PHẦN I. MỞ ĐẦU

Dạng *bài toán về cấu hình tập hợp* không phải dạng mới lạ trong môn Tin học, đã xuất hiện tương đối nhiều trong các năm trở lại đây, trong các đề HSG QG và đề thi vòng 2. Các bài tập thuộc dạng này cài đặt thường phức tạp, dễ sai khiến học sinh lúng túng khi gặp bài toán như vậy.

Đây là một phương pháp rất hiệu quả để giải nhiều bài toán tối ưu và số lượng các bài toán tin học giải bằng phương pháp này cũng rất lớn.

Các bài toán về cấu hình tập hợp thường có dạng như sau: Cho một tập hợp các cấu hình thỏa mãn điều kiện nào đó được sắp xếp theo thứ tự từ điển, cần thực hiện các yêu cầu:

- Đếm số lượng phần tử của tập hợp trên.
- Cho một phần tử thuộc tập hợp trên, tìm thứ tự từ điển của phần tử đó.
- Cho thứ tự từ điển tìm phần tử thuộc tập hợp trên có thứ tự từ điển đó.

Nhận thấy đây là một chuyên đề thú vị và rất cần cho những học sinh tham gia các kì thi học sinh giỏi. Vì vậy, tôi quyết định tìm hiểu và viết chuyên đề: "*Các bài toán về cấu hình tập hợp*".

MỤC LỤC

PHẦN II. NỘI DUNG

I. CÁC BÀI TOÁN CƠ BẢN

Ta sẽ tìm hiểu các bài toán về cấu hình tập hợp thông qua ba bài toán cơ bản sau đây.

1. Bài toán 1

Cho tập hợp tất cả các xâu nhị phân có độ dài N được sắp xếp theo thứ tự từ điển. Ví dụ N = 3 ta có tập hợp: 000, 001, 010, 011, 100, 101, 110, 111.

Trong tập hợp trên, ta có:

- Xâu nhị phân 100: ở vị trí số 5.
- Vị trí số 7 trong tập là xâu nhị phân: 110.

Yêu cầu: Với một số N (N <= 63) cho trước, hãy cho biết:

- Xâu nhị phân S (có độ dài N) nằm ở vị trí nào của tập.
- Vị trí thứ K ($1 <= K <= 10^{18}$) là xâu nhị phân nào?

Dữ liệu vào: Cho trong file **NHIPHAN.INP**:

- Dòng đầu chứa một số nguyên N, là độ dài của các xâu nhị phân.
- Dòng thứ hai chứa một xâu nhị phân S có độ dài bằng N.
- Dòng thứ ba chứa một số nguyên K.

Dữ liệu ra: In ra file **NHIPHAN.OUT**:

- Dòng đầu tiên chứa một số nguyên là vị trí của S trong tập hợp.
- Dòng thứ hai ghi xâu nhị phân có N phần tử nằm ở vị trí thứ K.

NHIPHAN.INP	NHIPHAN.OUT
3 100 7	5 110

1.1. ***Phân tích****:*

Thực tế vị trí của xâu nhị phân S trong tập chính là dạng thập phân của S cộng thêm 1. Tuy nhiên để làm quen với Quy hoạch động vị trí cấu hình, ta sẽ làm theo cách sau:

a. Xác định số lượng cấu hình thuộc tập trên khi biết một đoạn các tiền tố của các cấu hình đó.

Giả sử ta biết được một đoạn tiền tố $a_1a_2…a_K$ (K <= N) của một xâu nhị phân n phần tử. Vấn đề đặt ra là có bao nhiêu xâu thuộc tập hợp có đoạn tiền tố như trên.

Với bài toán trên ta dễ dàng nhận thấy rằng: Số bit nhị phân còn lại chưa biết là (N – K) bit mà mỗi bit chỉ nhận giá trị 0 hoặc 1 nên số phần tử thỏa mãn là 2^{N-K} .

Vậy gọi F[i] là số lượng phần tử thỏa mãn khi biết (N – i) phần tử tiền tố.

F[0] = 1

F[i] = F[i – 1] * 2

b. Xác định xâu nhị phân S nằm vị trí nào?

Ta xét xâu nhị phân S: $a_1a_2a_3\ldots a_{N-1}a_N$.

Ta chỉ quan tâm đến các bit 1 của S, giả sử bit 1 xuất hiện ở vị trí t thì S: $a_1a_2\ldots a_{t-1}1\ \ldots a_{N-1}a_N$

Khi đó S sẽ có vị trí lớn hơn vị trí những cấu hình có dạng là $a_1a_2\ldots a_{t-1}\ 0\ x_{t+1}\ldots x_{n-1}x_n$(trong đó x_i nhận các giá trị 0 hoặc 1). Như đã tính ở trên thì sẽ có F[N - t] cấu hình có dạng $a_1a_2\ldots a_{t-1}\ 0\ x_{t+1}\ldots x_{N-1}x_N$ có vị trí bé hơn S.

Gọi vt là số lượng các cấu hình có vị trí bé hơn S.

vt = $\sum$ F[N - t_i], với t_i là vị trí xuất hiện các bit 1 trong S.

Suy ra, vị trí của S trong tập hợp là vt + 1.

c. Xác định xâu nhị phân nào nằm ở vị trí K?

Để tìm cấu hình S ở vị trí K, ta duyệt qua tất các bit của S từ 1 đến N. Nếu bit thứ t của S nhận giá trị 1 : $a_1a_2\ldots a_{t-1}1\ \ldots a_{N-1}a_N$ thì rõ ràng S lớn hơn F[t – 1] cấu hình có dạng $a_1a_2\ldots a_{t-1}\ 0\ x_{t+1}\ldots x_{N-1}x_N$. Thế nên:

- Nếu k <= F[N – t] thì bit thứ t của S nhận giá trị 0.
- Nếu k > F[N – t] thì bit thứ t của S nhận giá trị 1 và gán lại k = k – F[N – t].

