#HW2

姓名：宋子維 系級：資工二 學號：B03902042

Problem 1

1. First, we know the recurrence relation

   $C(n,\ m) = C(n-1,\ m) + C(n-1,\ m-1),\ N \geq n > m > 0$. We can use a 2D array to store $C(n,\ m)$ in $C[n][m]$. Fill the array from the upper left corner and when $j = 0\ or\ j = i,\ C[i][j] = 1$. Then we can write down pseudo code.

   $For\ i \leftarrow 0\ to\ N$
   $\quad For\ j \leftarrow 0\ to\ N$
   $\quad\quad if\ j = 0\ or\ j = i$
   $\quad\quad\quad C[i][j] \leftarrow 1$
   $\quad\quad else$
   $\quad\quad\quad C[i][j] \leftarrow C[i-1][j] + C[i-1][j-1]$

   Then the complexity can be analyzed.

   $\sum_{i=0}^{N} \sum_{j=0}^{N} 1 = O(N^2)$

2. Let $f(n,\ m)$ be the number of ways to go from (0, 0) to (n, m) and $\forall i,\ f(i,\ 0) = 1\ and\ f(0,\ i) = 1$.

   Then we can find the recurrence relation.

   $f(n,\ m) = f(n,\ m-1),\ if\ n = m$
   $f(n,\ m) = f(n-1,\ m) + f(n,\ m-1),\ if\ n \neq m$
   There is the pseudo code below.

   $//\ f[n][m]\ is\ the\ number\ of\ ways\ to\ go\ from\ (0,\ 0)\ to\ (n,\ m)$
   $//\ f[n][n]\ is\ C_n$

   $f[i][0] \leftarrow 1,\ f[0][i] \leftarrow 1,\ for\ any\ i \leq N\ else\ f[i][j] \leftarrow 0\ //\ initial$
   $For\ i \leftarrow 1\ to\ N$
   $\quad For\ j \leftarrow 1\ to\ N$
   $\quad\quad if\ i = j$
   $\quad\quad\quad f[i][j] \leftarrow f[i][j-1]$
   $\quad\quad else$
   $\quad\quad\quad f[i][j] \leftarrow f[i][j-1] + f[i-1][j]$

   Then the complexity can be analyzed.

   $\sum_{i=1}^{N} \sum_{j=1}^{N} 1 = O(N^2)$

3. reference: http://openhome.cc/Gossip/AlgorithmGossip/SeparateNumber.htm

   Let $f(n,\ m)$ be the number of ways to partition n into groups of size at most m and then

   $P_n = f(n\ ,n)$ clearly. $\forall i \leq N,\ f[i][0] = 1\ and\ f[i][1] = 1$

   $if\ j > i,\ f[i][j] = f[i][i]$

   Take a look at $P_5 = 5 \rightarrow$ 5 = 4+1= 3+2 = 3+1+1 = 2+2+1 = 2+1+1+1 = 1+1+1+1+1

   ⇒ $P_5 = f(5,\ 5) = f(4,\ 1) + f(3,\ 2) + f(2,\ 3) + f(1,\ 4) + f(0,\ 5)$

   $= f(4,\ 1) + f(3,\ 2) + f(2,\ 2) + f(1,\ 1) + f(0,\ 0)$

$\Rightarrow\ P_n = \sum_{i=1}^{n} f(n-i,\ min(i-k,\ i))$

Then we can write down pseudo code.

$//P_n\ is\ stored\ in\ f[n][n]$

$f[i][0] \leftarrow 1,\ f[i][1] \leftarrow 1,\ for\ all\ i \le N,\ else\ f[i][j] \leftarrow 0\ //\ initial$

$For\ i \leftarrow 2\ to\ N$
$\quad For\ j \leftarrow 2\ to\ N$
$\quad\quad For\ k \leftarrow 1\ to\ min(i,\ j)$
$\quad\quad\quad f[i][j] \leftarrow f[i][j] + f[i-k][min(i-k,\ k)]$

Then complexity can be analyzed.

