# Homework #2 Solution

Contact TAs: ada@csie.ntu.edu.tw

## Problem 1

(1) (5%)

By Pascal's triangle, we have the recurrence relation:

$$\binom{n}{m} = \begin{cases} 1 & , m = 0 \text{ or } m = n \\ \binom{n-1}{m-1} + \binom{n-1}{m} & , 0 < m < n \\ 0 & , \text{otherwise} \end{cases}$$

Filling the table directly gives an $O(N^2)$ algorithm.

```
int dp[N+1][N+1];
for(int i=0; i<=N; i++)
{
  for(int j=0; j<=i; j++)
  {
    if(j == 0 || j == i)
      dp[i][j] = 1;
    else
      dp[i][j] = dp[i-1][j-1] + dp[i-1][j];
  }
}
```

(2) (7%)

Let $f(n, m)$ be the number of valid ways to go from $(0, 0)$ to $(n, m)$, then $C_n = f(n, n)$.

Initial condition is $f(0, 0) = 1$. For other valid $(n, m)$, any path must have the last step come from either $(n-1, m)$ or $(n, m-1)$.

So we got the following recurrence relation:

$$f(n, m) = \begin{cases} 1 & , n = m = 0 \\ 0 & , n < 0 \text{ or } m < 0 \text{ or } n < m \\ f(n-1, m) + f(n, m-1) & , \text{otherwise} \end{cases}$$

Notice that we can write $f(n, m) = f(n-1, m) + f(n, m-1)$ for the third case even if $(n-1, m)$ and/or $(n, m-1)$ is invalid, because if so, their value will be set to 0.

Again, filling the table directly gives an $O(N^2)$ algorithm.

```
int f[N+1][N+1], C[N+1];
for(int i=0; i<=N; i++)
{
  for(int j=0; j<=N; j++)
  {
    if(i == 0 && j == 0)
      f[i][j] = 1;
    else if(i < j)
      f[i][j] = 0;
    else
    {
      f[i][j] = 0;
      if(i-1 >= 0)
        f[i][j] += f[i-1][j];
      if(j-1 >= 0)
        f[i][j] += f[i][j-1];
    }
  }
}
for(int i=1; i<=N; i++)
  C[i] = f[i][i];
```

It's interesting that if the constraint $n \geq m$ doesn't exist, the number of ways are just binomial numbers, $f(n,m) = \binom{n}{m}$.

In fact, there's a formula for Catalan numbers:

$$C_n = \frac{\binom{2n}{n}}{n+1}$$

.

(3) (10%)

Let $f(n,m)$ be the number of different ways to partition $n$ identical things into groups with size at most $m$, then $P_n = f(n,n)$.

Among partitions of $f(n,m)$, consider those with exactly $k$ groups of size $m$. For these partitions, removing such $k$ groups will results in partitions of $n-km$ with groups of size at most $m-1$. Because this is a one-to-one correspondence, so the number of partitions is just $f(n-km, m-1)$. It's valid for all $k \geq 0$ and $n - km \geq 0$, so we should sum over $k$.

The initial condition is $f(0,0) = 1$, where 0 has no group to partition. Also $f(n,0) = 0$ for $n > 0$, because it's impossible to partition $n$ using "no groups".

So we got the following recurrence relation:

$$f(n,m) = \begin{cases} 1 & , n = 0, m = 0 \\ 0 & , n > 0, m = 0 \\ \sum_{k=0}^{\lfloor \frac{n}{m} \rfloor} f(n-km, m-1) & , n \geq 0, m > 0 \end{cases}$$

Filling the table directly gives an $O(N^3)$ algorithm.

```
int f[N+1][N+1], P[N+1];
for(int i=0; i<=N; i++)
{
  for(int j=0; j<=N; j++)
  {
    if(j == 0)
      f[i][j] = (i == 0 ? 1 : 0);
    else
    {
      f[i][j] = 0;
      for(int x=i; x>=0; x-=j)
        f[i][j] += f[x][j-1];
    }
  }
}
for(int i=1; i<=N; i++)
  P[i] = f[i][i];
```

In fact, this algorithm runs in $O(N^2 \log N)$ time.

For $m \geq 1$, calculating $f(n, m)$ takes $O(n/m)$ time, so the total runtime is

$$\begin{aligned} & \sum_{n=0}^{N} \sum_{m=1}^{N} O(n/m) \\ \leq & \sum_{n=0}^{N} \sum_{m=1}^{N} O(N/m) \\ = & O\left(N^2 \sum_{m=1}^{N} \frac{1}{m}\right) \\ = & O(N^2 \log N) \end{aligned}$$

## Problem 2

(1) (5%) For a given HH-code string `hhcode`, `dp[i]` define as the number of all subsequence that end with character `hhcode[i]` and appears in the HH-code in the first time.

```
char hhcode[N];
int dp[N];
for(int i=0; i<N; i++){
    dp[i] = 0;
    for(int j=i-1; j>=0; j--){
        dp[i] += dp[j];
        if(hhcode[j] == hhcode[i]) break;
    }
    if(dp[i] == 0) dp[i] += 1;
}
int ans = 0;
for(int i=0; i<N; i++){
    ans += dp[i];
}
```

The code calculate the dp table with $O(N^2)$ time complexity.

