CS170–Fall 2022 — Homework 5 Solutions

Deming Chen cdm@pku.edu.cn

Last Modified: January 8, 2023

Collaborators: NONE

2 Bounding Sums

An exponential function satisfies that equality. For example,

$$f_1(n) = 2^{n-1}. \tag{1}$$

Then we have

$$\sum_{i=1}^{n} f_1(i) = 1 + 2 + \cdots + 2^{n-1} = 2^n - 1 = \Theta(f_1(n)). \tag{2}$$

The equality does not hold for a constant function, for example, $f_2(n) = 1$.

3 True and False Practice

(a) It is true that Kruskal's works with negative edges. Since the number of edges in a spanning tree is fixed $(n-1)$, adding a constant to every edge's weight does not influence the final outcome. So we can add

$$\max_{e \in E}\{|w(e)|\} \tag{3}$$

to the weights of all edges. Then all edges are non-negative.

(b) The statement is false, since the lengths of the paths are not the same. For instance, see Figure 1. Before modifying G, the shortest path from S to B is $S \to A \to B$. After adding 2 to all edges, the shortest path becomes $S \to B$.

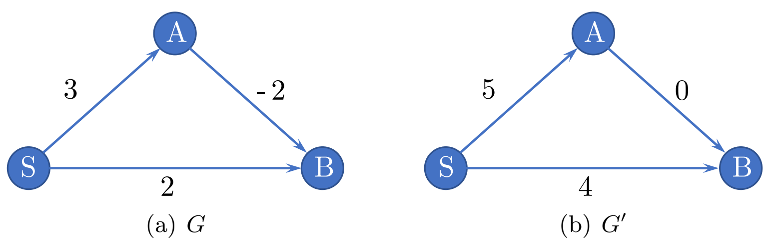

Figure 1: (a) a graph G with negative edges; (b) modified graph G'. Their shortest path from S to B is different.

(c) The statement is false. Just as shown in Figure 1 (b) (see the weight of edge AB as a small positive number ε). The increasing order of their distance from S is S, B, A, which is not a valid topological sort.

(d) The statement is false. For example, in Figure 2, initially we have two vertices a and b, so the MST is unique $\{\{a, b\}\}$. After u is added to the graph, the new MST is $\{\{u, a\}, \{u, b\}\}$.

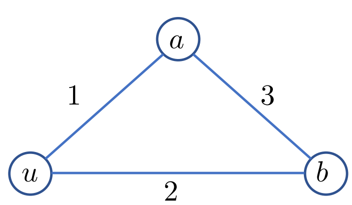

Figure 2: After u is added to the graph, the original MST is not a subset of the new MST.

4 Agent Meetup

(a) As shown in figure 3, the rectangle can be divided into 8 regions. It is easy to prove that, up to one agent can fit in a region (including its boundary). As a result, we can fit up to 8 agents within this rectangle.

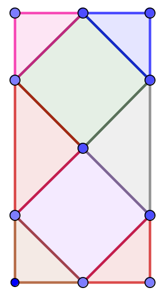

Figure 3: The rectangle are divided into 8 regions.

(b) The main idea is that, divide all agents into two groups according to their y-coordinates. We first find the minimum Manhattan distance between two agents within each group, and check whether there are two agents in different groups between which the distance is even shorter.

Algorithm 1. Find Minimum Manhattan Distance

`function FMMD`($P = \{(x_i, y_i)\}_{i=1}^{n}$)
`Input:` a set P of n points (x_i, y_i), where $x_i \in \mathbb{Z}, y_i \in \mathbb{Z}$ $(i = 1, 2, \cdots, n)$
`Output:` the minimum Manhattan distance between any two points

$P_1, P_2, y_{\mathrm{p}} = \texttt{Divide}(P)$
$d_1 = \texttt{FMMD}(P_1)$, $d_2 = \texttt{FMMD}(P_2)$, and $d = \min\{d_1, d_2\}$
Construct $\tilde{P} = \{(x, y) \in P | y_{\mathrm{p}} - d \leqslant y \leqslant y_{\mathrm{p}} + d\}$
Sort points in $\tilde{P}$ by x-coordinate, and rename them so that $x_1 \leqslant x_2 \leqslant \cdots \leqslant x_m$
$d_{\min} \leftarrow d$
For $i = 1$ to $m - 1$:
    $j = i + 1$
    while $x_j < x_i + d$:
        $d_{\min} \leftarrow \min\{d_{\min}, |x_i - x_j| + |y_i - y_j|\}$
return $d_{\min}$

The `Divide` procedure divides points set P into two sets such that y-coordinates of all points in P_1 is no larger than y_{p}, and y-coordinates of all points in P_2 is larger than y_{p}. This procedure takes time $\mathcal{O}(n)$.

Now we prove that Algorithm 1 is correct. Let's use $\mathbf{x}_1$ and $\mathbf{x}_2$ to denotes the shortest pair of points. There are three scenarios: both points are in P_1, both points are in P_2, and two points in different groups. $\texttt{FMMD}(P_1)$ and $\texttt{FMMD}(P_2)$ cover the first two scenarios. If it is the third scenario, the minimum distance must be less than what we get in the first two scenario, i.e. d. So, the difference between their x-coordinates is less than d. After checking all such pairs of points, we can get the Minimum Manhattan Distance.