1.2. Cài đặt:

```
#include<bits/stdc++.h>
usingnamespace std;

typedeflonglongLL;
typedef pair<int, int>II;

int main() {
freopen("NHIPHAN.inp", "r", stdin);
    freopen("NHIPHAN.out", "w", stdout);

int n;
    string S;
LL k;

    cin >> n;
    cin >> S;
    cin >> k;

// Init
LL f[64];
    f[0] = 1;
for (int i = 1; i <= 63; ++i) {
```

```
        f[i] = f[i - 1] * 2;
    }

// Position of S
LL ans = 0;
for (int i = 0; i < n; ++i) {
if (S[i] == '1') {
            ans += f[n - (i + 1)];
        }
    }
    cout << (ans + 1) << endl;

// k-th binary number
for (int i = 1; i <= n; ++i) {
if (k <= f[n - i]) {
            cout << 0;
        } else {
            cout << 1;
            k -= f[n - i];
        }
    }
    cout << endl;
return 0;
}
```

1.3 Test:

Đường dẫn: *https://drive.google.com/file/d/1KiYx57ntfygQddU6XukJIoTZzYQR9wsr/view?usp=sharing*

2. Bài toán 2

Một tập hợp S gồm các dãy N bit 0, 1 trong đó không có hai bit 1 nào kề nhau. Ví dụ N = 5 thì S gồm các dãy 00000, 00001, 00101,… Tập S được sắp xếp theo thứ tự từ điển.

Yêu cầu: Cho một số nguyên N (N < 63) cho biết:

- Xâu nhị phân S (có độ dài N) nằm ở vị trí nào của tập.
- Vị trí thứ K (K <= 10^{18}) là xâu nhị phân nào?

Dữ liệu vào: Cho trong file **NHIPHAN2.INP**:

- Dòng đầu chứa một số nguyên N, là độ dài của các xâu nhị phân.
- Dòng thứ hai chứ một xâu nhị phân S có độ dài bằng N.
- Dòng thứ ba chứa một số nguyên K.

Dữ liệu ra: In ra file **NHIPHAN2.OUT**:

- In trên từng dòng là kết quả từng yêu cầu của bài toán.

NHIPHAN2.INP	NHIPHAN2.OUT
5 00001 3	2 00010

2.1 Phân tích:

a. Xác định số lượng cấu hình thuộc tập trên khi biết một đoạn các tiền tố của các cấu hình đó.

Gọi F[i] là số lượng dãy nhị phân có i bit mà không có hai bit 1 liền kề nhau.

F[0] = 1, F[1] = 2.

F[i] = F[i – 1] (nếu chọn bit i = 0) + F[i – 2] (nếu chọn bit i = 1)

Giả sử ta biết được một đoạn tiền tố K phần tử $a_1a_2…a_K\ x_{K+1}…x_{N-1}x_N$ (K <= N) của một xâu nhị phân n phần tử thì số lượng cấu hình phù hợp là:

- Nếu ***bit thứ*** K = 0 thì sẽ có F[N – K] cấu hình phù hợp.
- Nếu ***bit thứ*** K = 1 thì sẽ có F[N – K – 1] cấu hình phù hợp do bit thứ K + 1 phải bằng 0.

b. Xác định xâu S nằm ở vị trí nào và xâu nào nằm ở vị trí K ?

Ta nhận thấy hai yêu cầu trên làm tương tự như bài toán 1.

2.2 Cài đặt:

```
#include<bits/stdc++.h>
usingnamespace std;

typedeflonglongLL;
typedef pair<int, int>II;

int main() {
freopen("NHIPHAN2.inp", "r", stdin);
    freopen("NHIPHAN2.out", "w", stdout);

int n;
    string S;
LL k;

    cin >> n;
    cin >> S;
    cin >> k;

// Init
LL f[64];
    f[0] = 1;
    f[1] = 2;
for (int i = 2; i <= n; ++i) {
        f[i] = f[i - 1] + f[i - 2];
    }
```

```
// Position of S
LL ans = 0;
for (int i = 0; i < n; ++i) {
if (S[i] == '1') {
int remain = n - (i + 1);
ans += f[remain];
        }
    }
    cout << (ans + 1) << endl;

// k-th element
for (int i = 1, p = 0; i <= n; ++i) {
for (int d = 0; d < 2; ++d) {
if (d == 1 && p == 1) {
break;
            }
LL g = (d == 0) ? f[n - i] : f[max(0, n - i - 1)];
if (k <= g) {
                cout << d;
break;
            } else {
                k -= g;
            }
        }
    }
    cout << endl;
return 0;
}
```

2.3 Test:

Đường dẫn:
https://drive.google.com/file/d/19KMliMNFJ0yIltvakgCCmayJo4eUM6Fw/view?usp=sharing

3. Bài toán 3:

Xét tất cả các hoán vị của dãy số tự nhiên (1, 2, …, N) (1 <= N <= 20), các hoán vị được sắp xếp theo thứ tự từ điển.

Yêu cầu:

- Cho trước 1 hoán vị: Tìm số hiệu hoán vị đó trong dãy đã sắp xếp.
- Cho trước số hiệu của 1 hoán vị trong dãy đã sắp xếp, tìm hoán vị đó.

Dữ liệu vào: Cho trong file **SHHV.inp**:

- Dòng đầu tiên chứa số nguyên dương N.
- Dòng thứ hai chứa N số $a_1, a_2, \ldots, a_N$ (dãy hoán vị N phần tử).
- Dòng thứ ba chứa số K (số hiệu hoán vị trong dãy N phần tử).

Dữ liệu ra: In ra file **SHHV.out**:

- Dòng đầu tiên ghi số K (số hiệu hoán vị của dãy a_i).
- Dòng thứ hai ghi N số b_1, b_2, …, b_N (dãy hoán vị có số hiệu K).

SHHV.INP	**SHHV.OUT**
3 2 1 3 4	3 2 3 1

3.1 Phân tích

a. Xác định số lượng cấu hình thuộc tập trên khi biết một đoạn các tiền tố của các cấu hình đó.