(2) (5%) `dp[i]` has the same definition with last problem. We speed up the caculation with partial sum.

```

char hhcode[N];
int psum[2];
int dp[N];
bool flag[2] = {};

for(int i=0; i<N; i++){
    if( hhcode[i] == '0'){
        dp[i] = psum[0];
        if(!flag[0]) dp[i] += 1; //first time '0' appears
        psum[1] += dp[i];
        psum[0] = dp[i];
        flag[0] = true;
    }
    else if( hhcode[i] == '1'){
        dp[i] = psum[1];
        if(!flag[1]) dp[i] += 1; //first time '1' appears
        psum[0] += dp[i];
        psum[1] = dp[i];
        flag[1] = true;
    }
}
int ans = 0;
```

```
for(int i=0; i<N; i++){
    ans += dp[i];
}
```

The code calculate the dp table with $O(N)$ time complexity.

(3) (10%) We add one more dimension in dp. `dp[i][j]` define as the number of all subsequence that end with character `hhcode[i]` with length $j$ and appears in the HH-code in the first time.

```
char hhcode[N];
int psum[2][K+1];
int dp[N][K+1];

psum[0][0] = 1;
psum[1][0] = 1;

for(int i=0; i<N; i++){
    if( hhcode[i] == '0'){
        for(int j=1; j<=K; j++){
            dp[i][j] = psum[0][j-1];
        }
        for(int j=0; j<=K; j++){
            psum[1][j] += dp[i][j];
            psum[0][j] = dp[i][j];
        }
    }
    else if( hhcode[i] == '1'){
        for(int j=1; j<=K; j++){
            dp[i][j] = psum[1][j-1];
        }
        for(int j=0; j<=K; j++){
            psum[0][j] += dp[i][j];
            psum[1][j] = dp[i][j];
        }
    }
}

int ans = 0;
for(int i=0; i<N; i++){
    ans += dp[i][K];
}
```

## Problem 3

In this problem, we will learn some tricks to optimize dynamic programming algorithms.

(1) Fast Matrix Exponentiation (10%): Sometimes, the recursive formula in a dynamic programming problem is a linear combination of the answers in the previous state. That is, the recursive formula has the form

$$\begin{bmatrix} \texttt{dp}_1(n) \\ \texttt{dp}_2(n) \\ \vdots \\ \texttt{dp}_m(n) \end{bmatrix} = \begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,m} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m,1} & a_{m,2} & \cdots & a_{m,m} \end{bmatrix} \begin{bmatrix} \texttt{dp}_1(n-1) \\ \texttt{dp}_2(n-1) \\ \vdots \\ \texttt{dp}_m(n-1) \end{bmatrix}$$

For example, let $\texttt{dp}_1(n) = f(n)$ and $\texttt{dp}_2(n) = f(n-1)$, the formula below calculate the $n$-th fibonacci number with $f(0) = 0, f(1) = 1$.

$$\begin{bmatrix} f(n) \\ f(n-1) \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} f(n-1) \\ f(n-2) \end{bmatrix}$$

**When analysing the time complexity in this problem, assume that each of the arithmetic operations $+, -, \times, /$ takes $\mathcal{O}(1)$ time.**

1. (4%) Prove that for an $m \times m$ matrix $A$, both $A^n$ and $\sum_{i=0}^{n} A^i$ could be calculated in $\mathcal{O}\left(m^3 \lg n\right)$ time.

   *Hint: You should have learned how to calculate $A^n$ from homework 1.*
   *To calculate $\sum_{i=0}^{n} A^i$, consider the matrix $\begin{bmatrix} A & I \\ 0 & I \end{bmatrix}$.*

   To calculate $A^n$, if $n = 2k$, $A^n = A^{2k} = (A^k)^2$. If $n = 2k+1$, $A^n = (A^k)^2 \cdot A$. Let $T(n)$ be the time needed to calculate $A^n$, we have $T(n) = T(n/2) + \mathcal{O}\left(m^3\right)$, solving the recursive formula yields $T(n) = \mathcal{O}\left(m^3 \log_2 n\right)$.
   To calculate $\sum_{i=0}^{n} A^i$, consider $B^n$ where $B = \begin{bmatrix} A & I \\ 0 & I \end{bmatrix}$. We shall prove that $B^n = \begin{bmatrix} A^n & \sum_{i=0}^{n-1} A^i \\ 0 & I \end{bmatrix}$. The equality holds for $n = 1$. If $n = k$ holds, calculate

   $$B^{k+1} = \begin{bmatrix} A^k & \sum_{i=0}^{k-1} A^i \\ 0 & I \end{bmatrix} \begin{bmatrix} A & I \\ 0 & I \end{bmatrix} = \begin{bmatrix} A^{k+1} & \sum_{i=0}^{k} A^i \\ 0 & I \end{bmatrix}$$

   by using a well known result

   $$\begin{bmatrix} A_1 & A_2 \\ A_3 & A_4 \end{bmatrix} \begin{bmatrix} B_1 & B_2 \\ B_3 & B_4 \end{bmatrix} = \begin{bmatrix} A_1B_1 + A_2B_3 & A_1B_2 + A_2B_4 \\ A_3B_1 + A_4B_3 & A_3B_2 + A_4B_4 \end{bmatrix},$$

   if all the $A_i, B_i$ are square matrices which have the same dimensions. Thus by induction, we have shown the equality holds.
   To calculate $\sum_{i=0}^{n} A_i$, we just need to calculate $B^{n+1}$ and take the upper-right submatrix, which could be done in $\mathcal{O}\left(m^3 \log_2 n\right)$.

2. (6%) Solve the following problem.
   Recently Saki has discovered some new creatures that come from the new world. She analyses their DNA sequences and has the following findings.

(a) Unlike the livings on earth which have only 4 different kinds of nucleotide, they have $K$ different kinds of nucleotide.