$\sum_{i=2}^{N}\sum_{j=2}^{N}\sum_{k=1}^{min(i,j)} 1 \le \sum_{i=1}^{N}\sum_{j=1}^{N}\sum_{k=1}^{N} 1 = O(N^3)$

Problem 2

1. dicuss with B03902048

Maintain two 2D-arrays with size $N \times N$, one called $endzero$, and the other called $endone$. Each $(i,\ j)$ entry in $endzero$ means the number of subsequences of the first j characters of HH-code with length i and ending with 0. Similarly, $(i,\ j)$ entry in $endone$ is the same as $endzero$, but ending with 1. Thus when $i > j$, entry $(i,\ j)$ equals to 0 because there is no subsequence with length i when the the size of input sequence is less than i.

Suppose the HH-code is s (begin is index 1 and end is index N). First, filling up the 1st row from left to right of two arrays. Since 1st row is subsequences with length 1, then it is easy to find that $endzero[1][j]$ is 1 if there exists some $1 \le t \le j$ such that $s[t] = 0$. $endone[0][j]$ follows the same rules but changes $s[t] = 0$ to $s[t] = 1$. And this thing can be done in $O(N)$. Then fill 2nd row from 2nd colume to Nth colume of two arrays, and so on until finishing $(n,\ n)$ entry. But how to fill $(i,\ j)$ entry with $i \ne 1,\ i < j$ of $endzero$? If $s[j] = 0$, then $endzero[i][j] \leftarrow endzero[i-1][j-1] + endone[i-1][j-1]$; otherwise, $endzero[i][j] \leftarrow endzero[i][j-1]$. It is because if $s[i] = 0$, then appending 0 at subsequences of first $i-1$ characters with length $j-1$, then new sequences will end with 0 no matter what the end of origin sequences is. But if $s[i] = 1$, it is useless to append it to the any subsequences. Hence, the number of $endzero[i][j]$ is equal to $endzero[i][j-1]$. To fill $endone[i][j]$, follow the same rules above, but just swap 0 and 1.

Then the answer is the sum of the number of subsequences of first N characters with length 1 to N ending with 0 and 1, that is,

$endzero[1][N] + endzero[2][N] ... + endzero[N][N] +$

$endone[1][N] + endone[2][N] ... + endone[N][N]$

which can be calculates in $O(N)$.

To fill up two 2D-arrays takes $2O(N^2)$.

Hence, total time comolexity is $O(N) + 2O(N^2) + O(N) = O(N^2)$.

```
// N is the length of HH−code
// input HH−code :  s with begin = index 1 and end  =  index N

ans ← 0
if N = 1 // input length = 1,  then answer is 1 clearly
        ans ← 1
        return

For i ← 1 to N // initial
        For j ← 1 to N
                endzero[i][j] ← 0
                endone[i][j] ← 0

// fill first row of two arrays
fo ← 0, fz ← 0 // first 1 and 0 appear at fo and fz
For i ← 1 to N // find first position of 1 and 0
        if s[i] = 0 and fz = 0
                fz ← i
        if s[i] = 1 and fo = 0
                fo ← i

For i ← 1 to N
        if i ≥ fo
                endone[1][i] ← 1
        if i ≥ fz
                endzero[1][i] ← 1

// fill other entries
For i ← 2 to N
        For j ← i to N
                if s[j] = 0
                        endzero[i][j] ← endzero[i − 1][j − 1] + endone[i − 1][j − 1]
                endone[i][j] ← endone[i][j − 1]
                else if s[j] = 1
                        endone[i][j] ← endzero[i − 1][j − 1] + endone[i − 1][j − 1]
                endzero[i][j] ← endzero[i][j − 1]

// calculate the answer
For i ← 1 to N
        ans ← ans + endzero[i][N]
        ans ← ans + endone[i][N]
// ans is the solution
```

2. reference: http://stackoverflow.com/questions/5151483/how-to-find-the-number-of-distinct-subsequences-of-a-string

$// \ N \ is \ the \ length \ of \ HH-code$
$// \ input \ HH-code: \ s \ with \ begin = index \ 1 \ and \ end \ = \ index \ N$
$// \ last[0] \ and \ last[1] \ record \ last \ postition \ of \ 0 \ and \ 1 \ respectively$
$// \ dp[i] = the \ number \ with \ distinct \ subsequences \ of \ first \ i \ characters$