Giả sử ta biết được một đoạn tiền tố K phần tử $a_1a_2…a_K\ x_{K+1}…x_{N-1}x_N$ (K <= N) của một hoán vị N phần tử thì số lượng cấu hình phù hợp là: (N – K)! do ta có N – K số chưa dùng có thể điền vào bất kì vị trí nào còn trống.

Gọi F[i] là số lượng hoán vị có i chữ số.

F[i] = i! hay F[i] = F[i – 1] * i

b. Xác định vị trí của dãy hoán vị cho trước:

Ta xét hoán vị S: $a_1a_2…a_{N-1}a_N$.

Để biết vị trí K của hoán vị S, ta duyệt qua tất cả các phần tử của S từ 1 đến N. Giả sử ta đang xét đến phần tử t của S thì ta được một đoạn tiền tố S cố định: $a_1a_2…a_t\ x_{t+1}…x_N$ với ($x_{t+1}…x_N$ là một hoán vị của các phần tử chưa xuất hiện trong t phần tử đầu).

Nếu tồn tại 1 phần tử M chưa xuất hiện trong t phần tử đầu và $M < a_t$ (1) thì ta thấy $a_1a_2…a_t\ x_{t+1}…x_N > a_1a_2…M\ x_{t+1}…x_N$. Số lượng cấu hình thõa mãn $a_1a_2…M\ x_{t+1}…x_N$ là F[N – T]. Vậy cứ tương ứng với mỗi M thõa mãn (1) thì S có số hiệu hoán vị lớn hơn F[N – t] cấu hình khác.

Gọi vt là số lượng cấu hình bé hơn S:

vt = ∑ find(t) * F[N – t], với find(t) là số lượng các số chưa xuất hiện trong t phần tử đầu và có giá trị bé hơn a_t.

Suy ra, vị trí của S trong tập hợp là vt + 1.

c. Xác định hoán vị có vị trí cho trước:

Để tìm cấu hình S ở vị trí K, ta duyệt qua tất các phần tử của S từ 1 đến N. Nếu phần tử thứ t của S nhận giá trị M: $a_1a_2…a_{t-1}M\ …a_{N-1}a_N$ thì rõ ràng S lớn hơn find(t) * F[N - t] cấu hình có dạng $a_1a_2…a_{t-1}\ 0\ x_{t+1}…x_{N-1}x_N$. Thế nên, ta duyệt qua tất cả các phần tử M chưa xuất hiện trong t – 1 phần tử đầu theo thứ tự từ bé đến lớn:

- Nếu K <= F[N – t] thì phần tử thứ t của S nhận giá trị M.
- Nếu k > F[N – t] thì gán lại k = k – F[N – t].

3.2 Cài đặt:

```
#include<bits/stdc++.h>
usingnamespace std;

typedeflonglongLL;
typedef pair<int, int>II;

int main() {
freopen("SHHV.inp", "r", stdin);
    freopen("SHHV.out", "w", stdout);

int n, a[21];
LL k;

    cin >> n;
for (int i = 1; i <= n; ++i) {
        cin >> a[i];
    }
    cin >> k;

// Init
LL fact[21];
int mark[21];
    fact[0] = 1;
for (int i = 1; i <= 20; ++i) {
        fact[i] = fact[i - 1] * i;
    }
for (int i = 1; i <= n; ++i) {
        mark[i] = 0;
    }

// Position of a[]
LL ans = 0;
for (int i = 1; i <= n; ++i) {
int c = 0;
for (int v = 1; v <= a[i] - 1; ++v) {
if (mark[v] == 0) {
                c += 1;
            }
        }
        ans += c * fact[n - i];
        mark[a[i]] = 1;
    }
    cout << (ans + 1) << endl;

// k-th permutation
for (int i = 1; i <= n; ++i) {
        mark[i] = 0;
```

```
    }
for (int i = 1; i <= n; ++i) {
for (int v = 1; v <= n; ++v) {
if (mark[v] == 1) {
continue;
            }
if (k <= fact[n - i]) {
                cout << v <<" ";
                mark[v] = 1;
break;
            } else {
                k -= fact[n - i];
            }
        }
    }
    cout << endl;
return 0;
}
```

3.3 Test

Đường dẫn:

https://drive.google.com/file/d/1zNpO66c4RXbiwrrPHp76xrZDAobZHD9I/view?usp=sharing

II. BÀI TẬP ỨNG DỤNG

Bài 1: Counting Digits

1.1 Đề bài:

Nguồn:https://vn.spoj.com/problems/MDIGITS

Cho hai số nguyên a, b. Viết tất cả các số nằm giữa a, b; tính cả 2 số này.
Tính xem mỗi chữ số 0, 1, .., 9 mỗi số xuất hiện bao nhiêu lần.
Ví dụ, nếu a = 1024 và b = 1032, dãy sẽ là 1024 1025 1026 1027 1028 1029 1030 1031 1032 và có 10 số 0, 10 số 1, 7 số 2, ...

Dữ liệu vào: Cho trong file **MDIGITS.inp**:

- Không quá 500 dòng. Mỗi dòng là hai số nguyên a,b với
- 0 < a, b < 100000000.
- Kết thúc là hai số 0 0.

Dữ liệu ra: In ra file **MDIGITS.out**:

- Với mỗi bộ test, in ra một dòng 10 số nguyên là số chữ số 0, 1, .. 9 tương ứng.

MDIGITS.inp	MDIGITS.out
1 10	1 2 1 1 1 1 1 1 1 1
44 497	85 185 185 185 190 96 96 96 95 93
346 542	40 40 40 93 136 82 40 40 40 40
1199 1748	115 666 215 215 214 205 205 154 105 106
1496 1403	16 113 19 20 114 20 20 19 19 16
1004 503	107 105 100 101 101 197 200 200 200 200
1714 190	413 1133 503 503 503 502 502 417 402 412
1317 854	196 512 186 104 87 93 97 97 142 196
1976 494	398 1375 398 398 405 499 499 495 488 471
1001 1960	294 1256 296 296 296 296 287 286 286 247
0 0	

1.2 Hướng dẫn giải thuật:

Bài toán có thể quy về đếm số lượng mỗi chữ số xuất hiện trong tất cả các số nhỏ hơn hoặc bằng N (gọi số lượng này là Cal(N)).
Khi đó kết quả là Cal(B) - Cal(A – 1).