(b) Some nucleotide pairs cannot appear consecutively due to some chemical effects. In other words, there are $Q$ pairs $(\beta_{i,1}, \beta_{i,2})$ such that the nucleotide of the $\beta_{i,1}$-th and $\beta_{i,2}$-th kind could not appeared consecutively (with either $\beta_{i,1}$-th or $\beta_{i,2} - th$ kind appearing first).

(c) The nucleotide at a position $a_i$ in the sequences will always be a certain kind of nucleotide $b_i$. There are $P$ of such positions.

(d) The length of the DNA sequence is between $L$ and $R$ where $\max(a_i) \leq L \leq R$.

Saki wonders how many possible DNA sequences exist, but since the answer may be large, she only want to know the answer mod $M$.
Give an algorithm to solve the problem in $\mathcal{O}\left(K^3 P \lg R\right)$ time.

Let $\texttt{dp}_i(n)$ be the number of DNA sequences which end with nucleotide of the $i$-th kind and it does not violate the first three rules. Specially, let $\texttt{dp}_0(n)$ be the number of sequences that do not have a last nucleotide, which is always zero except $\texttt{dp}_0(0) = 1$. Let

$$P = \begin{bmatrix} 0 & 0 & \cdots & 0 \\ 1 & a_{1,1} & \cdots & a_{1,K} \\ 1 & a_{2,1} & \cdots & a_{2,K} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & a_{K,1} & \cdots & a_{K,K} \end{bmatrix}, \quad \texttt{dp}(n) = \begin{bmatrix} \texttt{dp}_0(n) \\ \texttt{dp}_1(n) \\ \texttt{dp}_2(n) \\ \vdots \\ \texttt{dp}_K(n) \end{bmatrix}$$

Where $a_{i,j}$ is 0 if nucleotide of the $i$-th and $j$-th kinds cant appeared consecutively, otherwise 1. Then if $n \neq a_i \ \forall i$, we could write $\texttt{dp}(n) = P \cdot \texttt{dp}(n-1)$, Since we could get a sequence of length $n$ that ends with $i$-th nucleotide by adding a nucleotide from a sequence of length $n-1$ that ends with $j$-th nucleotide, provided that these two nucleotide could appeared consecutively.
If $n = a_i$ for some $i$, which means that the position have to be a specific nucleotide, say the $b_i$-th nucleotide. Then, we additionally multiply $Q_{b_i} = [q_{j,k}]$,
where $q_{j,k} = 1$ if $j = k = b_i$, otherwise 0, to zero out $\texttt{dp}_x(n) \ \forall x \neq b_i$.
That is, $\texttt{dp}(n) = Q_{b_i} \cdot P \cdot \texttt{dp}(n-1)$. So, the number of valid sequences that satisfied all four rules equals

$$\sum_{i=L}^{R} \sum_{j=1}^{n} \texttt{dp}_j(i) = \begin{bmatrix} 0 \\ 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix}^{\intercal} \left( \sum_{i=L-a_P}^{i=R-a_P} P^i \right) Q_{b_K} P^{a_K - a_{K-1}} \cdots Q_{b_2} P^{a_2 - a_1} Q_{b_1} P^{a_1} \begin{bmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

Which can be calculate by using $P + 1$ fast matrix Exponentiation. Using the result from the previous question, the algorithm runs in $\mathcal{O}\left(K^3 P \log_2(R)\right)$ time.

(2) Convex Hull Optimization (20%): For dynamic programming problems of the form

$$\texttt{dp}(i) = \min_{j<i} a_j x_i + b_j, \quad \text{with } a_j \text{ decreasing and } x_i \text{ increasing.} \tag{1}$$

It's easy to calculate all the `dp` values in $\mathcal{O}\left(n^2\right)$ time, but we could do it better.

1. (3%) Let $f_j(x) = a_j x + b_j$, so that $\texttt{dp}(i) = \min_{j<i} f_j(x_i)$. Define $\bar{x}(j,k)$ for $j < k$ to be the minimal $x$ such that $f_k(x) \le f_j(x)$.
   That is, if $x_i \ge \bar{x}(j,k)$, then for $\texttt{dp}(i)$, $f_k(x_i) \le f_j(x_i)$, so it is better to "transfer" from $k$ rather than $j$. Notice that since $a_j$ is decreasing, the function is well defined.
   Please give a formula for $\bar{x}(j,k)$.

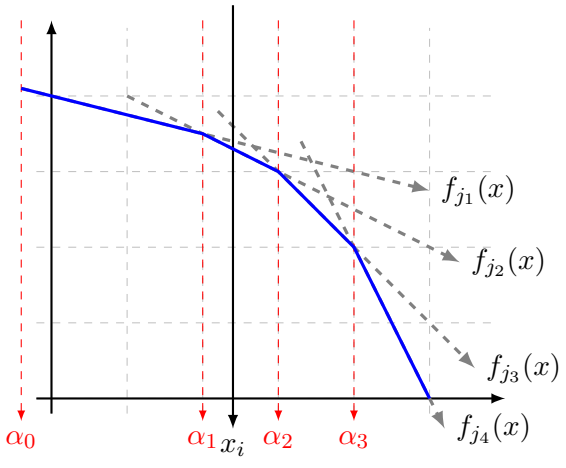
   We simply calculate

$$\begin{aligned} & f_j(x) > f_k(x) \\ \Leftrightarrow \quad & a_j x + b_j > a_k x + b_k \\ \Leftrightarrow \quad & x > \frac{b_k - b_j}{a_j - a_k} \end{aligned}$$

