CS 331: Algorithms and Complexity (Spring 2015)

Unique numbers: 51825, 51830, 51835, 51840

Assignment 6

Due on: Wednesday, March 25^{th}, by 11.59pm

Q1 [10 points] Given integers $a_1, \ldots, a_n$, design a dynamic programming algorithm that determines whether there exists a partition of the numbers into 3 disjoint subsets P, Q, R such that the sum of the numbers in each set are equal. In other words,

$$\sum_{a_i \in P} a_i = \sum_{a_i \in Q} a_i = \sum_{a_i \in R} a_i$$

For example-

- Given $\{1, -4, 5, 11, -2, 4\}$, it is possible to define the partitions as $P = \{1, 4\}$, $Q = \{5\}$ and $R = \{11, -2, -4\}$.
- Given $\{1, 2, 3, 9\}$, it is not possible to parition into three equal-weight sets.

Sol. Let $W = \sum_{i \in [n]} |a_i|$ be the upper bound of the summation. In the dynamic programming, we use $f(i, s_1, s_2, s_3, b_1, b_2, b_3) \in \{True, False\}$ that $i \in [n], s_1, s_2, s_3 \in [-W, W]$ and $b_1, b_2, b_3 \in \{True, False\}$ to denote whether there exists a partition of $a_1, \cdots, a_n$ into $\{P, Q, R\}$ such that $\sum_{i \in P} = s_1, \sum_{i \in Q} = s_2, \sum_{i \in R} = s3$ and $b_1, b_2, b_3 = True$ to indicate whether P, Q, R are not empty respectively.

Base Case: $f(0, 0, 0, 0, false, false, false) = true$ and the rest of $f(0, *, *, *, *, *, *)$ are false.

Recurrence formula:

$$\begin{aligned} f(i, s_1, s_2, s_3, b_1, b_2, b_3) = \quad & f(i-1, s_1 - a_i, s_2, s_3, T, b_2, b_3) OR f(i-1, s_1 - a_i, s_2, s_3, F, b_2, b_3) \\ OR \quad & f(i-1, s_1, s_2 - a_i, s_3, b_1, T, b_3) OR f(i-1, s_1, s_2 - a_i, s_3, b_1, F, b_3) \\ OR \quad & f(i-1, s_1, s_2, s_3 - a_i, b_1, b_2, T) OR f(i-1, s_1, s_2, s_3 - a_i, b_1, b_2, F) \end{aligned}$$

The final answer would be $f(n, \sum_{i \in [n]} a_i/3, \sum_{i \in [n]} a_i/3, \sum_{i \in [n]} a_i/3, True, True, True)$. It is not difficult to implement the dynamic programming by first enumerating i from 1 to n then enumerating $s_1, s_2, s_3, b_1, b_2, b_3$ for all possibilities. The running time of this dynamic programming is $O(nW^3)$.

Remark: it is not difficult to save some time and space by removing s_3 because $s_1 + s_2 + s_3 = \sum_{j \leq i} a_j$, which reduce the running time to $O(nW^2)$.

Q2 [10 points] Given positive integers n and W, design a dynamic programming algorithm that finds the number of possible n element sets $\{x_1, \ldots, x_n\}$ for which $\sim_i x_i^2 = W$, where each x_i is a non-negative integer.

For example, if $n = 3$ and $W = 45$, then there are the following 2 sets of 3 elements $\{0, 3, 6\}$, $\{2, 4, 5\}$ which satisfy the condition.

Sol. Let $f(i, j, S)$ denote the **number** of subsets with size j in $\{0, \cdots, i\}$, whose quadratic summation is S.

Base Case: $f(0, 0, 0) = 1$ and $f(0, 1, 0) = 1$ because of $\emptyset$ and $\{0\}$, $f(0, *, *) = 0$ for the rest.

Recursion formula: $f(i, j, S) = f(i-1, j, S) + f(i-1, j-1, S - i^2)$, there are only two possibilities of the subsets whose quadratic summation is S: either contains i or does not include i.

Pseudocode:

1. $f(0,0,0) = 1, f(0,1,0) = 1, m = [\sqrt{W}]$
2. For $i = 1$ to m
3.   For $j = 1$ to n
4.   for $S = 0$ to W
5. $f(i,j,S) = f(i-1,j,S) + f(i-1,j-1,S-i^2)$
6.   END for S
7.   END for j
8. END For i
9. Return $f(m,n,W)$

The running time of this algorithm is $O(mnW) = O(nW^{1.5})$.

Q3 [5 points] Q7.1 from textbook

Sol. (a) It is 2. There are three ways to cut the graph:

1. $\{s\}, \{u,v,t\}$
2. $\{s,u,v\}, \{t\}$
3. $\{s,v\}, \{u,t\}$

(b) It is 4. $\{s,v\}$ and $\{u,t\}$.

Q4 [5 points] Q7.2 from textbook

Sol. (a) It is 18. No, it is not a maximum flow.

(b) There are several min-cut in the network, for example, one side is s and the the top blank vertices, the other side is the rest of vertices. The capacity of min-cut is $8+5+3+5 = 21$ (another solution may be one side is s and the three blank vertices, the other side is d,t).

Q5 [10 points] Q7.7 from textbook

Sol. We construct a graph $G = (V,E,C)$ as follows:

1. $V = \{s, v_1, \cdots, v_n, u_1, \cdots, u_k, t\}$ where s is the source, $v_1, \cdots, v_n$ correspond to the n clients, $u_1, \cdots, u_k$ correspond to the k base stations and t is the terminal.
2. There are n edges from s to $v_1, \cdots, v_n$ separately, each edge with capacity 1.
3. There are k edges from $u_1, \cdots, u_k$ to t separately, each edge with capacity L.
4. There is an edge from v_i to u_j with capacity 1 if and only if the distance from client i to base j is within r.

Then we apply the max-flow algorithm to find the max flow in G. If the max-flow equals to n, we claim there is a solution and find it as follows: assign each client i to the base j when there is one unit flow on the edge (u_i, v_j). Otherwise, the max-flow is less than n, we claim there is no solution to make every client can be connected simultaneously. The running time of this algorithm is polynomial in n and k.