Student Name: Duc Tu Luong
Last 3-digit ID: 122

# Homework #7

**1.** a. Problem model:
k network routes from one source to one destination
n files to be transfer from the source to the destination
each file fi has length li, the transfer speed along all k paths are identical.
How to assign files to each of the k routes so that to finish the transfer of all n files in the shortest time?

b. This problem belongs to class P

c. Polynomial-time algorithm:
- This problem can be solved approximately in polynomial time by load balancing technique. We apply LPT-scheduling-list algorithm to find the schedule to send file via 3 paths.
- Pseudo code:

```
LPT-Greedy-Scheduling(m, n, t1, t2,..., tn) {
    Sort jobs so that t1 ≥ t2 ≥ ... ≥ tn
    for i = 1 to m {
        Li ← 0
        J(i) ← ϕ
    }
    for j = 1 to n {
        i = argmin_k Lk
        J(i) ← J(i) ∪ {j}
        Li ← Li + tj
    }
    return J(1),..., J(m)
}
```

- Example, given 10 files, their lengths are: 5, 6, 7, 3, 2, 10, 9, 8, 6, 4, and 3 paths, the optimal solution is to transfer them in 20/speed time unit.
First, we sort 10 files in decreasing order of lengths: 10, 9, 8, 7, 6, 6, 5, 4, 3, 2, after applying to LPT-Greedy-Scheduling, the schedule is as following:
Path 1: 10, 6, 3, 2
Path 2: 9, 6, 4,
Path 3: 8, 7, 5,
=> approximate solution is 21
- Time complexity of this algorithm is O(nlogn), n is the number of files.
- Approximation guarantee: P = 21/20 = 1.05

**2.** a. Problem model:
Consider Multiple Interval Scheduling problem. A processor is available for a specific period and can only run one job at a time. Given n jobs, each job has a set of intervals of time during which it needs to use the processor. For a given number k, is it possible to accept at least k of the jobs so that no two of the accepted jobs have any overlap in time?

b. This problem belongs to NP-Complete class

c. Proof:
- This problem is in NP class, because given a set of k intervals, we can check in polynomial time that is there any of them overlap or none.
- We consider Independence Set (IS) problem, which is in NP-Complete class. IS problem is specified by a graph G = (V, E) and a number k.
- Now we prove that IS $\leq_P$ Multiple Interval Scheduling problem.
Let us assume there are n vertices and m edges in graph G, where each job is represented by a vertex and each time interval is represented by an edge. Two nodes are the endpoints of an edge if two corresponding jobs need that time interval to run on the processor. We can conclude that no 2 jobs can be both accepted if they are the endpoints of some edge. Therefore, if there exists an independence set of size k in G, then there exists a subset of size k of n jobs without any overlap jobs, and vice versa. Thus, we can conclude that this problem belongs to NP-Complete class.

**3.** a. Problem model:
Given n airports, m direct flights scheduled among these airports. The aim is to select a subset of airports in n to build service facility, the cost to build such service facility at each site is different, specified as Ci for airport site Ai. The goal is to select a subset of airport sites with minimized cost and ensure all flights are served.

b. This problem belongs to class NP-Complete

c. Proof:
- We call this problem Airport Service Cover problem and it belongs to NP class, because given a subset of n airports which are chosen to build service facility, we can check in polynomial time that whether this subset of airport sites is built with minimized cost and ensure all flights are covered.
- We consider Weighted Vertex Cover problem which is in NP-Complete class.
- Now we need to prove that Weighted Vertex Cover $\leq_P$ Airport Service Cover problem.
Let us assume that there is a graph G = (V, E), where each vertex represents airport site, each edge represents direct flight, if there is a direct flight between 2 airport sites, we draw an edge between 2 corresponding vertices, and a number k. Now we create an instance of Airport Service Cover problem, we consider following case: instance of Airport Service Cover problem returns “Yes”, meaning there exists a subset of airport sites of size k with minimized cost and all flights covered; if and only if there exists a

subset of size k of minimized weighted vertex cover in graph G; and vice versa. Thus, we conclude that this problem belongs to NP-Complete class.