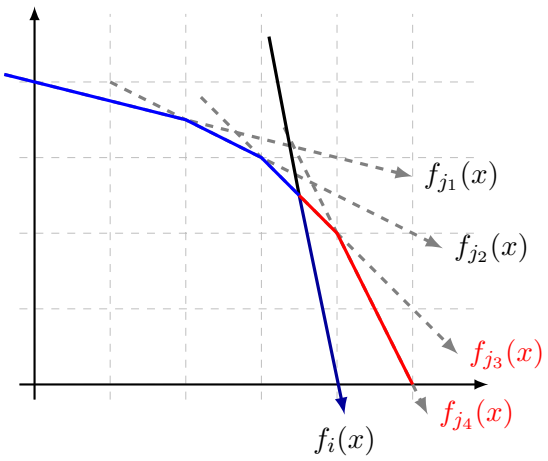
   so $\bar{x}(j,k) = \frac{b_k - b_j}{a_j - a_k}$

2. (10%) Describe an algorithm to calculate $\texttt{dp}(i)$ for all $1 \le i \le n$ in $\mathcal{O}(n)$ time.

   *Hint: See $f_j(x)$ as a line. Use a deque to maintain a sequence $(j_1, j_2, \cdots, j_m)$, such that $j_1 < j_2 < \cdots < j_m$, and there exists $\alpha_0 < \alpha_1 < \cdots < \alpha_m = \infty$ such that $\alpha_0 \le x_k$ and $f_{j_\gamma}(x)$ is the minimum of $f_1(x), f_2(x), \cdots, f_{k-1}(x)$ for $\alpha_{\gamma-1} \le x \le \alpha_\gamma$. If this could be achieved, how to calculate $\texttt{dp}(i)$?*
   *Finally, how to add a new line (namely, $f_i(x)$) into the deque?*




Calculating $\texttt{dp}(i)$




Adding $f_i(x)$

   Let's calculate $\texttt{dp}(i)$ with increasing order.
   Assuming that we are calculating $\texttt{dp}(k)$, and we've maintain a dequeue which has elements $[j_1, j_2, \cdots, j_m]$, such that $j_1 < j_2 < \cdots < j_m$, and there exists $\alpha_0 < \alpha_1 < \cdots < \alpha_m = \infty$

such that $\alpha_0 \le x_k$ and $f_{j_\gamma}(x)$ is the minimum of $f_1(x), f_2(x), \cdots, f_{k-1}(x)$ for $\alpha_{\gamma-1} \le x \le \alpha_\gamma$.

Now, to obtain $\texttt{dp}(k)$, we check that whether $x_k \le \alpha_1$ or not. If yes, by the assumption $\texttt{dp}(k) = f_{j_1}(x_k)$ and we're done. Or else if $x_k > \alpha_1$, we shall preform a `pop_front` to the deque (so the deque becomes $[j'_1, j'_2, \cdots, j'_{m'=m-1}] = [j_2, j_3, \cdots, j_m]$) and recheck the condition until finally $x_k \le \alpha_1$.

Finally, we need to add $i$ into the dequeue (since $i$ could be used for $i' > i$) and ensure that the assumption still holds when we calculates $\texttt{dp}(k+1)$, so the procedure could continue. We check that whether $\alpha_{m-1} > \bar{x}(j_m, i)$ or not. (notice that $\alpha_{m-1} = \bar{x}(j_{m-1}, j_m)$.) If the condition doesn't hold, it means that when $f_{j_m}$ is finally smaller than $f_{j_{m-1}}$, $f_i$ is already smaller than $f_{j_m}$, so $j_m$ is useless and we shall preform a `pop_back`. We continue to check the last element in the deque until the condition finally holds. Then we preform a `push_back` to push $i$ into the deque and we repeat the procedure to calculate $\texttt{dp}(i+1)$.
It is easy to see that the algorithm runs in $\mathcal{O}(n)$ since every element would only be pushed and pop out once.

3. (7%) Solve the following problem.
Manaka is a girl who lives in the ocean, and she likes to feed fish. There are $a_i$ fish at position $i$, for $0 \le i \le n$, where 0 represents the westernmost. Manaka wishes to feed every fish, so she decides to place some food at some positions. Placing food at position $i$ would consume Manaka $c_i$ unit of physical power, and the food is enough for any number of fish to eat. Because of the ocean current, the fish could only swim toward the west. Each of the fish will swim to the nearest position that has food in their west. It costs 1 unit of physical power for a fish to swim an unit distance. Since swimming is a strength consuming work, she wants the sum of the physical power consumed by her and all of the fish to be minimal.
Give an algorithm to solve the problem in $\mathcal{O}(n)$ time.

*Hint: You could use the result from 2 directly, but you need to show your recursive formula has the same form as equation (1), and all the parameters could be calculated in $\mathcal{O}(1)$ time.*

Let $c(j, i)$ be the cost to feed all the fish between $j$ and $i$ at position $j$. We have

$$\begin{aligned} c(j,i) &= 0 \cdot a_j + 1 \cdot a_{j+1} + \cdots + (i-j) \cdot a_i + c_j \\ &= \sum_{k=j}^{i} (k-j) a_k + c_j \\ &= \sum_{k=j}^{i} (k a_k) - j \cdot \sum_{k=j}^{i} a_k + c_j \\ &= p_i - p_{j-1} - j(s_i - s_{j-1}) + c_j \end{aligned}$$

