ECS 122A Homework 5

Hardy Jones
999397426
Professor Gysel
Fall 2014

1. We can use the Floyd-Warshall algorithm to do the brunt of the work here. We just need to modify some parts of it, and perform some post computation.

   Given a graph G where

   $$G.V = \{C_1, C_2, \ldots, C_n\}$$

   $$G.E = \{H_{i,j} \mid C_i \text{ is connected to } C_j \text{ by a highway}\}$$

   We can construct most of the algorithm the same, and just change the final case in the recursive formula.

   $$D^{(k)}[i,j] = \begin{cases} 0 & \text{if } k = 0 \text{ and } i = j \\ t(H_{i,j}) & \text{if } k = 0 \text{ and } H_{i,j} \in G.E \\ \infty & \text{if } k = 0 \text{ and } i \neq j \text{ and } H_{i,j} \notin G.E \\ min(D^{(k-1)}[i,j], D^{(k-1)}[i,k] + t(C_k) + D^{(k-1)}[k,j]) & \text{if } k \geq 1 \end{cases}$$

   Using this formulation, we construct FLOYD-WARSHALL-MODIFIED similar to the method for constructing FLOYD-WARSHALL in class – taking care to use the modified cases where appropriate. The runtime of this algorithm has not changed, as the nested update function is still $O(1)$. So the complexity is still $\Theta(n^3)$

```
function FLOYD-WARSHALL-MODIFIED(G, t)
    n ← |G.V|
    Create n × n array D^(0)
    for i ← 1 to n do
        for j ← 1 to n do
            if i = j then
                D^(0)[i, j] = 0
            else if H_{i,j} ∈ G.E then
                D^(0)[i, j] = t(H_{i,j})
            else
                D^(0)[i, j] = ∞
            end if
        end for
    end for
    for k ← 1 to n do
        Create n × n array D^(k)
        for i ← 1 to n do
            for j ← 1 to n do
                D^(k)[i, j] = min(D^(k−1)[i, j], D^(k−1)[i, k] + t(C_k) + D^(k−1)[k, j])
            end for
        end for
    end for
    return D^(n)
end function
```

Since this gives us back an $n \times n$ array with just the times from one city to another (including travel times between cities), we still need to add on the time necessary to travel from each depot to the highway.

For each i, j entry in the array, where $i \neq j$, we need the time from the depot to the highway in C_i and the time from the highway to the depot in C_j. Then we can just return the array.

Our completed algorithm is:

```
function MOVE-FREIGHT(G, t, d)
    n ← |G.V|
    D ← FLOYD-WARSHALL-MODIFIED(G, t)
    for i ← 1 to n do
        for j ← 1 to n do
            if i ≠ j then
                D[i, j] = D[i, j] + d(C_i) + d(C_j)
            end if
        end for
    end for
    return D
end function
```

Since the added loops only contribute $\Theta(n^2)$ time to the algorithm, the runtime of MOVE-FREIGHT is governed by the call to FLOYD-WARSHALL-MODIFIED. Thus, the runtime of this algorithm is $\Theta(n^3)$.

2. (a) Given a graph G, a size k and a certificate $S \subseteq G.V$, we need to check two things:

   i. $|S| = k$

   ii. $\forall (i, j) \in G.E; i \in S, j \in S$, or $\{i, j\} \subseteq S$

   2(a)i can be checked in $O(|S|)$ time by counting the size of S.

   2(a)ii can be checked in $O(|G.E|)$ time by iterating over $G.E$ and checking each vertex for membership in S.

   These are both linear times, so our verification algorithm runs in polynomial time. Thus VERTEX-COVER-DEC $\in$ **NP**.

```
function VERIFY-VERTEX-COVER(G, k, S)
    result ← True
    if |S| ≠ k then
        result ← False
    else
        for all (i, j) ∈ G.E do
            if i ∉ S or j ∉ S then
                result ← False
            end if
        end for
    end if
    return result
end function
```

   (b) Given a graph G, a size k and a certificate $M \subseteq G.E$, we need to check two things:

   i. $|M| = k$

ii. $\forall v \in G.V$ at most one edge of M is incident to v

2(b)i can be checked in $O(|M|)$ time by counting the size of M.

2(b)ii can be checked in $O(|M||G.V|)$ time by keeping track of the incident edges.

These are both linear times, so our verification algorithm runs in polynomial time. Thus BIPARTITE-MATCHING-DEC $\in$ **NP**.

```
function VERIFY-BIPARTITE-MATCHING(G, k, M)
    result ← TRUE
    if |M| ≠ k then
        result ← FALSE
    else
        for all v ∈ G.V do
            incident ← FALSE
            for all (i, j) ∈ M do
                if (v = i or v = j) and incident then
                    result ← FALSE
                else if v = i or v = j then
                    incident ← TRUE
                end if
            end for
        end for
    end if
    return result
end function
```

3. (a) True

   We can look through each of the subproblems that are possible, find an optimal one of the subproblems by testing each with the decision problem and only exploring the optimal subproblem we chose.

   (b) True

   Take a problem instance in A, reduce it to B in polynomial time, then run the polynomial algorithm in B. The overall runtime is still polynomial

   (c) False

   Since $\forall x \in \mathbf{NP}, x \leq_P 3SAT$; $3SAT \in \mathbf{NP\text{-}Complete}$; $\forall x, y, z \in \mathbf{NP}, x \leq_P y$ and $y \leq_P z \implies x \leq_P z$; and $\mathbf{P} \subseteq \mathbf{NP}$ that conjecture would imply that any problem in $\mathbf{P}$ has no polynomial time algorithm. But this is clearly false, as $\mathbf{P} \neq \emptyset$.