Xét biểu diễn thập phân của N: $a_m a_{m-1} \ldots a_1$.
Đếm số lượng các chữ số xuất hiện trong các số một đoạn phần tử tiền tố và K phần tử hậu tố chưa biết $a_M a_{M-1} \ldots a_{K+1} \mid x_K x_{K-1} \ldots x_1$ (những phần tử a đã biết còn x chưa biết).

Xét phần tử x_K:

- Nếu chọn $x_K < a_K$ thì:
  - Số x_k xuất hiện ở vị trí thứ k 10^{K-1} lần.
  - Các chữ số 0…9 xuất hiện trong $x_{k-1}x_{k-2}…x_1$ mỗi số $10^{K-1}*(K-1) / 10 = 10^{K-2} * (K-1)$ lần.
- Nếu chọn $x_k = a_k$ thì:
  - Số x_k xuất hiện ở vị trí thứ k $a_{k-1}a_{k-2}…a_1$ lần.

Vậy duyệt qua tất cả các tiền tố của N thì ta tính được kết quả.

Do số 0 đứng đầu không có ý nghĩa nên phải trừ đi số lượng chữ số 0 vô nghĩa này đi. Số lượng số 0 vô nghĩa là: $\sum 10^k$ với $0 <= k < m$.

1.3 Chương trình:

```
#include<bits/stdc++.h>
usingnamespace std;

int p10[10];

vector <int> Cal(intn) {
    vector <int> res(10, 0);
int cnt = 0, x = 0;

while (n> 0) {
int d = n % 10;
n /= 10;

for (int i = 0; i < d; i++) {
            res[i] += p10[cnt];
for (int j = 0; j < 10; j++) res[j] += p10[cnt - 1] * cnt;
}

        res[d] += x + 1;
        x = d * p10[cnt] + x;
cnt++;
    }
for (int i = 0; i < cnt; i++) res[0] -= p10[i];
return res;
}
void query(inta, intb) {
    vector <int> Ca = Cal(a - 1);
    vector <int> Cb = Cal(b);

for (int i = 0; i < 10; i++) cout << Cb[i] - Ca[i]<<" ";
    cout << endl;
}
int main() {
    freopen("MDIGITS.inp", "r", stdin);
```

```
    freopen("MDIGITS.out", "w", stdout);

    p10[0] = 1;
for (int i = 1; i < 10; i++) p10[i] = p10[i - 1] * 10;

while (true) {
int a, b;
        cin >> a >> b;
if (a == 0 && b == 0) break;
if (a > b) swap(a, b);
        query(a, b);
    }
return 0;
}
```

1.4 Test:

Đường dẫn:

https://drive.google.com/file/d/1vE_hUNEne561f1z0T723RE1vHvG7YI7z/view?usp=sharing

1.5 Cảm nhận:

Đây là bài toán điển hình về cấu hình tập hợp, việc xác định số lượng dãy con phù hợp với một tiền tố cho trước khá đơn giản.

Có độ phức tạp đối với mỗi test là O(n * 10), với n là số chữ số của N.

Bài 2: Thứ tự dãy con.

2.1 Đề bài:

Nguồn: *https://vn.spoj.com/problems/C11SEQ2*

Cho dãy số nguyên A gồm N phần tử đôi một khác nhau. Từ dãy A chọn ra K phần tử và giữ nguyên thứ tự như trong A tạo thành một dãy con.Sắp xếp tất cả các dãy con K phần tử theo thứ tự từ điển.

Yêu cầu:

- Hãy tìm dãy con có thứ tự từ điển thứ M
- Cho dãy con K phần tử của dãy A. Hãy cho biết thứ tự từ điển của dãy con đó

Dữ liệu vào: Cho trong file **C11SEQ2.inp**:

- Dòng đầu gồm 2 số nguyên N và K. (1 <= K <= N <= 60).
- Dòng thứ 2 ghi N số nguyên a[1], a[2],…, a[n] (-10^6 <= a[i] <= 10^6).
- Dòng thứ 3 ghi số nguyên M (theo yêu cầu 1). (1 <= M <= 2^63)
- Dòng thứ 4 ghi K số nguyên là 1 dãy con của dãy A (theo yêu cầu 2).

Dữ liệu ra: In ra file **C11SEQ2.out**:

- Dòng 1: trả lời yêu cầu 1, ghi ra dãy con K phần tử tìm được, giữa 2 số có 1 khoảng trắng.
- Dòng 2: trả lời yêu cầu 2, thứ tự từ điển của dãy con đó.

C11SEQ2.inp	**C11SEQ2.out**
6 4 7 9 5 3 2 4 8 7 9 3 4	7 9 3 4 8

2.2 Hướng dẫn giải thuật:

Đặt C[i][k] là số cách chọn k phần tử trong dãy a từ phần tử thứ i đến thứ n.
Công thức truy hồi:

C[i][k] = C[i + 1][k] + C[i + 1][k – 1]

Gọi ans[] là dãy kết quả. Quy ước ans[0] = 0.

Để tìm số thứ i của dãy kết quả thì ta chỉ cần duyệt qua tất cả các số từ vị trí của ans[i – 1] + 1 trở đi theo thứ tự tăng dần:

+ Nếu M <= C[p + 1][K – i] thì chọn giá trị a[p] cho vào phần tử thứ i.

+ Nếu M > C[p + 1][K – i] thì gán lại M = M – C[p + 1][K – i].

Xác định thứ tự từ điển: Với mỗi phần tử thứ i ta duyệt qua các phần tử từ vị trí ans[i – 1] đến vị trí phần tử ans[i] có giá trị bé hơn ans[i], nếu thay mỗi phần tử ấy cho ans[i] thì ta đều nhận được các dãy có từ điển bé hơn nên cập nhật C[p +1][K – i] vào kết quả.

