CMPUT204
Department of Computing Science
University of Alberta

Assignment #4 Solutions
Due: Nov 24, 2004

---

**Question 1.** (15 marks) The *square* of an undirected graph $G = (V, E)$ is a new graph $G^2 = (V, E^2)$ where $(u, v) \in E^2$ if and only if there exists a $w \in V$ and $(u, w) \in E$ and $(w, v) \in E$. Describe efficient algorithms for computing $G^2$ if the graph was represented with (i) adjacency lists, and (ii) and adjacency matrix. You must also analyze the running time of your algorithm.

(i) We need to compute $E^2$. For each vertex look at the adjacency list, and take each of these vertices adjacency lists and union them together. A list union occurs for every entry in the adjacency list, which totals $2|E|$. Each list union will be linear in the length of the lists which could total no larger than $2|V|$. So the running time is $\Theta(|V||E|)$.

(ii) Let $E$ be the adjacency matrix with edges denoted by 1's and no edges denoted by 0's. Compute $E^2$ by multiplying the matrix $E$ with itself. Iterate over the entries setting any non-zero entry to 1. Running time is $\Theta(|V|^3)$, using naive matrix multiplication algorithms.

---

**Question 2.** (15 marks) A vertex $s$ of a *directed* graph $G = (V, E)$ is called a *sink* if for every vertex $v \in (V - \{s\})$, $(v, s) \in E$ and $(s, v) \notin E$. In other words, every vertex has an edge to $s$ and no edge from $s$. Write an algorithm that given a directed graph, finds a sink (or return that one does not exist) in only $O(|V|)$ time. The graph is given by its adjacency matrix $A$. Notice that a running time of $O(|V|)$ is remarkable given that the input to the procedure is $\Omega(|V|^2)$.

```
FindSink(V, E)
    i ← 1
    j ← 2
    while j ≤ |V| do
        if E(i, j) = 1 then i ← j
        j ← j + 1
    if j < |V| and IsSink(V, E, j) return j
    else return nil

IsSink(V, E, j)
    for i ← 1 to |V|
        if E(i, j) = false or E(j, i) = true then return false
    return true
```

---

**Question 3.** (20 marks)
**a.** (10 marks) Exercise 21.3-2 on page 509 of the textbook.

```
Find-Set(x)
    rx ← x
    while p[rx] ≠ rx do
        rx ← p[rx]
    while p[x] ≠ rx do
        tmp ← p[x]
        p[x] ← rx
        x ← tmp
    return rx
```

**b.** (10 marks) Exercise 21.3-3 on page 509 of the textbook.

Assume $n = 2^k$, call MAKE-SET($i$) for $i = 1 \ldots n$. For the remaining $m - n$ operations. Do $n$ unions of the pairs $(1,2),(3,4),(5,6),\ldots,(n-1,n)$. Then do $n/2$ unions of the pairs $(1,3),(5,7),(9,11),\ldots,(n-3,n-1)$. Then $n/4$ unions of the pairs $(1,5),(9,13),\ldots,(n-7,n-3)$. Continue until you union $(1,n/2)$. Repeat this union for the rest of the $m-n$ operations. If size of resulting tree will be $logn$, and the final union call will repeatedly cost $\Theta(logn)$ to find the and compare the representative elements. If $m$ is much larger than $n$ these calls will dominate to give $\Theta(mlogn)$.

---

**Question 4.** (15 marks) Consider the disjoint-sets abstract data type, but modified so that each member element also has an associated key. MAKE-SET($x,k$) creates a new element $x$ with key $k$. FIND($x$) returns the smallest key of any member in the same set as $x$. UNION remains unchanged. Write an efficient implementation of this modified version of disjoint sets. Compare its running time to that of the best implementation of basic disjoint sets.

```
MAKE-SET(x, k)
    p[x] ← x
    r[x] ← 0
    k[x] ← k

UNION(x, y)
    rx ← FINDREP(x)
    ry ← FINDREP(y)
    if r[rx] < r[ry] then
        p[rx] ← ry
        k[ry] ← min{k[ry], k[rx]}
    else
        p[ry] ← rx
        k[rx] ← min{k[ry], k[rx]}
        if r[rx] = r[ry] then r[rx] ← r[rx] + 1

FIND(x)
    rx ← FINDREP(x)
    return key[rx]

FINDREP(x)
    if p[x] = x then return x
    rx ← FINDREP(p[x])
    p[rx] ← rx
    return rx
```

---

**Question 5.** (20 marks)

**a.** (10 marks) Prove that every connected graph with $|V| > 2$ has at least two vertices such that if one or both were removed along with their incident edges, the resulting graph is connected.

Consider a spanning tree rooted at some vertex $v$ (such as might result from BFS). If $v$ has two children then the tree also has two leaves (one in each subtree), which can be removed without disconnecting the other vertices in the tree, which are a subset of the remaining graph edges. If $v$ has a single child then there's at least one leaf, which can be removed along with $v$, without disconnecting the other vertices.

**b.** (10 marks) Describe an efficient algorithm to find two such vertices.

Run BFS on some vertex $v$. Iterate the parent array, checking off vertices that are parents. There must be one leaf (the last vertex removed from the queue). If there are more than one, then return two. If not, then return the one leaf and $v$.