BU CLA CS 530: Analysis of Algorithms
Spring 1999

Selected Solutions for Problem Set 6

Problem 1.

$G = (V, E)$ is a connected undirected multigraph where $|V| = n \geqslant 1$ and $|E| = m \geqslant n - 1$.

1. The hard part is how to design the data structure representing G. This is already described in the hint. The time required for a single contraction is $\mathcal{O}(n)$. There are different ways of writing the implementation details (omitted here). Whichever way you choose, you can keep $n \times n$ to be the size of the adjacency matrix throughout the operation of CONTRACT.

2. In addition to the adjacency matrix $A = (a_{ij})$, where a_{ij} is the multiplicity of edge (i, j), it is useful to simultaneously maintain a one-dimensional array $D = (d_1, \ldots, d_n)$ where d_i is the degree of vertex i. Maintaining D simultaneously with A does not increase the running time of a single contraction, i.e., it is still $\mathcal{O}(n)$. The following algorithm will select uniformly at random an edge (i, j) in $\mathcal{O}(n)$ time:

   $r \leftarrow \text{RANDOM}(1, \sum_{k=1}^{n} d_k)$
   $i \leftarrow 1$
   $j \leftarrow 1$
   while $r > d_i$ **do**
       $r \leftarrow r - d_i$
       $i \leftarrow i + 1$
   while $r > a_{ij}$ **do**
       $r \leftarrow r - a_{ij}$
       $j \leftarrow j + 1$
   return edge (i, j)

   I omit the correctness proof for this algorithm (not too difficult).

3. The **while** loop in algorithm CONTRACT is iterated $(n - 2)$ times. Each iteration requires $\mathcal{O}(n)$ time for selecting an edge uniformly at random and $\mathcal{O}(n)$ time to contract a single edge, for a total of $\mathcal{O}(n) + \mathcal{O}(n) = \mathcal{O}(n)$ time per iteration. Hence, the total running time of CONTRACT is $(n - 2) \cdot \mathcal{O}(n) = \mathcal{O}(n^2)$.

Problem 2. (Exercize 27.2-9, page 600, in [CLR])

For any 2 vertices u and v in G, define a flow network $G_{u,v}$ consisting of the directed version of G with all edge capacities 1, and with $s = u$ (the source) and $t = v$ (the sink). $G_{u,v}$ has $\mathcal{O}(E)$ edges, and the vertices of $G_{u,v}$ are the set V. We want all the capacities to be 1 so that the number of edges crossing a cut = the capacity of the cut. Let $f_{u,v}$ denote a maximum flow in $G_{u,v}$.

Claim: For any u in V, the edge connectivity is $k = \min\{|f_{u,v}| : v \text{ is a vertex in } V - \{u\}\}$.

Proof of the claim:

Let $m = \min\{|f_{u,v}| : v \text{ is a vertex in } V - \{u\}\}$. We first prove that $k \geqslant m$. Suppose we remove only $(m-1)$ edges from G. For any vertex v, by the max-flow min-cut theorem, u and v are still connected. (The max flow from u to v is at least m, hence any cut separating u from v has capacity at least m, which means at least m edges cross any such cut. Thus at least 1 edge is left crossing the cut when we remove $(m-1)$ edges.) Thus every vertex is connected to u, which implies that the graph is still connected. So at least m edges must be removed to disconnect the graph, i.e. $k \geqslant m$.

We next prove that $k \leqslant m$. Consider a vertex v with the minimum $|f_{u,v}|$. By the max-flow min-cut theorem, there is a cut of capacity $|f_{u,v}|$ separating u and v. Since all edge capacities are 1, exactly $|f_{u,v}|$ edges cross this cut. If these edges are removed, there is no path from u to v and so our graph becomes disconnected. Hence $k \leqslant m$.

Based on the above claim, we can find the edge connectivity k of G as follows.

```
EDGE-CONNECTIVITY(G)
1.  Select any vertex u in V
2.  for each vertex v in V − {u}
3.      do set up the flow network G_{u,v} as described above
4.          find the maximum flow f_{u,v} in G_{u,v}
5.  return min{|f_{u,v}| : v is a vertex in V − {u}}
```

The maximum-flow algorithm is run $(|V|-1)$ times, each time on a flow network with $\mathcal{O}(V)$ vertices and $\mathcal{O}(E)$ edges.

Problem 3.

Let every edge in G have capacity 1, and find a maximum flow f^* in G. By removing a simple path p from s to t which is carrying positive flow, we reduce the value of the maximum flow $|f^*|$ by exactly one unit, because the flow along p cannot exceed the capacity of any of its edges (which is 1, by design). We can thus remove $|f^*|$ edge-disjoint paths from s to t before reducing the flow to zero. This shows that if $|f^*| \geqslant k$, then there are at least k edge-disjoint paths from s to t in G.

Conversely, if there are k edge-disjoint paths from s to t, then $|f^*| \geqslant k$ because we can push one unit of flow along each of these paths.

We have shown that there exist k edge-disjoint paths from s to t in G if and only if $|f^*| \geqslant k$. It suffices therefore to run one of the efficient algorithms to find a maximum flow from s to t in G, when all edge capacities are 1.

Problem 4. (Problem 27-1, page 625, in [CLR])

(a) For a flow network $G = (V, E)$ with both edge and vertex capacities we have: edge capacity $c : E \to \mathbf{R}$ (as before) and vertex capacity $d : V \to \mathbf{R}$. The previous requirements of "(edge) capacity constraint", "skew symmetry", and "flow conservation" (page 580 in [CLR]) are the same. But there is a new requirement:

Vertex capacity constraint: For every $v \in V$ we require $\sum_{u \in V}\{f(u,v) \mid f(u,v) > 0\} \leq d(v)$.