$For \ i \leftarrow 0 \ to \ N \ // \ initial$
$\quad dp[i] \leftarrow 0$
$last[0] \leftarrow 0 \ and \ last[1] \leftarrow 0$
$dp[0] \leftarrow 1 \ // \ Assume \ empty \ set \ is \ a \ subsequence, \ so \ the \ result \ must \ minus \ 1$

$For \ i \leftarrow 1 \ to \ N$
$\quad dp[i] \leftarrow dp[i-1] \times 2 \ // \ choose \ s[i] \ or \ not$
$\quad if \ last[s[i]] \ != \ 0 \ // \ If \ s[i] \ has \ appeared, \ remove \ redundance$
$\quad\quad dp[i] \leftarrow dp[i] - dp[last[s[i]]-1]$
$\quad last[s[i]] \leftarrow i$
$// \ dp[N]-1 \ is \ the \ answer$

Complexity = $\sum_{i=1}^{N} O(1) = O(N)$

3. dicuss with B03902048

The algorithm is the same as 1, but it only needs to fill 1st to Kth row of two arrays. That is, it takes $O(KN)$ to fill it, and the answer is $endzero[K][N] + endone[K][N]$. Hence, the complexity is $O(N) + 2O(KN) + O(1) = O(KN)$.

Problem 3

1. (a) Multiply two $m \times m$ matrices takes $m \times m \times m = m^3$ time, which we have proved in class. As for exponentiation, we can use divide and conquer method, split n into half until it is equal to 1.

$while \ n \ > \ 0 \ // \ O(log \ n)$
$\quad if \ n \ is \ even$
$\quad\quad A \leftarrow A \times A \ // \ O(m^3)$
$\quad\quad n \leftarrow n/2$
$\quad else$
$\quad\quad ans \leftarrow ans \times A \ // \ O(m^3)$
$\quad\quad n \leftarrow n-1$
$// \ ans \ is \ A^n$

Then the complexity can be analyzed.
$T(m, \ n) = O(m^3) \times O(\log n) = O(m^3 \log n)$

Let $f(n) = \sum_{i=0}^{n} A^i \ and \ f(0) = I, \ f(1) = A. \ Then$

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

$\Rightarrow$ $$\begin{bmatrix} f(n) \\ I \end{bmatrix} = \begin{bmatrix} A & I \\ 0 & I \end{bmatrix}^n \begin{bmatrix} f(0) \\ I \end{bmatrix}$$

Hence, we can apply A^n algorithm to solve $\begin{bmatrix} A & I \\ 0 & I \end{bmatrix}^n$. And

two 2×2 matrices with each entry is a $m \times m$ matrix multiplication takes $O(1) \times O(m^3) = O(m^3)$.

Then the total complexity is $O(m^3) \times O(\log n) = O(m^3 \log n)$.

(b) discuss with B03902048, B03902056

Let $dp_i(l)$ represents the number of DNA with length l ending with n_i nucleotide and corresponding to Saki's findings. Hence, the number of existing DNA with length l is $\sum_{j=1}^{K} dp_j(l)$, which takes $O(K)$ to sum it up. Then the relation can be written as:

$$\begin{bmatrix} dp_1(l) \\ dp_2(l) \\ \vdots \\ dp_K(l) \end{bmatrix} = M \begin{bmatrix} dp_1(l-1) \\ dp_2(l-1) \\ \vdots \\ dp_K(l-1) \end{bmatrix} = M^l \begin{bmatrix} dp_1(0) \\ dp_2(0) \\ \vdots \\ dp_K(0) \end{bmatrix}$$

where M is a $K \times K$ matrix and each entry is either 0 or 1. The ith row in M means n_i nucleotide, and if n_i can be a neighbor of n_j then $M[i][j]$ is 1; otherwise, $M[i][j]$ is 0. Hence, it takes $O(K^2) + O(Q) = O(K^2)$ time complexity to go through Q pairs to inital and determine matrix M since $Q_{max} = O(K^2)$.