If we define

$$s_i = \sum_{k=0}^{i} s_k \quad \text{and} \quad p_i = \sum_{k=0}^{i} k s_k$$

Now we could write down the recursive equation

$$\begin{aligned}
\texttt{dp}(i) &= \min_{j<i} \left(\texttt{dp}(j) + c(j+1, i)\right) \\
&= \min_{j<i} \left(\texttt{dp}(j) + p_i - p_j - (j+1)(s_i - s_j) + c_{j+1}\right) \\
&= p_i - s_i + \min_{j<i} \left(-j \cdot (s_i) + (\texttt{dp}(j) - p_j + j s_j - s_j + c_{j+1})\right) \\
&= p_i - s_i + \min_{j<i} \left(a_j \cdot x_i + b_j\right)
\end{aligned}$$

If we let $a_j = -j$ and $x_i = s_i$.
Notice that $a_j \searrow$ and $x_i \nearrow$. So the recursive equation has the same form as equation (1). If we pre-calculate all the $s_i, p_i$ once at the beginning, which takes $\mathcal{O}(N)$, the algorithm then runs in $\mathcal{O}(N)$ time by the previous result.

Problem (3) (4) is here just to prevent that you think the problem above is quite easy and feel bored. **You don't need to submit these problems. If you do, your solution would be judge, but you won't get any points from these problems.**

(3) Quadrangle Inequality Optimization:
Anna and Elsa know that the kids love snowmen, so they decide to build $K$ snowmen. There are $n$ kids who live in position $a_1 < a_2 < \cdots < a_n$, and each of them will go to play with the nearest snowman. Anna and Elsa wonders where to build the $K$ snowmen so that the sum of distances each kids have to walk would be the minimum.

1. Let $\texttt{dp}(k, i)$ represent the minimum total distance to build $k$ snowmen for the first $i$ kids. The recursive formula of this problem has the form

$$\texttt{dp}(k, i) = \min_{0 \le j < i} \texttt{dp}(k-1, j) + w(j, i), \quad 0 \le k \le i \le n$$

where $w(j, i)$ means the minimum distances of the kids from $j+1$ to $i$ if they play with the same snowman. Briefly explain that one of the possible place to build snowman for minimum cost is the position $a_t$ where $t = \lfloor (i+j)/2 \rfloor$ and that $w(j, i) = w(j, i-1) + a_i - a_t$.

Let $f(x)$ be the cost to build a snowmen at $x$ for $a_{j+1}$ to $a_i$.

$$f(x) = \sum_{k=j+1}^{i} |a_k - x| = \sum_{a_k < x} (x - a_k) + \sum_{a_k > x} (a_k - x)$$

If we limit $j+1 \le k \le i$. Differentiate $f(x)$ gives

$$f'(x) = \#(a_k < x) - \#(a_k > x).$$

Since $f(x)$ will not reach a minimum if $f(x) \neq 0$, the snowmen should be build at which the number of kids that are on its left should be equal to the number that are on its right. If $i - j$ is odd, that is, there are an odd number of $a_k$, the snowmen should be build at the median, namely $a_t$. If $i - j$ is even, than the snowmen should be build between $a_t, a_t + 1$. But since $f'(x)$ is zero there, it's fine to build at $a_t$.
So now we could write

$$w(j, i) = \sum_{k=\lceil (i+j+1)/2 \rceil}^{i} a_k - \sum_{k=j+1}^{\lfloor (i+j+1)/2 \rfloor} a_k$$

and

$$w(j, i-1) = \sum_{k=\lceil (i+j)/2 \rceil}^{i} a_k - \sum_{k=j+1}^{\lfloor (i+j)/2 \rfloor} a_k.$$

if $i + j$ is odd, then

$$\lceil \frac{i+j+1}{2} \rceil = \lfloor \frac{i+j+1}{2} \rfloor = \lceil \frac{i+j}{2} \rceil = \lfloor \frac{i+j}{2} \rfloor + 1$$

so $w(j, i) - w(j, i-1) = a_i - a_t$. if $i + j$ is even, then

$$\lceil \frac{i+j+1}{2} \rceil - 1 = \lfloor \frac{i+j+1}{2} \rfloor = \lceil \frac{i+j}{2} \rceil = \lfloor \frac{i+j}{2} \rfloor$$

so again $w(j, i) - w(j, i-1) = a_i - a_t$.

2. We could calculate all the `dp` value in $\mathcal{O}\left(n^2K\right)$ time easily. Define

$$A(k,i) = \arg\min_{0\le j<i} \texttt{dp}(k-1,j) + w(j,i)$$

If we know that $A(k,i-1) \le A(k,i) \le A(k+1,i)$, Prove that all the `dp` value could be calculate in $\mathcal{O}\left(N^2\right)$ time, Assuming that there exists a good dynamic programming order such that Both $A(k,i-1)$ and $A(k+1,i)$ is calculate before $A(k,i)$.

*Hint: Calculate*

$$\sum_{k=1}^{n}\sum_{i=1}^{n} A(k+1,i) - A(k,i-1) = \sum_{d=i-k=1}^{n}\sum_{i=d+1}^{n} A(i-d+1,i) - A(i-d,i-1)$$

*. Expand* $\sum_{i=d+1}^{n} A(i-d+1,i) - A(i-d,i-1)$.

