|E|)$ times for each value of K.

(F) **[4 Points]** Conclude that **MaxFlowByScaling** can be implemented so that it runs in $O(E^2 \lg C)$ time.