**4.** a. Problem model:
Telecommunication company needs to base stations in various location, the distance to a based station affects the communication quality. Given n available different locations, select k locations to build k base stations so that the maximum distance from any house to its closest station is minimized.

b. This problem belongs to class NP-Complete

c. We can solve this problem approximately in polynomial time by using greedy algorithm, the solution is as following:
- We start by selecting any location in n available locations, and add it to our set of chosen locations. Then, we select the farthest location from our set, and add it to our set, repeat k times and we will have our k locations to build k base stations so that the maximum distance from any house to its closest station is minimized.
- Pseudo code:

GREEDY-CENTER-SELECTION (k, n, s1, s2…, $s_n$)
    C ← ∅
    REPEAT k times
        Select a site $s_i$ with maximum distance dist($s_i$, C)
        C ← C ∪ $s_i$
RETURN C

- Time complexity: O(logn), n is number of all available locations
- This solution provide approximation guarantee of P = 2

**5.** a. Problem model:
Given a collection of n job requests to select from, each job i pays $p_i$ amount of reward, there a (n – 1) pairs of conflict jobs, and no job that is free of conflict with other jobs. The goal is to select the set of no-conflict jobs and maximized the total amount of reward.

b. This problem belongs to class P

c. Solution:
- We construct a graph G = (V, E), each vertex in G represents each job in n jobs and weight of each vertex represents reward of corresponding. With each pair of conflict jobs, we draw an edge between 2 corresponding vertices. Now we have a graph G with n nodes, (n – 1) edges, our goal is to find the maximum weighted independence set of nodes.

- We use dynamic programming to solve this problem, the pseudo code is as following:

```
Weighted-Independent-Set-In-Tree(T) {
    Root the tree at a node r
    For each (node u of T in post-order) {
        if (u is a leaf) {
            M_in [u] = w_u
            M_out[u] = 0
        } else {
            M_in [u] = Σ_{v∈children(u)} M_out[v] + w_v
            M_out[u] = Σ_{v∈children(u)} max(M_out[v], M_in[v])
        }
    }
    return max(M_in[r], M_out[r])
}
```

- Time complexity: O(n), n is number of nodes in G

**6.** a. Problem model:
In a city with m roads and n crossroad points, only 10 security cameras are installed at some selected crossroad to cover all roads. Is it possible to place ≤ 10 security cameras at the crossroads that cover all roads in the city.

b. This problem belongs to class NP-Complete

c. Proof:
- We call this problem Roads Cover problem and it is in NP class, because given a set of crossroad points where security cameras are installed, we can check in polynomial time that whether all roads in the city are covered or not.
- We consider Small Vertex Cover problem, which is in NP-Complete class. Small Vertex Cover is specified by a graph G = (V, E) and number k = 10.
- Now we need to prove that Small Vertex Cover problem $\leq_P$ Roads Cover problem. Let us assume that there are n vertex and m edges in graph G, where each crossroad is represented by a node, and each road is represented by an edge, if 2 crossroad points share a road, we connect their corresponding vertex by an edge. We initialize an instance of Roads Cover problem, now we consider following case: if instance of Roads Cover problem returns “Yes”, meaning there exists a subset of n crossroad points ≤ 10, where security cameras are installed to cover all roads in the city if and only if there exists a small vertex cover with size ≤ 10 in graph G; and vice versa. As a result, we can conclude that this problem belongs to NP-Complete class. The algorithm to find small vertex cover in a graph G is as following:

```
Vertex-Cover(G, k) {
    if (G contains no edges)
        return true
    if (G contains ≥ k * n edges)
        return false
    let (u, v) be any edge of G
    a = Vertex-Cover(G - {u}, k-1)
    b = Vertex-Cover(G - {v}, k-1)
    return a or b
}
```

- Time complexity: Small Vertex Cover problem with size ≤ k takes $O(2^k kn)$ time