Calculate

$$\sum_{k=1}^{n}\sum_{i=1}^{n} A(k+1,i) - A(k,i-1) = \sum_{d=i-k=1}^{n}\sum_{i=d+1}^{n} A(i-d+1,i) - A(i-d,i-1)$$

. Notice that

$$\begin{aligned}
&\sum_{i=d+1}^{n} A(i-d+1,i) - A(i-d,i-1) \\
&= A(n-d+1,n) - A(n-d,n-1) + A(n-d,n-1) - A(n-d-1,n-2) + \cdots \\
&\quad + A(2,d+1) - A(1,d) \\
&= A(n-d+1,n) - A(1,d) \\
&< n
\end{aligned}$$

Since $A(k,i) < n\ \forall k,i$, so

$$\sum_{k=1}^{n}\sum_{i=1}^{n} A(k+1,i) - A(k,i-1) < n^2 = \mathcal{O}\left(N^2\right)$$

. Hence we could improved simply by rewriting the recursive formula to

$$\texttt{dp}(k,i) = \min_{A(k,i-1)\le j\le A(k+1,i)} \texttt{dp}(k-1,j) + w(j,i)$$

.

*3. Prove that the inequality $A(k,i-1) \le A(k,i) \le A(k+1,i)$ hold for this problem.

*Hint: First proof that $w$ satisfied the* **quadrangle inequality**

$$w(j,i) + w(j+1,i+1) \le w(j+1,i) + w(j,i+1).$$

*Then proof that* `dp` *also satisfied the quadrangle inequality by induction in* $k$

$$\texttt{dp}(k,i) + \texttt{dp}(k+1,i+1) \le \texttt{dp}(k+1,i) + \texttt{dp}(k,i+1).$$

*$A(k, i-1) \le A(k, i)$ could be proved directly using the quadrangle inequality of $w$. To proof the later, use the one with* `dp`.

First we proof that $w$ satisfied the quadrangle inequality

$$w(j,i) + w(j+1,i+1) \le w(j+1,i) + w(j,i+1).$$

We have

$$\begin{aligned}
& w(j,i) + w(j+1,i+1) \le w(j+1,i) + w(j,i+1) \\
\Leftrightarrow\ & w(j,i) + w(j+1,i) + a_{i+1} - a_t \le w(j+1,i) + w(j,i) + a_{i+1} - a_{t'} \\
\Leftrightarrow\ & a_t \ge a_{t'} \\
\Leftrightarrow\ & t \ge t'
\end{aligned}$$

Where $t = \lfloor (i+1+j+1)/2 \rfloor, t' = \lfloor (i+1+j)/2 \rfloor$. It obvious that $t \ge t$, hence the inequality holds.
Notice that we could deduce that

$$w(j,i) + w(j',i') \le w(j',i) + w(j,i'), \quad \forall i < i', j < j'$$

Since adding $w(j+1,i) + w(j+2,i+1) \le w(j+2,i) + w(j+1,i+1)$ to the original inequality gives

$$w(j,i) + w(j+2,i+1) \le w(j+2,i) + w(j,i+1)$$

proceed similarly gives

$$w(j,i) + w(j+\alpha,i+\beta) \le w(j+\alpha,i) + w(j,i+\beta)$$

which gives the extend inequality.
Then we proof that `dp` also satisfied the quadrangle inequality

$$\texttt{dp}(k,i) + \texttt{dp}(k+1,i+1) \le \texttt{dp}(k+1,i) + \texttt{dp}(k,i+1).$$

It is easy to check that the inequality holds when $k = 1$. Let $j_{0,0} = A(k,i), j_{1,1} = A(k+1,i+1), j_{1,0} = A(k+1,i), j_{0,1} = A(0,1)$, There are two cases:

Case #1 : $j_{0,1} \le j_{1,0}$

$$\begin{aligned}
& \texttt{dp}(k,i) + \texttt{dp}(k+1,i+1) \\
=\ & \texttt{dp}(k-1,j_{0,0}) + w(j_{0,0},i) + \texttt{dp}(k,j_{1,1}) + w(j_{1,1},i+1) \\
\le\ & \texttt{dp}(k-1,j_{0,1}) + w(j_{0,1},i) + \texttt{dp}(k,j_{1,0}) + w(j_{1,0},i+1) \\
\le\ & \texttt{dp}(k,j_{1,0}) + w(j_{1,0},i) + \texttt{dp}(k-1,j_{0,1}) + w(j_{0,1},i+1) \\
=\ & \texttt{dp}(k,i+1) + \texttt{dp}(k,i+1)
\end{aligned}$$

Where the first inequality holds because of the minimality of the `dp`, the second holds due to the quadrangle inequality.

Case #2 : $j_{0,1} > j_{1,0}$

$$\begin{aligned}
& \texttt{dp}(k,i) + \texttt{dp}(k+1,i+1) \\
=\ & \texttt{dp}(k-1,j_{0,0}) + w(j_{0,0},i) + \texttt{dp}(k,j_{1,1}) + w(j_{1,1},i+1) \\
\le\ & \texttt{dp}(k-1,j_{1,0}) + w(j_{1,0},i) + \texttt{dp}(k,j_{0,1}) + w(j_{0,1},i+1) \\
\le\ & \texttt{dp}(k,j_{1,0}) + w(j_{1,0},i) + \texttt{dp}(k-1,j_{0,1}) + w(j_{0,1},i+1) \\
=\ & \texttt{dp}(k,i+1) + \texttt{dp}(k,i+1)
\end{aligned}$$

Where the first inequality holds because of the minimality of the `dp`, the second holds by induction.

Then we proof that $A(k, i-1) \leq A(k, i)$. If not, let $j = A(k, i-1), j' = A(k, i)$. then $j > j'$. We have $\texttt{dp}(k-1, j) + w(j, i-1) < \texttt{dp}(k-1, j') + w(j', i-1)$ and $\texttt{dp}(k-1, j') + w(j', i) < \texttt{dp}(k-1, j) + w(j, i)$. Adding the two inequality gives

$$w(j, i-1) + w(j', i) < w(j', i-1) + w(j, i)$$

Which contradict with the quadrangle inequality.