We construct a new flow network $G' = (V', E')$ with edge capacities only such that every maximum flow in G corresponds to a maximum flow in G', and vice-versa. Denote the edge capacity of G' by $c' : E' \to \mathbf{R}$. The construction of G' from G consists in splitting every vertex $v \in V$ into two vertices, v^{in} and v^{out}, and inserting a new edge $(v^{\text{in}}, v^{\text{out}})$ with capacity $c'(v^{\text{in}}, v^{\text{out}}) = d(v)$ between the two parts of the split vertex. Formally,

$$\begin{array}{rcll} V' & =_{\text{def}} & \{\ v^{\text{in}}, v^{\text{out}} \mid v \in V\ \} & \\ E' & =_{\text{def}} & \{\ (u^{\text{out}}, v^{\text{in}}) \mid (u,v) \in E\ \} \ \cup\ \{\ (v^{\text{in}}, v^{\text{out}}) \mid v \in V\ \} & \\ c'(u^{\text{out}}, v^{\text{in}}) & =_{\text{def}} & c(u,v) & \text{for every } (u,v) \in E \\ c'(v^{\text{in}}, v^{\text{out}}) & =_{\text{def}} & d(v) & \text{for every } v \in V \end{array}$$

If the source vertex in G is s and the sink vertex is t, then the source vertex in G' is s^{in} and the sink vertex is t^{out}. You get full credit if you reach this far, but to fully complete the argument you still need to prove (not too difficult) that every maximum flow in G corresponds to a maximum flow in G', and vice-versa.

Let us determine the cost of constructing G' from G, and then running a max flow algorithm on G' — this is needed in part (b). The cost of splitting every vertex v into v^{in} and v^{out}, to obtain V', is $\mathcal{O}(V)$. The cost of inserting a new edge $(v^{\text{in}}, v^{\text{out}})$ and assigning it capacity $d(v)$, repeated $|V|$ times, is also $\mathcal{O}(V)$. The total cost of this construction is therefore $\mathcal{O}(V)$. Note that $|V'| = 2 \cdot |V|$ and $|E'| = |E| + |V|$. If we run an implementation of the FORD-FULKERSON method on G', the cost is $\mathcal{O}(E' \cdot |f^*|) = \mathcal{O}((E+V) \cdot |f^*|)$, where f^* is the maximum flow (assuming also that the capacities are integral values). If we choose to run the EDMONDS-KARP implementation on G', its running time is $\mathcal{O}(V'E'^2)$ (see the end of Section 27.2 in [CLR]). Hence, the total cost of finding a max flow in the original G is:

$$\mathcal{O}(V) + \mathcal{O}(V'E'^2) \;=\; \mathcal{O}(V'E'^2) \;=\; \mathcal{O}(2 \cdot V \cdot (E+V)^2)$$

Under the realistic assumption (why?) that $|E| = \Omega(V)$, or equivalently $|V| = \mathcal{O}(E)$, we obtain $\mathcal{O}(2 \cdot V \cdot (E+V)^2) = \mathcal{O}(VE^2)$ for the cost of finding a max flow in the original G.

(b) We reduce the "escape problem" to the "problem of maximum flow in a network with edge and vertex capacities". The reduction consists in transforming the input grid of the escape problem into such a flow network $G = (V, E)$. Initially there are m starting points on the grid and $4n - 4$ boundary points. The construction of $G = (V, E)$ proceeds as follows:

1. Introduce an artificial source vertex s and connect s to each of the m starting points.
2. Introduce an artificial sink vertex t and connect each of the $4n - 4$ boundary points to t.
3. Each undirected edge between points u and v in the original grid is changed into two directed edges (u, v) and (v, u) in G.
4. Assign edge capacity $= 1$ to each edge in G and vertex capacity $= 1$ to each vertex in G.

This completes the construction of G. To determine whether there is an escape in the original grid, we find a maximum flow through G (using any of the algorithms proposed in Chapter 27 of [CLR]): If the value of the maximum flow in $G = m$, then there is an escape in the original grid, else (if it is $< m$) there is no escape in the original grid.

You get full credit if you reach this far. More rigorously, however, you also need to prove this last assertion, i.e. there is an escape in the original grid iff the maximum flow in $G = m$ (not too difficult).

If you run the EDMONDS-KARP implementation of the FORD-FULKERSON method on G, the running time is $\mathcal{O}(VE^2)$, as shown in part (a). But we need to determine this running time in terms of the input size of the escape problem. The input grid has n^2 points and $2n^2 - 2n$ undirected edges (why?). Hence, G has $2 + n^2 = \mathcal{O}(n^2)$ vertices and $m + (4n-4) + 2(2n^2 - 2n) = m + 4n^2 - 4 = \mathcal{O}(n^2)$ (directed) edges. Hence, the running time is $\mathcal{O}(VE^2) = \mathcal{O}(n^6)$. This upper bound is correct, but we can make it tighter. In part (a), we showed an upper bound $\mathcal{O}((E+V) \cdot |f^*|)$ for all FORD-FULKERSON-based algorithms, and in the particular case of the escape problem, $|f^*| \leq 4n - 4$ (why?) and $|E| + |V| = \mathcal{O}(n^2)$. Hence, a tighter bound for the running time is $\mathcal{O}((E+V) \cdot |f^*|) = \mathcal{O}(n^3)$.

Assaf Kfoury
Created: 99.02.25