II.3 Chương trình:

```
#include<bits/stdc++.h>
usingnamespace std;

#definelllonglong
constint MAXN = 62;

int N, K;
int a[MAXN], b[MAXN], ans[MAXN];
ll M, c[MAXN][MAXN], res;
vector <int> adj[MAXN];

bool cmp(intu, intv) {
return a[u] < a[v];
}

int main() {
    freopen("C11SEQ2.inp", "r", stdin);
    freopen("C11SEQ2.out", "w", stdout);

    cin >> N >> K;
for (int i = 1; i <= N; i++) cin >> a[i];

for (int i = 0; i <= N + 1; i++) c[i][0] = 1;
for (int k = 1; k <= N; k++)
for (int i = N; i > 0; i--)
            c[i][k] = c[i + 1][k] + c[i + 1][k - 1];

for (int i = N; i > 0; i--) {
        adj[i] = adj[i + 1];
        adj[i].push_back(i);
        sort(adj[i].begin(), adj[i].end(), cmp);
    }

    cin >> M;
int tm = 1;
for (int k = 1; k <= K; k++) {
for (int i = 0; i < (int)adj[tm].size(); i++) {
int p = adj[tm][i];
if (c[p + 1][K - k] < M) M -= c[p + 1][K - k];
else {
                ans[k] = a[p];
                tm = p + 1;
```

```
break;
            }
        }
    }
for (int i = 1; i <= K; i++) cout << ans[i] <<" "; cout << endl;

for (int i = 1; i <= K; i++) cin >> b[i];
    tm = 1;
for (int k = 1; k <= K; k++) {
for (int i = 0; i < (int)adj[tm].size(); i++) {
int p = adj[tm][i];
if (a[p] < b[k]) res += c[p + 1][K - k];
else {
                tm = p + 1;
break;
            }
        }
    }
    cout << res + 1<< endl;

return 0;
}
```

2.4 Test:

Đường dẫn:

https://drive.google.com/file/d/1VGaXP4cN_FGFZ6xM8GjAk_fQV3pv74So/view?usp=sharing

2.5 Cảm nhận:

Đây là một bài toán về cấu hình tập hợp ở mức độ dễ yêu cầu cần thông hiểu phương pháp làm bài dạng này.

Độ phức tạp của bài toán là O(N * K).

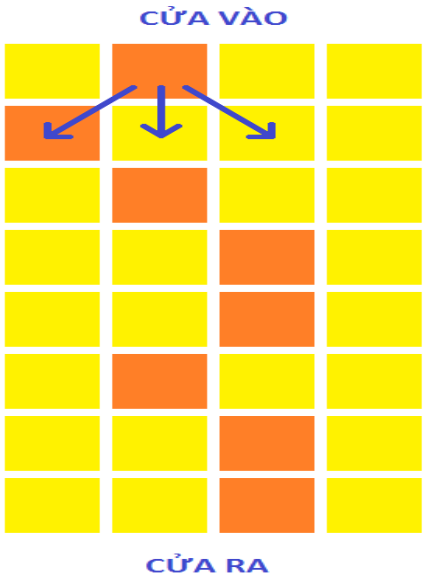
Bài toán 3: Bảo mật ngân hàng.

3.1 Đề bài:

Nguồn:https://vn.spoj.com/problems/VMCODE

Sau nhiều năm theo đuổi các giải thưởng của kì thi VNOI Marathon (VM), cuối cùng Bờm cũng kiếm được một số tiền khá khá. Để số tiền đó ngày một lớn hơn, Bờm quyết định mở một ngân hàng. Mọi việc đều hoàn thành xong xuôi, từ việc tuyển nhân viên, đăng kí kinh doanh, quảng cáo, ... đến việc xây dựng một hệ thống an toàn để cất trữ tiền bạc. Nhưng có một việc đang làm hao tốn rất nhiều thời gian của Bờm, "nhờ nó" mà đến giờ ngân hàng vẫn chưa đưa vào sử dụng được. Đó là việc suy nghĩ mật mã cho hệ thống an toàn.

Sau khi xem xét lại hệ thống an toàn Bờm nhận thấy rằng: Nó là một căn phòng hình chữ nhật trải dài M mét và rộng N mét. Căn phòng được lát bằng các viên gạch đơn vị 1x1 mét vuông. Các hàng gạch được đánh số từ 1 đến M, và các cột được đánh số từ 1 đến N; viên gạch ở hàng I cột J kí hiệu là (I, J). Để mở khóa, người dùng sẽ đứng ở cửa vào, đi M bước từ hàng đầu tiên đến hàng cuối cùng, phải qua ĐÚNG các ô gạch đã xác định làm mật mã (trong hình ví dụ ở dưới chính là các ô màu cam), và kết thúc ở cửa ra. Khi đứng ở viên gạch (I, J) do giới hạn độ dài một bước chân nên một người chỉ có thể đi đến được các viên gạch tại (I+1, J-1) (I+1, J) (I+1, J+1). Để hỗ trợ cho việc GHI NHỚ mật mã của người dùng, mỗi viên gạch - khi bước lên nó - sẽ phát ra một nốt nhạc có độ cao xác định.



Sau khi nghịch, Bờm thấy rằng nếu đi theo qui luật trên và theo một thứ tự nào đó, Bờm sẽ có được những bản nhạc rất hay. Thế là Bờm đi lục lại bản nhạc ưa thích của mình thuở nhỏ (thật may mắn là nó có đúng M nốt nhạc). Bờm dự tính sẽ chọn nó làm mật mã an toàn, nhưng làm như thế sẽ có ngày kẻ xấu tìm ra mật mã và đột nhập vào kho bạc ngân hàng. Vì thế nên Bờm đã nghĩ ra một cách: chọn một số K, trong những bản nhạc tạo được từ cách đi trên và có thứ tự từ điển lớn hơn bản nhạc ưu thích của Bờm, lấy ra bản nhạc có thứ tự từ điển lớn thứ K.

Bờm sức thì có hạn mà lại muốn tìm chính xác ra bản nhạc kia. Nên hôm nay Bờm nhờ đến kì thi VM13 giải quyết giùm bài toán này. Nếu giải đúng, Bờm sẽ đồng ý tài trợ và vì thế mà giải thưởng của các bạn có thể sẽ tăng lên rất nhiều đó!