$For\ i \leftarrow 1\ to\ K$

$\quad For\ j \leftarrow 1\ to\ K$

$\quad\quad M[i][j] \leftarrow 1$

$For\ i \leftarrow 1\ to\ Q$

$\quad M[\beta_{i,1}][\beta_{i,2}] \leftarrow 0$

$\quad M[\beta_{i,2}][\beta_{i,1}] \leftarrow 0$

Suppose x_i denotes the position i in the DNA with length λ. Since there are P postitions that are immutable, we can partition a DNA sequence into P+1 subsequences, which are $\{x_1, x_2, \dots x_{a_1-1}\}$, $\{x_{a_1+1}, x_{a_1+2} \dots x_{a_2-1}\}$ .... $\{x_{a_p+1}, x_{a_p+1} \dots x_\lambda\}$ with length $\lambda_i \le R$ respectively and then these parts will share the same matrix M because each subsequence must follow the rule. Then we can find the number of DNA sequences in each part, and multiply them (takes $O(P)$) into the final answer. Hence it takes $\sum_{i=1}^{P+1} O(K^3 \log \lambda_i)$ to perform fast matrix exponentiation on $P+1$ parts of DNA. Then the total complexity is

$$O(K^2) + \sum_{i=1}^{P+1} (O(K^3 \log \lambda_i) + O(K)) + O(P) = O(K^3 P \log R)$$

2. (a) Since a_j is decreasing, $\overline{x}(j, k)$ always exists.

For all x such that $f_k(x) \le f_j(x)\ where\ j < k.$

$\Rightarrow$ $a_k x + b_k \le a_j x + b_j$

$\Rightarrow$ $(a_j - a_k)x + (b_j - b_k) \ge 0,\ where\ a_j - a_k > 0$

$\Rightarrow$ $x \ge \frac{b_k - b_j}{a_j - a_k}$

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

(b) reference: http://wcipeg.com/wiki/Convex_hull_trick

Actually, $\alpha_t = \overline{x}(j_t,\ j_{t+1})$ for all $t \geq 1$.

Deque: Bottom ________________ front

$j_1 \quad j_2 \quad .... \quad j_m$

____

Suppose the deque maintains one sequence $\{j_1,\ j_2,\ ..j_m\}$, and the end with j_1 is bottom, the other is front. When adding $f_k(x)$ into the deque, calculate $\overline{x}(j_u,\ k)$ from $u \leftarrow m$ down to 1 until $u = t$ such that $\overline{x}(j_t,\ k) > \alpha_{t-1}$. Next, pop $j_m,\ j_{m-1}..j_{t+1}$ from the front and delete $\alpha_m = \infty,\ \alpha_{m-1}...\alpha_t$ from $< \alpha >$. Then let $j_{t+1} \leftarrow k$ and push it into deque from the front. And then add $\alpha_t \leftarrow \overline{x}(j_t,\ k)$ and ∞ to $< \alpha >$ such that $< \alpha >$ is ordered. Finally, to calculate $dp(i) = \min\limits_{j<i} f_j(x_i)$, pop j_l from $l \leftarrow 1$ up to t from the bottom until $l = \gamma - 1$ such that $\alpha_{\gamma-1} \leq x_i \leq \alpha_\gamma$. And we need to delete α_0 to $\alpha_{\gamma-2}$ to update $< \alpha >$. Then $f_{j_\gamma}(x_i)$ is the answer of $dp(i)$.

Prove:

I. Pop from the front: update sequences so that the property can hold.

Since if $x_i \geq \overline{x}(a,\ b)$, then for $dp(i)$, $f_b(x_i) \leq f_a(x_l)$, so it would be better to transfer from b rather than a. Then when $\overline{x}(j_u,\ k) \leq \alpha_u = \overline{x}(j_u,\ j_{u+1})$, I claim that $f_k(x)$ is the minimum one among $f_k(x), f_{j_u}(x)\ and\ f_{j_{u+1}}(x)$ when $x \geq \overline{x}(j_u,\ k)$.

Prove: There are two cases.

Case I: $\overline{x}(j_u,\ k) \leq x \leq \overline{x}(j_u,\ j_{u+1})$

Since $\overline{x}(j_u,\ k) \leq x$, $f_k(x) \leq f_{j_u}(x)$.

Since $x \leq \overline{x}(j_u,\ j_{u+1})$, $f_{j_u}(x) \leq f_{j_{u+1}}(x)$

$\Rightarrow f_k(x) \leq f_{j_u}(x) \leq f_{j_{u+1}}(x)$

Case II: $\overline{x}(j_u,\ k) \leq \overline{x}(j_u,\ j_{u+1}) \leq x$

Since $\overline{x}(j_u,\ k) \leq x$, $f_k(x) \leq f_{j_u}(x)$.