Finally we proof that $A(k, i) \leq A(k+1, i)$. Suppose not, let $j = A(k, i), j' = A(k+1, i)$. then $j > j'$. We have $\texttt{dp}(k-1, j) + w(j, i) < \texttt{dp}(k-1, j') + w(j', i)$ and $\texttt{dp}(k, j') + w(j', i) < \texttt{dp}(k, j) + w(j, i)$ by the minimality of `dp`. Adding the two inequality gives

$$\texttt{dp}(k-1, j) + \texttt{dp}(k, j') < \texttt{dp}(k-1, j') + \texttt{dp}(k, i)$$

Which contradict with the quadrangle inequality.

(4) Divide and Conquer:
After a serious disaster, Kanba and Ringo decide to rebuild a new metro in their home town. The new metro system has $n$ stations, with the route going through station $1, 2, 3, \cdots, n-1, n$ and then back to station 1 (That is, the route is cyclic). Station $i$ is located at $(x_i, y_i)$. Due to some analysis in geology, for each station $i$, there are $m_i$ posible depth $0 > z_{i,1} > z_{i,2} > \cdots > z_{i,m_i}$ where the station could be located (i.e ., there are total $\sum m_i = m$ candidates). The route would be a straight segment connect consecutive stations, so if they choose to build the station at $z_{i,a_i}$ for each station, the total length would be $d(1,2) + d(2,3) + \cdots + d(n-1,n) + d(n,1)$, where

$$d(i,j) = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2 + (z_{i,a_i} - z_{j,a_j})^2}$$

Since the cost is proportional to the total length, they want to choose the best location $a_i$ for each station to minimize the total length.

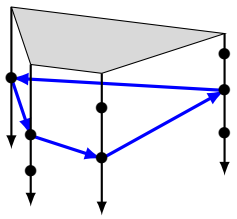


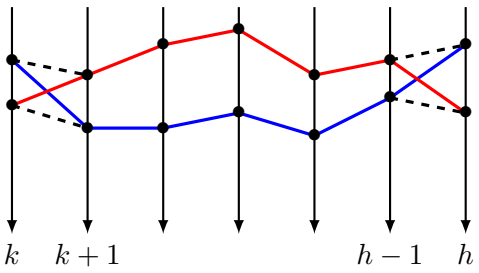
Figure 1: A possible route

We define $p_{i,j}$ to be the point $(x_i, y_i, z_{i,j})$, ans we use a tuple $A = (a_1, a_2, \cdots, a_n)$ to represent a route with points $p_{1,a_1}, p_{2,a_2}, \cdots, p_{n,a_n}$.

1. Let $A = (a_1, a_2, \cdots, a_n)$ and $B = (b_1, b_2, \cdots, b_n)$ be two routes. If there exists $1 \le k < h \le n$ such that $a_k \ge b_k, a_{k+1} < b_{k+1}$ and $a_{h-1} < b_{h-1}, a_h \ge b_h$, (i.e., the two routes cross each other). Briefly explain that if we changes these routes to

$$A' = (a_1, a_2, \cdots, a_k, b_{k+1}, \cdots, b_{h-1}, a_h, \cdots, a_n)$$
$$B' = (b_1, b_2, \cdots, b_k, a_{k+1}, \cdots, a_{h-1}, b_h, \cdots, b_n).$$

Then one of them will be better than before, or the length of both of them will remain the same.




By this lemma, we could safely assume that if $A, B$ are two routes that pass through two different point $p, q$ with minimal length, then these routes will not cross each other.

*Hint: Calculate the sum of these two routes and compare to the origin. Use the same idea of the hint from Problem 3.(1) in homework 1.*

We compare the sum of length of two routes. Define $|A|$ to be the length of route $A$, then

we have

$$\begin{aligned}&|A| + |B| - |A'| - |B'| \\ =\ & d(a_k, a_{k+1}) + d(b_k, b_{k+1}) + d(a_h, a_{h+1}) + d(b_h, b_{h+1}) \\ & - d(a_k, b_{k+1}) - d(b_k, a_{k+1}) - d(a_h, b_{h+1}) - d(b_h, a_{h+1})\end{aligned}$$

Since $a_k \geq b_k, a_{k+1} < b_{k+1}$, segment $p_{k,a_k}, p_{k+1,a_{k+1}}$ and $p_{k,b_k}, p_{k+1,b_{k+1}}$ cross each other. By Problem 3.(1) in homework 1, we know that $d(a_k, a_{k+1}) + d(b_k, b_{k+1}) \geq d(a_k, b_{k+1}) + d(b_k, a_{k+1})$. Apply a similar result to $a_h, b_h, a_{h+1}, b_{h+1}$ we also have $d(a_h, a_{h-1}) + d(b_h, b_{h-1}) \geq d(a_h, b_{h-1}) + d(b_h, a_{h-1})$. So $|A| + |B| \geq |A'| + |B'|$, hence at least one new route is better, or both the lenght of the two routes remain the same.

2. If we know that the best route pass through station 1 at $p_{1,\hat{a}_1}$, give an algorithm to find the minimal route in $\mathcal{O}(m \log m)$ time.
   Of course, this algorithm could also be apply to the case such that the best route pass through a point $p_{k,\hat{a}_k}$ with $k \neq 1$.

