COMP 362 Assignment 2

Christopher Hundt 110220945

February 19, 2004

1. Assume $G = (V, E)$. Call CDFS(G) to run.

```
CDFS(G)
1  for each v ∈ V
2       do colour[v] ← WHITE
3  time ← 0
4  for each v ∈ V
5       do if colour[v] = WHITE
6            then if CDFS-VISIT(v) = FALSE
7                   then return "Directed Cycle!"
```

```
CDFS-VISIT(v)
1  colour[v] ← GRAY
2  for each u such that (v, u) ∈ E
3       do if colour[u] = WHITE
4            then CDFS-VISIT(u)
5            else if colour[u] = GRAY
6                   then return FALSE
7  colour[v] ← BLACK
8  return TRUE
```

Suppose there is a directed cycle in G. Then there is some $u, v \in V$ such that $u \rightsquigarrow v \rightsquigarrow u$. Suppose u is the first one of the two visited by CDFS. Then u is coloured gray and v is still white. CDFS-VISIT does not finish with u until it has visited every vertex that u is adjacent to, and hence every vertex that those vertices are adjacent to, etc. Thus CDFS-VISIT reaches v before finishing u, so u is coloured gray when CDFS-VISIT(v) starts. Then, because $v \rightsquigarrow u$, there is some w such that $v \rightsquigarrow w$ and $(w, u) \in E$. CDFS-VISIT(w) will be called before finishing with v, and then, on line 5, it will find that u is coloured gray and the cycle will be detected.

Now suppose that CDFS reports a cycle. Then CDFS-VISIT found a gray node u such that, for the current node v, $(v, u) \in E$. But CDFS-VISIT colours each node black when it finishes with it, so this means that CDFS-VISIT(v) was a recursive call, and CDFS-VISIT(u) has not finished yet. But since CDFS-VISIT(u) just follows every path from u, this means there is a path from u to v and an edge from v to u. Thus there is a cycle in the graph.

Thus, CDFS reports a cycle if and only if one exists.

2. (a) The uv-entry of $A^k(G)$ is the number of paths of length k (that is, k edges) from u to v. We can prove this by induction. For $k = 1$, $a_{uv} = 1$ if there is an edge from u to v, and 0 otherwise. But having an edge from u to v is precisely the only way there can be a path of length 1 from u to v. So the property holds for $k = 1$. So we assume, by induction that the property holds for $k = j$. Now, w consider a_{uv}^{j+1}, noting that $A^{j+1} = A^j A$.

$$\begin{aligned}
a_{uv}^{j+1} &= \sum_{w \in V} a_{uw}^{j} a_{wv} && (1)\\
&= \sum_{w \in V} (\#\text{ paths of length } j \text{ from } u \text{ to } w) \times (1 \text{ if } (w,v) \in E,\ 0 \text{ otherwise}) && (2)\\
&= |\{(u, v_1, v_2, \ldots, v_{j-1}, w, v) \mid (u, v_1), (v_{j-1}, w), (w, v) \in E \text{ and } (v_i, v_{i+1}) \in E \text{ for all } i < j-1\}|\\
&= \text{number of paths of length } j+1 \text{ from } u \text{ to } v.
\end{aligned}$$

Note that equation (1) comes from the definition of matrix multiplication, and equation (2) follows from the inductive hypothesis.

(b) Now, the following algorithm will find the length of the smallest cycle of G and the number of cycles of that length, given $n = |V|$.

```
CYCLES(A)
B ← I, the identity matrix of size n
k ← 0
while B ≠ 0
      do k ← k + 1
         B ← BA
         for i ← 1 to n
              do m ← m + a_ii / k
         if m > 0
            then return "m cycles of length k"
return "no cycles"
```

Suppose there are x cycles of length y, and y is the smallest length of any cycle. Then each of the x cycles will involve y vertices, so there will be xy ways for a vertex to follow a path of length y and end up back at itself. Thus, the total along the diagonal column of $A^y(G)$ will be xy. CYCLES will not return before $k = y$ because the diagonal entries will remain 0 for $A^j(G)$ where $j < y$, because y is the smallest length of any cycle. Once CYCLES gets to $k = y$, it will total the diagonal entries and get $m = xy/k = xy/y = x$, and return "x cycles of length y," the correct answer.