Dữ liệu vào: Cho trong file **VMCODE.inp**:

- Dòng đầu tiên là ba số M, N, K;
- Dòng thứ hai là bản nhạc ưa thích của Bờm;
- M dòng tiếp theo, mỗi dòng N kí tự biểu diễn cho hệ thống an toàn của ngân hàng;

Dữ liệu ra: In ra file **VMCODE.out**:

- Một dòng duy nhất là bản nhạc Bờm cần tìm;

Giới hạn:

- $1 \leq M, N \leq 2.000$;
- $1 \leq K \leq 10^9$;
- Căn phòng chỉ gồm các kí tự la tinh 'a' đến 'z', đại diện cho độ cao của nốt nhạc phát ra của mỗi ô gạch. Độ cao 'a' < 'b' < ... < 'z';
- Bản nhạc A[1..M] có thứ tự từ điển nhỏ hơn B[1..M] khi và chỉ khi tồn tại số nguyên dương t nhỏ hơn hoặc bằng M sao cho A[t] < B[t] và A[i] = B[i] (với mọi i thỏa $1 \leq i < t$);

VMCODE.inp	VMCODE.out
3 3 4 efz ebr wqw yav	ewa

3.2 Hướng dẫn giải thuật:

Giả sử ta đã tạo dược xâu kết quả với các kí tự từ 1 đến (i – 1) và chúng ta cần tìm một trong các kí tự ở hàng i để điền thêm vào kết quả. Do đó ta cần tính nếu điền kí tự C nào đó vào thì có bao nhiêu cách khác nhau để tạo được xâu độ dài m có thứ tự từ điển lớn hơn xâu S, có i -1 kí tự đầu trùng với xâu đã tạo và kí tự thứ i là C. Sau khi tính được rồi, ta duyệt qua các kí tự từ ‘a’ đến ‘z’. Nếu đến kí tự C nào đó mà số cách tìm được V nhiều hơn hoặc bằng K thì ta gán kí tự C vào xâu kết quả ngược lại sẽ giảm K xuống V giá trị.

Ta xây dựng mảng F[i][j][k] với ý nghĩa:

F[i][j][0]: số cách khác nhau để tạo thành các xâu xuất phát từ ô (i, j).

F[i][j][1]: số cách khác nhau để tạo thành các xâu xuất phát từ ô (i, j) có thứ tự từ điển lớn hơn (m – i + 1) kí tự cuối cùng của xâu S.

Khởi tạo: với j = 1..n:

F[m][j][0] = 1

F[m][j][1] = 1 nếu a[m][j] > S[j]
F[m][j][1] = 0 nếu a[m][j] <= S[j]

Công thức truy hồi:

F[i][j][0] = F[i+1][j-1][0] + F[i+1][j][0] + F[i+1][j+1][0]

F[i][j][1] = F[i+1][j-1][0] + F[i+1][j][0] + F[i+1][j+1][0] nếu a[i][j] > s[i]

F[i][j][1] = F[i+1][j-1][1] + F[i+1][j][1] + F[i+1][j+1][1] nếu a[i][j] = s[i]

F[i][j][1] = 0 nếu a[i][j] < s[i]

Ta gọi mảng dp[C] là số cách khác nhau để tạo xâu độ dài m có thứ tự từ điển lớn hơn xâu S, có i-1 kí tự đầu tiên trùng với i-1 kí tự đầu của xâu kết quả và có kí tự thứ i là C.

- Với kí tự đầu tiên, ta xét các kí tự C có trong hàng đầu tiên của mảng A:
    - Nếu C > S[1] : dp[C] = sum(F[1][j][0]) (với A[1][j] = C).
    - Nếu C = S[1] : dp[C] = sum(F[1][j][1]) (với A[1][j] = C).
- Với các kí tự từ thứ 2 trở đi cách tính mảng dp sẽ khác một chút:

Trong quá trình tạo xâu kết quả đến vị trí thứ i, ta cần đồng thời xây dựng thêm một mảng G với ý nghĩa G[i][j] là số cách khác nhau để đi từ đầu đến ô (i,j) và tạo được xâu có (i-1) kí tự đầu trùng với (i-1) kí tự của xâu kết quả đã xây dựng được.

G[1][j] = 1 với j=1…n

G[i][j] = sum(G[i-1][j+x])

với x = -1..1 và G[i-1][j+x]=S[i-1]

Ta cần một biến logic ‘bigger’ để xét xem xâu đang xây dựng được có chắn chắn lớn hơn xâu S bất kể cách chọn các kí tự còn lại như thế nào hay chưa. Hay nói cách khác xâu kết quả tạo bởi i-1 kí tự đầu tiên đã có thứ tự từ điển lớn hơn xâu tạo bởi i-1 kí tự đầu tiên của xâu S hay chưa? (tức xâu tạo bởi các kí tự S[1] + S[2] + … + S[i]). Trong quá trình xây dựng các kí tự cho xâu kết quả, nếu ta thêm 1 kí tự C vào cho vị trí thứ i mà C > S[i] và bigger = False thì ta sẽ gán lại bigger = True.