*Hint: Let* $\texttt{dp}(i, j)$ *be the minimum length start from* $z_{1,\hat{a}_1}$ *and ends at* $z_{i,j}$*. For each station* $i$*, let* $k = \lfloor m_i/2 \rfloor$*, calculate* $\texttt{dp}(i, k)$ *by going through* $\texttt{dp}(i-1, x)$*. If* $\hat{x}$ *is the one which let* $\texttt{dp}(i, k)$ *be the smallest, Then by 1., when calculating* $\texttt{dp}(i, h)$ *for* $h < k$*, you only need to consider* $\texttt{dp}(i-1, y)$ *for* $y \leq \hat{x}$*.*

Let $\texttt{dp}(i, j)$ be the minimum length start from $z_{1,\hat{a}_1}$ and ends at $z_{i,j}$. The recursive formula is

$$\texttt{dp}(i, j) = \min_{1 \leq x \leq m_{i-1}} \texttt{dp}(i, x) + d(p_{i-1,x}, p_{i,j})$$

Now, for each station $i$, let $k = \lfloor m_i/2 \rfloor$, calculate $\texttt{dp}(i, k)$ by enumerate through $x$ from 1 to $m_{i-1}$. If $\hat{x}$ is the one which let $\texttt{dp}(i, k)$ be the smallest, Then by 1., when calculating $\texttt{dp}(i, h)$ for $h < k$, we only need to consider $\texttt{dp}(i-1, y)$ for $y \leq x$. similarly, when we are calculating $\texttt{dp}(i, h)$ for $h > k$, we only need to consider $\texttt{dp}(i-1, y)$ for $y \geq x$. Define $T(q, m)$ be the time needed to calculate $q$ $\texttt{dp}$ values, which there are $m$ possible candidates we need to enumerate through. Then $T(q, m) = T(q/2, m_1) + T(q/2, m_2) + m$, with some $m_1 + m_2 = m + 1$. We shall proof that $T(q, m) \leq (m-1)\log_2(q) + 2q - 1 + m - 1$. When $q = 1$, $T(1, m) = m$ so the inequality is valid, we check that

$$\begin{aligned}T(q, m) =& T(q/2, m_1) + T(q/2, m_2) + m \\ \leq& (m_1 + m_2 - 2)\log_2(q/2) + 2 \cdot 2(q/2) - 2 + m_1 + m_2 - 1 + m \\ \leq& (m-1)\log_2(q) - m + 1 + 2q - 2 + m + 1 - 2 + m \\ =& (m-1)\log_2(q) + 2q - 1 + m - 1.\end{aligned}$$

So by induction, we've proof that $T(n, m) = \mathcal{O}(m \log_2(n))$, so the complexity of this algorithm is $\sum_i T(m_i, m_{i-1}) = \mathcal{O}(m \log_2 m)$.

*3. By the previous result, if we enumerate through $z_{1,1}, z_{1,2}, \cdots, z_{1,m_1}$, we obtain an $\mathcal{O}\left(m^2 \log_2 m\right)$ solution. But again, by using a divide and conquer method, proof that we could get an $\mathcal{O}\left(m (\log_2 m)^2\right)$ solution.

*Hint: Choose a station* $i$ *and assume the best route would pass through point* $p_{i,k}$ *where* $k = \lfloor m_i/2 \rfloor$*. Calculate the best route* $A$ *in* $\mathcal{O}(m \log_2(m))$ *using the previous problem.*

*Now, since the route wouldn't cross, do a divide and conquer by arranging the points into two groups $P^+, P^-$, such that $P^+$ contains the points that is in $A$ or above $A$, $P^-$ contains the points that is in $A$ or below $A$. solve for the time complexity you'll get $T = \mathcal{O}\left(m\log_2(m)\log_2(m_i) + nm_i\log_2(m_i)\right)$. Take good care for $nm_i\log_2(m_i)$. Would choosing $i$ with the minimal $m_i$ help?*

Choose a station $i$, and we shall enumerate through all the posible depth $z_{i,1}, \cdots, z_{i,m_i}$ to be the depth in the best route. We proceed as follow.
Assume that the best route would pass through point $p_{i,k}$ where $k = \lfloor m_i/2 \rfloor$. Calculate the best route $A$ in $m\log_2(m)$ using the previous problem. Now, since the route wouldn't cross, do a divide and conquer by arranging all the points $P$ into two groups $P^+, P^-$, such that $P^+$ contains the points that is in $A$ or above $A$, $P^-$ contains the points that is in $A$ or below $A$. That is, let $Z = \{z_{i,1}, \cdots, z_{i,m_i}\}$, $Z^+ = \{z \in Z : z > z_k\}, Z^- = \{z \in Z : z < z_k\}$. We divide the original problem $(Z, P)$ to $(Z^+, P^+), (Z^-, P^-)$.
Let us now calculate the time complexity. Let $T(q, m)$ be the time needed to solve the problem with size $|Z| = q, |P| = m$ We have

$$T(q, m) = T(q/2, m_1) + T(q/2, m_2) + m\log_2(m)$$

for some $m_1 + m_2 = m + n$, since $P^+ \cup P^- = A$ and $|A| = n$. Solve for $T(m_i, m)$ we obtain

$$T = \mathcal{O}\left(m\log_2(m)\log_2(m_i) + nm_i\log_2(m_i)\right).$$

We shall take good care for $nm_i\log_2(m_i)$. If we choose $i$ with the minimal $m_i$, we have $m_i \leq m/n$ by the pigeonhole principle. Hence the time complexity becomes

$$T = \mathcal{O}\left(m\log_2(m)\log_2(m_i)\right) = \mathcal{O}\left(m\left(\log_2(m)\right)^2\right).$$

## Problem 4 - Lucky Number (Programming)

See the sample solution attached on the website.