In the end, we analysis the run-time of Algorithm 1. The Divide procedure takes time $\mathcal{O}(n)$. The sorting procedure takes time $\mathcal{O}(n \log n)$. The outer loop has $\mathcal{O}(n)$ iterations. Let's consider the inner loop. As shown in Figure 4, according to (a), there are at most 8 points in the green rectangle, and at most 8 points in the red rectangle, so the inner loop only compares up to 16 pairs of points. So, those loops take time $\mathcal{O}(n)$. We have

$$T(n) \leqslant 2T(n/2) + C \cdot n \log n. \tag{4}$$

As a result,

$$\begin{aligned} T(n) &\leqslant 2T\left(\frac{n}{2}\right) + C \cdot n \log n \\ &\leqslant 2\left[2T\left(\frac{n}{4}\right) + C \cdot \frac{n}{2} \log\left(\frac{n}{2}\right)\right] + C \cdot n \log n \\ &\leqslant 4T\left(\frac{n}{4}\right) + 2C \cdot n \log n \\ &\leqslant 8T\left(\frac{n}{8}\right) + 3C \cdot n \log n \\ &\leqslant \cdots \\ &\leqslant nT(1) + kC \cdot n \log n, \end{aligned} \tag{5}$$

where $k = \log_2 n$. So, $T(n) = \mathcal{O}(n \log^2 n)$.

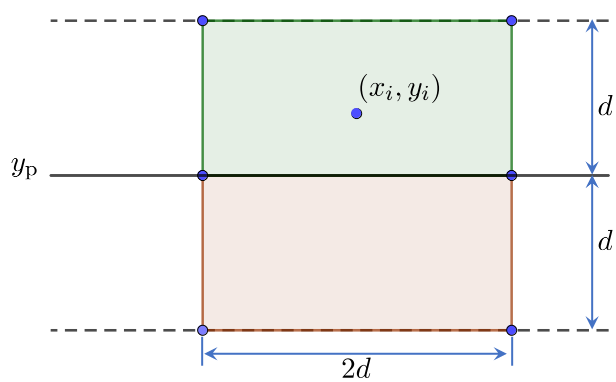

Figure 4: Caption

5 Money Changing

(a) If one able to do this for all integers $A > 0$, 1 must be one of these denominations, otherwise 1 cannot be expressed. If 1 is one of denominations, all positive integers can be expressed as so.

(b) A greedy algorithm is to use large denominations as many as possible. Without loss of generality, assume that $x_1 > x_2 > \cdots > x_n$.

Algorithm 2. Change Money

`procedure Change`(X, A)
`Input:` $X = \{x_1, x_2, \cdots, x_n\}$ `is a set of denominations where` $x_1 > x_2 > \cdots > x_n$`;` A `is a positive integer`
`Output:` n `non-negative integers` $a_1, a_2, \cdots, a_n$.

$a_1 = \lfloor A/x_1 \rfloor$
$a_2, a_3, \cdots, a_n =$ `Change`$(X \backslash \{x_1\}, A - a_1 x_1)$
`return` $a_1, a_2, \cdots, a_n$

(c) Suppose the optimal solution is $A = 25a_1 + 10a_2 + 5a_3 + a_4$. a_4 must be less than five, otherwise we can replace five *1*s with one *5*. Similarly, $a_3 \leqslant 1$. Thus, a_3 is either 0 or 1. If $a_3 = 1$, then $a_2 \leqslant 1$, otherwise, one *25* can replace two *10*s and one *5*. In this case, $10a_2 + 5a_3 + a_4 < 10 \times 1 + 5 \times 1 + 5 < 25$. If $a_3 = 0$, $a_2 \leqslant 2$, otherwise one *25* and one *5* can replace three *10*s. In this case, $10a_2 + 5a_3 + a_4 < 10 \times 2 + 5 \times 0 + 5 = 25$. All in all, $10a_2 + 5a_3 + a_4 < 25$, so $a_1 = \lfloor A/x_1 \rfloor$.

Since $5a_3 + a_4 < 5 \times 1 + 5 = 10$, $a_2 = \lfloor (A - 25a_1)/10 \rfloor$. Similarly, $a_3 = \lfloor (A - 25a_1 - 10a_2)/5 \rfloor$, and $a_4 = A - 25a_1 - 10a_2 - 5a_3$.

Therefore, Algorithm 2 gives the optimal solution.

(d) Consider denominations 1, 10 and 15. If $A = 20$, the optimal solution is $a_1 = a_3 = 0$ and $a_2 = 2$. However, the output of Algorithm 2 is five *1*s and one *15*.

6 Box Union

Let's consider a stack as a tree in a disjoint forest and a box as a node in a tree. For convenience, we call the number of boxes under a box the height of that box. Besides a parent pointer and its *size* number (the number of nodes in its subtree), a node also have a number h representing the height difference between itself and its parent. If a node is the root, its h number just represents its height. In addition, we suppose `under` operation does not only return the number of boxes under that box in its stack, but also returns the root of its tree.

```
operation under(x)
H = h(x)
if π(x) ≠ x:
    H_π, r = under(π(x))
    H ← H + H_π
    π(x) ← r
    h(x) ← H − h(r)
return H, r
```

The `put` operation is similar to union by size:

```
operation put(x,y)
H_x, r_x = under(x)
H_y, r_y = under(y)
if size(r_x) > size(r_y):
    π(r_y) ← r_x
    size(r_x) ← size(r_x) + size(r_y)
    h(r_y) ← h(r_y) − h(r_x) − size(r_y)
    h(r_x) ← h(r_x) + size(r_y)
else:
    π(r_x) ← r_y
    size(r_y) ← size(r_x) + size(r_y)
    h(r_x) ← h(r_x) + size(r_y) − h(r_y)
```