15-1 Bitonic Euclidean Traveling Salesman Problem

1. Sort the points using an O(nlogn) sort from the smallest to largest x-coordinate. Denote them as $P_1, P_2, ...P_n$.

2. Define a bitonic walk from P_i to P_j as a walk that starts from P_i, goes strictly from right to left to P_1, then goes strictly left to right to P_j, and passes every points between P_1 and $P_{\max(i,j)}$ exactly once. Denote $d(i, j)$ as the shortest bitonic walk from P_i to P_j. Due to symmetry, we only consider the situation that $i \geq j$.

Also use $\Delta(i, j)$ to denote the straight-line distance from P_i to P_j.

3. We need to compute $d(n,n)$. Look at the last two connected points in the shortest bitonic walk from P_n to P_n. It can go from P_1 to P_n, or P_2 to P_n,…, or P_{n-1} to P_n. Then

$$d(n,n) = \min\{d(n,1) + \Delta(n,1), d(n,2) + \Delta(n,2), ..., d(n,n-1) + \Delta(n,n-1)\}$$

4. To compute $d(i, j)$ for $i \geq j$, the base case is $d(2,1) = \Delta(2,1)$.

i) if $j < i-1$, the edge (P_i, P_{i-1}) must be in the shortest bitonic walk from P_i to P_j.

Then $d(n,n) = d(i-1, j) + \Delta(i, i-1)$

ii) if $j = i-1$. The similar idea in 3 applies on this case, but we look at he first two connected points in the shortest walk. It can be from P_i to P_1, or P_i to P_2,…, or P_i to P_{i-2}.

Then $d(i,i-1) = \min\{d(1,i-1) + \Delta(i,1), d(2,i-1) + \Delta(i,2), ..., d(i-2,i-1) + \Delta(i,i-2)\}$

Note that $d(1,i-1) = d(i-1,1)$ …

In order to track the shortest bitonic walk from P_n to P_n. We use $\pi(i)$ to record how we get the value $d(i,i-1)$

15-2 Printing Neatly

Given n words of lengths $l_1, l_2, ..., l_n$.
Use $c(i)$ to denote the minimum cost for printing words i through n (The minimum cost is defined as the minimum sum, over all lines except the last, of the cubes of the numbers of extra space characters at the ends of lines)
We need to compute $c(1)$.
For computing $c(i)$, we can put 1 or 2 or 3 … or p words in the first line, then printing from the (i+1)th, or the (i+2)th, …, or the (i+p+1)word through the n-th words neatly, where p is the maximum number of words starting from i-th word that can be fitted into a line.

Then $c(i) = \min_{i<j\leq i+p} (c(j+1) + M - j + i - \sum_{h=i}^{j} l_h)$

If starting from k-th word, the words from k to n can be fitted into a line. Then $c(i) = 0$, for $i >= k$.

The time complexity is $O(n^2)$, and the space complexity is $O(n)$.

16-1 Coin Changing

a. Grab as many quarters as possible. (i.e until the remainder change is less than 25).

Then grab as many dimes as possible, grab as many as dimes as possible, then grab as many nickels as possible, and complete by adding pennies.

Proof of correctness:

Assume our greedy algorithm generated an optimal solution S and there is a better solution called S'. We look at the number of coins in S and S', starting from the largest denomination. Because S' < S, then at some denomination i, $S_i' < S_i$. (S_i, S_i' is the number of coins for denomination i in the corresponding solution. S_i' can not be larger than S_i according to our strategy of selecting coins for denomination i) . In this case, in order to compensate for 1 coin in denomination i, S' has to select at least some number of coins in denomination i+1. For every two consecutive denominations in this problem, that is, 25 and 10, 10 and 5, 5 and 1. S' has to select at least two coins in denominations i+1 to compensate one coin in denomination i. This will bring one more coin. Therefore, the total number of coins in S' can not be less than that of S.

b. Similar idea in the proof of a. When c > 1 and $k \geq 1$. In order to compensate one C^{i+1}, at least c c^i coins need to be selected and c is at least 2. Then there can be a better solution.

c. Denomination (8,5,1) for changing of 10. Our greedy algorithm will generate one 8, and two 1. The optimal solution is two 5.

d. Use dynamic programming.

Given the denominations $\{d_1, d_2, ..., d_k\}$, and make changes for n cents.

Denote $c(n)$ as the minimum number of coins for making changes of n cents.

We can first select one coin of d_i , as long as $d_i \leq n$. We do not know in order to achieve the optimal solution, which d_i to choose, but we have at most k possibilities.

After choosing a coin of d_i, we find the optimal solution for $n - d_i$.

The optimal solution for n cents is just to take the minimum one of these k possibilities.

The recursion is as follows:

$$c(n) = \min_{i:d_i \leq n} \{1 + c(n - d_i)\}$$

We just record which d_i to choose for every step.

The time complexity is $O(nk)$, and the space complexity is $O(n)$.

16-2 Scheduling to minimize average completion time

a. Run tasks in ascending order of their completion time.

b. Use a priority queue in which tasks are sorted in ascending order of their remaining processing time, that is, the time they need to be completed.

Once a task is ready to run, if its completion time is less than the task being running, suspend current task, run the new one. Otherwise, put it into the priority queue and wait for running. Once a task is finished, choose the first task in the priority queue to run.