Như vậy ta sẽ có cách tính mảng dp như sau:

- Nếu bigger = false:
    - Nếu C > S[i] : dp[C] = sum(G[i][j] * F[i][j][0]) (với A[i][j]=C)
    - Nếu C = S[i] : dp[C] = sum(G[i][j] * F[i][j][1]) (với A[i][j]=C)
- Nếu bigger = true:

- dp[C] = sum(G[i][j] * F[i][j][0]) (với A[i][j]=C)

3.3 Chương trình:

```
#include<bits/stdc++.h>
usingnamespace std;

#definelllonglong
constint MAXN = 2010;
constll modulo = (int) 1e9 + 7;

int m, n, k;
string s, ans, a[MAXN];
ll f[MAXN][MAXN][2], g[MAXN][MAXN], dp[256];

int main() {
    freopen("VMCODE.inp", "r", stdin);
    freopen("VMCODE.out", "w", stdout);

    cin >> m >> n >> k;
    cin >> s, s = " " + s;
for (int i = 1; i <= m; i++) cin >> a[i], a[i] = " " + a[i];

for (int j = 1; j <= n; j++) {
        f[m][j][0] = 1;
if (a[m][j] > s[m]) f[m][j][1] = 1;
else f[m][j][1] = 0;
    }

for (int i = m - 1; i > 0; i--) {
for (int j = 1; j <= n; j++) {
            f[i][j][0] = min(modulo, f[i + 1][j - 1][0] + f[i + 1][j][0]
+f[i + 1][j + 1][0]);
if (a[i][j] > s[i]) f[i][j][1] = f[i][j][0];
else
if (a[i][j] == s[i]) f[i][j][1] = min(modulo, f[i + 1][j - 1][1] + f[i + 1]
[j][1] + f[i + 1][j + 1][1]);
else f[i][j][1] = 0;
        }
    }

bool bigger = false;
for (int i = 1; i <= m; i++) {
for (int c = 'a'; c <= 'z'; c++) dp[c] = 0;
if (i == 1) {
for (int j = 1; j <= n; j++) {
char c = a[1][j];
if (c > s[1]) dp[c] = min(modulo, dp[c] + f[1][j][0]);
else
if (c == s[1]) dp[c] = min(modulo, dp[c] + f[1][j][1]);
```

```
            }
for (int j = 1; j <= n; j++) g[1][j] = 1;
        } else {
for (int j = 1; j <= n; j++)
for (int x = -1; x <= 1; x++)
if (a[i - 1][j + x] == ans[i - 2]) g[i][j] = min(modulo, g[i][j] + g[i - 1]
[j + x]);

for (int j = 1; j <= n; j++) {
char c = a[i][j];
if (bigger) dp[c] = min(modulo, dp[c] + g[i][j] * f[i][j][0]);
else
if (c > s[i]) dp[c] = min(modulo, dp[c] + g[i][j] * f[i][j][0]);
else
if (c == s[i]) dp[c] = min(modulo, dp[c] + g[i][j] * f[i][j][1]);
            }
        }

for (int c = 'a'; c <= 'z'; c++)
if (dp[c] < k) k -= dp[c];
else {
                ans.push_back(char(c));
if (c > s[i]) bigger = true;
break;
            }
    }

    cout << ans;
return 0;
}
```

3.4 Test:

Đường dẫn:

https://drive.google.com/file/d/16jC_Lk4hGWds6HVGmYLtD5j8fEFmT9u5/view?usp=sharing

3.5 Cảm nhận:

Là bài toán hay về cấu hình tập hợp, phải nắm vững kĩ thuật làm bài dạng này.

Việc xác định số lượng các cấu hình phù hợp với một tiền tố cho trước đòi hỏi phải nhìn sâu vào các trường hợp và có nền tảng quy hoạch động vững vàng. Ngoài ra khi code thì kết quả có thể vượt quá giới hạn số nguyên 32 bit nên cần giới hạn giá trị để nó không vượt quá một số nguyên nhất định.

Độ phức tạp của bài toán là O(M * N).

Bài 4: Chef and Digits

4.1 Đề bài:

Nguồn:[*https://www.codechef.com/problems/DGTCNT*](https://www.codechef.com/problems/DGTCNT)

Chef yêu thích các số nguyên, nhưng không phải tất cả các số nguyên. Vì một vài lí do cá nhân, anh ấy coi các số nguyên a_0, a_1, ..., và a_9 là không may mắn. Nếu như cho một số nguyên x, giả sử tồn tại một chữ số i ($0 \le i \le 9$) xuất hiện chính xác ai lần khi biểu diên s˜ ố x dưới i cơ số 10 (không có số 0 ở đầu) thì Chef coi đó là không may mắn, Ngược lại thì đều là những số Chef thích. Chef muốn tính xem có bao nhiêu số nguyên giữa L và R (bao gồm cả R và L) mà Chef yêu thích. Bạn hãy giúp Chef!

Dữ liệu vào: Cho trong file **DGTCNT.inp**:

- Dòng đầu tiên của dữ liệu vào chứa số nguyên T – số test, T test được miêu tả như sau:
- Dòng đầu tiên của mỗi test chứa hai số nguyên L, R.
- Dòng thứ hai của mỗi test chứa chính xác 10 số nguyên, số nguyên thứ i là a_{i-1}.

Dữ liệu ra: In ra file **DGTCNT.out:**

- Ở mỗi test, in ra một số nguyên duy nhất là đáp án.

Giới hạn:

- $1 \le T \le 20$
- $1 \le L \le R \le 1018$
- $0 \le a_i \le 18$

DGTCNT.inp	DGTCNT.out
2 21 28 5 4 3 2 1 1 2 3 4 5 233 23333 2 3 3 3 3 2 3 3 3 3	6 19627

4.2 Hướng dẫn giải thuật:

Xét bài toán đơn giản hơn: Đếm số lượng trong nửa khoảng [1; N). Khi đã giải được bài toán này, kết quả bài toán gốc đơn thuần chỉ là kết quả trên [1; R + 1) - kết quả trên [1; L).

Xét dạng biểu diễn của N:a_1 a_2 a_3 … a_nvới n là số chữ số của N.

Ta tiến hành bao hàm loại trừ theo hướng: với một tập S, số lượng số mà các chữ số i trong S xuất hiện đúng c[i] là bao nhiêu.

Ta xét lần lượt các tiền tố của N. Giả sử đã cố định tới tiền tố thứ (i - 1) của N. Khi đó ta phải điền vào a_i một giá trị $d < a_i$ để đảm bảo là số sinh ra nhỏ hơn N.

