CS 521: Data Structures and Algorithms I
Homework 3

Dustin Ingram

November 7, 2011

1. **Solution:**

   - **Adjacency-list:** For every vertex in a given vertex's adjacency-list, this algorithm traverses it's respective adjacency list (two edges away) to compute the square:

Adjacency-Square

```
for V ∈ G do
  for V′ ∈ G.Adj[V] do
    for V″ ∈ G.Adj[V′] do
      if V ≠ V″ then
        G[V].append(V″)
      end if
    end for
  end for
end for
```

   Because this algorithm operates on the adjacencies-of-adjacencies, the running time is given by $O(V^3)$.

   - **Adjacency-matrix:** Similar to the previous algorithm, this algorithm travels two hops away from a given vertex. Since an adjacency-matrix gives us a 1 if there is an edge and a 0 if there is not, we can use this to our advantage – if both are 1, we add 1 to $G(i, j)$, otherwise we add 0.

Matrix-Square

```
for i = 1 ... V do
  for j = 1 ... V do
    for j' = 1 ... V do
      if i ≠ j then
        G(i, j) = G(i, j) + (G(i, j') × G(j, j'))
      end if
    end for
  end for
end for
```

The three nested loops result in a running time of $O(V^3)$.

2. **Solution:** The entries of the matrix product BB^T (where B^T is the transpose of B and B is the incidence matrix of a directed graph $G = (V, E)$) are represented such that

$$|BB^T(i,j)| = \begin{cases} \text{Degree of } i & \text{if } i = j, \\ \text{Number of edges between } i \text{ and } j & \text{if } i \neq j. \end{cases}$$

3. **Solution:**

- **Adjacency-list:** The adjacency-list algorithm passes through each vertex V in $G(V, E)$, adding V to the adjacency list of G^T for each adjacent vertex:

Adjacency-Transpose

```
for V ∈ G do
  for V' ∈ G.Adj[V] do
    G^T[V'] = V
  end for
end for
```

This iterates through V vertices and each of their roughly $\frac{E}{V}$ adjacent vertices, for a total runtime of $O(V + E)$.

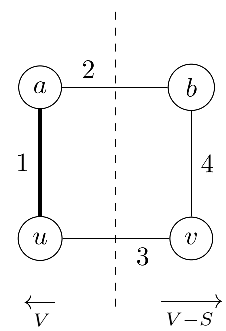
- **Adjacency-matrix:** Since G is represented by a square $V \times V$ matrix, this is simply the transposition of the matrix. Since this algorithm visits every element in the matrix, it runs in $O(V^2)$ time.

Matrix-Transpose

```
for i = 1 ... V − 1 do
  for j = i + 1 ... V do
    G^T(i, j) = G(j, i)
  end for
end for
```

If we do not need to produce a new matrix G^T, however, we can use G as G^T simply by inverting the indices, i.e. since $G(i,j) = G^T(j,i)$, we can produce any value in G^T in $O(1)$.

4. **Solution:** Here, (u,v) is a safe edge for A, as it will eventually become part of the MST of G, however, in this iteration (after (a,u) has been added to the MST) it is not a light edge (a minimum edge crossing S and $V - S$).



5. **Solution:** Decreasing the weight of an edge in T, the MST of G, still satisfies the two properties of an MST:

   (a) All vertices in G are connected in T';

   (b) The sum of the edges of T' are the minimum possible combination of edges to maintain (a).

   More formally, this definition maintains the edge weights of T if the edge is not (x,y) and if it is (x,y), subtracts k from this single edge. As long as $k > w(x,y)$, (thus making $w'(u,v)$ negative), this holds.

6. **Solution:** If the graph $G(E,V)$ already has a minimum spanning tree T computed, the speed at which we can update the MST if we add a new vertex v' and it's incident edges E' to G depends on two variables: the number of vertices in T, and the number of incident edges to be added.
   For a base case, consider that $E' = 1$, i.e., that a single vertex v' and a single edge to *some* vertex in T is added. Since this single edge is the only possible path to v', it must become a part of the MST.
   If there are more than one incident edges in E', any of these edges will induce a cycle in T (after the first minimum edge is added), with the maximally weighted edge in the cycle not necessarily being an incident edge. Additionally, there can be at most V incident edges (if there were one to every vertex V in T).
   If we add every incident edge in E' to the edges in T (which can be at most $V - 1$ edges), this results in a graph with at most $2V - 1$ edges. We can run either Kruskal's or Prim's algorithm on the resulting graph

to produce the MST, which will run in $O(E \lg V)$, but since we know E of this new graph can be no more than $2V - 1$, this is in fact $O(V \lg V)$.

7. **Solution:** We can use the figure from solution 4 for this problem as well. If we partition the graph G into subgraphs such that $a, u \in V_1$ and $b, v \in V_2$, recursively solving the MST for these subgraphs will result in adding the edge (a, u) to V_1 and similarly (b, v) to V_2. It is clear that adding (a, b), the light edge crossing the cut will not produce an MST.