Now suppose there are no cycles in the graph. Then the algorithm will never find that any diagonal entries are greater than 0, so $m = 0$ always. However, with no cycles, there can be no paths of length $> |V|$. Thus, eventually $A^k(G) = 0$ for $k > |V|$, so the while loop will end, and the algorithm will terminate with the correct answer: "no cycles."

3. We note that, if $c(v) = x$ for $v \neq s$ and v not adjacent to s, then there is some u such that $(u, v) \in E$ and $c(u) = x$ or $w(u, v) = x$. In fact, for such a u, $c(v) = \min\{c(u), w(u, v)\}$ by definition of the weight of a path. Thus, we have

$$c(v) = \max\{\min\{c(u), w(u, v)\} \mid (u, v) \in E\}. \tag{3}$$

Now let us define $c^{(l)}(v) = \max\left\{c^{(l-1)}(v), \max\left\{\min\left\{c^{(l-1)}(u), w(u, v)\right\} \mid (u, v) \in E\right\}\right\}$. We know that, if a path p from s to v of maximum weight has weight x, then $c(v) = x$. Now we claim:

Claim 1 *If the maximum weight path from s to v has length (number of edges) no more than l, then $c^{(l)}(v) = c(v)$, with $c^{(l)}(v)$ defined as above, after adding (for convenience) that $c^{(k)}(s) = \infty$ for any k and $c^{(0)}(v) = 0$ for any $v \neq s$.*

We can prove this claim inductively. For $l = 1$, we have

$$\begin{aligned} c^{(1)}(v) &= \max\left\{c^{(0)}(v), \max\left\{\min\left\{c^{(0)}(u), w(u, v)\right\} \mid (u, v) \in E\right\}\right\} \\ &= \max\left\{0, \max\left\{\min\left\{c^{(0)}(u), w(u, v)\right\} \mid (u, v) \in E\right\}\right\}. \end{aligned}$$

Now the only way we can have a maximum weight path of length 1 from s to v is if $(s, v) \in E$. Then $\min\{c^{(0)}(s), w(s, v)\} = \min\{\infty, w(s, v)\} = w(s, v)$. But, for any other $u \sim v$, we have $\min\{c^{(0)}(u), w(u, v)\} = \min\{0, w(u, v)\} = 0 \leq w(s, v)$. Thus $c^{(1)}(v) = w(s, v) = c(v)$.

Now suppose that the claim is true for $l = k$. Then consider a maximum weight path p from s to v of length $k + 1$. Let $p = (s, v_1, v_2, \ldots, v_k, v)$ and let $q = (s, v_1, v_2, \ldots, v_k)$. Then $c^{(l)}(v_k) \geq w(q)$ by the induction hypothesis. Because p is a maximum weight path, $w(p) = c(v)$. But $w(p) = \min\{w(q), w(v_k, v)\}$. Thus, $c^{(l+1)}(v) \geq c(v)$. But clearly $c^{(l+1)}(v) \leq c(v)$ because it only uses actual paths to v. Therefore, $c^{(l+1)}(v) = c(v)$

Now consider the following algorithm.

```
Capacities(G, s)
1  for each v ∈ V \ {s}
2       do c[v] ← 0
3  c[s] ← ∞
4  for i ← 1 to |V| − 1
5       do for each (u, v) ∈ E
6              do if c[v] < min{c[v], w(u, v)}
7                    then c[v] ← min{c[v], w(u, v)}
```

Claim 2 *After l iterations of the for loop on line 4 of* Capacities, $c[v] = c^{(l)}(v)$.

This is easy to see, because Capacities is clearly just a direct implementation of the recursive definition of $c^{(l)}(v)$.

Thus, after $|V| - 1$ iterations of the for loop on line 4 of Capacities, $c[v]$ is the maximum weight path with at most $|V| - 1$ edges, so it is the maximum weight path globally.