Since $\overline{x}(a,\ c)$ is always between $\overline{x}(a,\ b)$ and $\overline{x}(b,\ c)$ for $a < b < c$ (simply by gragh), then $\overline{x}(j_{u+1},\ k) \leq \overline{x}(j_u,\ k) \leq x$.

Therefore, $f_k(x) \leq f_{j_{u+1}}(x)$.

Hence, the claim is true. Then we can delete α_u and j_{u+1} since $f_{j_u}(x)$ and $f_{j_{u+1}}(x)$ are not minimum when $x \geq \overline{x}(j_u,\ k)$ (but do not delete j_u since $f_{j_u}(x)$ may be minimum when $x \leq \overline{x}(j_u,\ k)$).

II. Pop from the bottom: delete useless α and j

Since x_i is increasing, then $dp(i+1)$ should be found from x_i. Hence, delete all $\alpha_t,\ j_t$ such that $t \neq 0$ and $\alpha_{t+1} < x_i$.

Analysis:

Each element of two sequences in the deque is only pushed or popped once. And to calculate $\overline{x}(j,\ k)=\frac{b_k-b_j}{a_j-a_k}$ and to compare $\overline{x}(j,\ k)$ with α take $O(1)$. Hence, the complexity is the max size of the deque = $O(n)$.

(c) reference: http://wenku.baidu.com/view/ef259400bed5b9f3f90f1c3a.html?re=view

discuss with B03902048

將座標軸倒置，原點置於最東方：以下的$c_i,\ a_i$其實是原本題目中$c_{n-i},\ a_{n-i}$。設position i的x座標為x_i，且**dp(i)表示在第i個點放置飼料，x_0到x_i的魚和人所消耗的體力最小值**，因此dp(n)為在第n個點放飼料(必定，否則x_n的魚吃不到飼料)，x_0到x_n的魚和人所消耗的體力最小值，即為所求。填dp表格時，必須由小填到大，則我們能得到以下關係式：

$dp(i)=\min\limits_{j<i}(dp(j)+cost(j,\ i)+c_i)$

$cost(j,\ i)=\sum\limits_{k=j+1}^{i}a_k(x_i-x_k)$ (在x_{j+1}到x_i區間的魚游到x_i的花費)

$=\sum\limits_{k=j+1}^{i}a_k(x_i-x_k)=x_i\sum\limits_{k=j+1}^{i}a_k-\sum\limits_{k=j+1}^{i}x_ka_k$

Let $suma(x)=\sum\limits_{i=1}^{x}a_i,\ sum(x)=\sum\limits_{i=1}^{x}x_ia_i$

$\Rightarrow cost(j,\ i)=x_i(suma(i)-suma(j))-(sum(i)-sum(j))$

$=-x_i*suma(j)+sum(j)+x_i*suma(i)-sum(i)$

$\Rightarrow$ $dp(i)=\min\limits_{j<i}(dp(j)+cost(j,\ i)+c_i)$

$=\min\limits_{j<i}(dp(j)-x_i*suma(j)+sum(j))+x_i*suma(i)-sum(i)+c_i$

Let $d_j=-suma(j)$. When calculating $dp(i)$ $x_i*suma(i)-sum(i)+c_i$ is a constant k since i is given.

Then $b_j=dp(j)+sum(j)+k$.

Therefore, $dp(i)=\min\limits_{j<i}d_jx_i+b_j$.

$<suma(i)=\sum\limits_{k=1}^{i}a_k>$ is increasing since $a_k>0,\ \forall k$, and then $<d_i=-suma(i)>$ is a decreasing sequence. $<x_i>$ is increasing since it represents the x-coordinate of i. As for parameters calculating, we can calculate all $sum(i)$ and $suma(i)$ for $0\le i\le n$ and put them into arrays before filling dp array which takes $O(n)$ to do it. Then when calculating $dp(i)$, all parameters can be accessed in $O(1)$ time.

Hence, $dp(i)=\min\limits_{j<i}d_jx_i+b_j$ has the same form as equation (1), and all the parameters can be calculated in $O(1)$.

Therefore, the complexity is $O(n)+O(n)=O(n)$, and the answer is $dp(n)$.