$a_1\ a_2\ a_3\ \ldots\ a_{i-1}\ d\ \ldots$

Khi đã thử điền giá trị d vào a_i, phần còn lại (các chữ số từ (i + 1) tới n) có thể điền bất kỳ mà không phụ thuộc vào N. Khi đó bài toán quy về là đếm số lượng số có (n - i) chữ số, các chữ số x trong tập S xuất hiện đúng c(x) lần. Bài toán này có thể giải bằng toán tổ hợp cơ bản:

- Gọi sum là tổng số lần xuất hiện của các chữ số trong S.
- Gọi cnt là số lượng chữ số trong S (cnt = |S|).
- Đặt len = n - i.
- Ta chọn ra sum vị trí trong len vị trí, đặt các chữ số cần thiết vào, các vị trí còn lại ta đặt bừa các chữ số còn lại.
- Như vậy công thức là:

$$C¿$$

Chú ý là ta còn phải xét các số có ít chữ số hơn N. Trường hợp này sẽ đơn giản hơn: ta cố định độ dài của số, cố định chữ số đầu của số (để không cần phải xét trường hợp bắt đầu bằng số 0), rồi áp dụng công thức như trên.

4.3 Chương trình:

```
#include<bits/stdc++.h>
usingnamespace std;

typedeflonglongLL;
typedef pair<int, int>II;

constint MAXN = 20;
LL fact[MAXN + 10];
LL C[MAXN + 10][MAXN + 10];

void Init() {
    fact[0] = 1;
for (int i = 1; i <= MAXN; ++i) {
        fact[i] = fact[i - 1] * i;
    }
for (int i = 0; i <= MAXN; ++i) {
        C[i][0] = 1;
for (int j = 1; j <= i; ++j) {
            C[i][j] = C[i - 1][j - 1] + C[i - 1][j];
        }
    }
}

int Convert(LLN, inta[]) {
```

```
int n = 0;
while (N> 0) {
a[++n] = N % 10;
N /= 10;
    }
    reverse(a + 1, a + 1 + n);
return n;
}

LL Count(intlen, intc[], intmask) {
int sum = 0;
int cnt = 0;
for (int i = 0; i < 10; ++i) {
if (mask>> i & 1) {
            sum += c[i];
            cnt += 1;
        }
    }
if (sum >len) {
return 0;
    }
LL ans = fact[sum];
for (int i = 0; i < 10; ++i) {
if (mask>> i & 1) {
            ans /= fact[c[i]];
}
    }
    ans = ans * C[len][sum];
for (int i = 0; i <len - sum; ++i) {
ans = ans * (10 - cnt);
    }
return ans;
}

LL Count(LLN, intc[]) {
int a[MAXN + 10];
int n = Convert(N, a);

LL ans = 0;
for (int mask = 0; mask < 1 << 10; ++mask) {
LL sum = 0;
int cnt[10];
for (int i = 0; i < 10; ++i) {
            cnt[i] = c[i];
        }
for (int i = 1; i <= n - 1; ++i) {
for (int d = 1; d < 10; ++d) {
if ((mask >> d & 1) == 1 && cnt[d] == 0) {
continue;
```

```
                }
                cnt[d] -= 1;
                sum += Count(i - 1, cnt, mask);
                cnt[d] += 1;
            }
        }
for (int i = 1; i <= n; ++i) {
for (int d = (i == 1) ? 1 : 0; d < a[i]; ++d) {
if ((mask >> d & 1) == 1 && cnt[d] == 0) {
continue;
                }
                cnt[d] -= 1;
                sum += Count(n - i, cnt, mask);
                cnt[d] += 1;
            }
if (cnt[a[i]] == 0 && (mask >> a[i] & 1) == 1) {
break;
            }
            cnt[a[i]] -= 1;
        }
if (__builtin_popcount(mask) % 2 == 0) {
ans += sum;
        } else {
            ans -= sum;
}
    }
return ans;
}

int main() {
    freopen("DGTCNT.inp", "r", stdin);
    freopen("DGTCNT.out", "w", stdout);
Init();

int TC;
    scanf("%d", &TC);
while (TC--) {
LL l, r;
int a[10];

        scanf("%lld%lld", &l, &r);
for (int i = 0; i < 10; ++i) {
            scanf("%d", &a[i]);
        }

        printf("%lld\n", Count(r + 1, a) - Count(l, a));
    }
return 0;
}
```

4.4 Test:

Đường dẫn:

https://drive.google.com/file/d/1bYEBSrqmWrNGn1cPoZ2hASKv2cjO-76b/view?usp=sharing

4.5 Cảm nhận:

Bài toán này đòi hỏi phải có kiến thức cơ bản về bao hàm loại trừ (inclusion-exclusion principle). Thực chất còn có một cách giải khác chỉ sử dụng quy hoạch động thuần túy, cùng tư tưởng nhưng nhìn chung thì không tự nhiên và rõ ràng như cách làm này.

Ngoài ra, bài toán còn yêu cầu có một lượng kiến thức chắc về toán tổ hợp cũng như kỹ năng cài đặt khéo léo, vì nếu không cẩn thận sẽ phải xét rất nhiều trường hợp (ví dụ như trường hợp số có ít hơn n chữ số, nếu không cẩn thận sẽ phải xét thêm trường hợp chữ số 0 ở đầu).

Độ phức tạp cho mỗi test là O(2^10 * n * 10 * 10), với n là số chữ số của (R + 1).

Bài 5: Xâu đối xứng .

5.1 Tóm tắt đề bài:

Nguồn: VOI17

Cho N, K và tập chữ cái O. Một xâu S độ dài N được gọi là đẹp nếu như.

- Có thể sắp xếp lại S để tạo thành một xâu palindrome.
- Tồn tại k vị trí $1 < i_1, i_2, ..., i_k < N$ sao cho $S[1..i_j]$ với mọi j có thể sắp xếp để trở thành một palindrome.

Cho T. Tìm xâu đẹp thứ T theo thứ tự từ điển.

5.2 Hướng dẫn giải thuật:

Trước tiên ta sắp xếp và loại trùng tập O. Đặt m = |O|.

Xét một xâu T chỉ gồm các ký tự trong O. Đặt mask(T) = một mask nhị phân, trong đó bit i = 1 tức là ký tự O[i] xuất hiện lẻ lần trong O, bit i = 0 nếu nó xuất hiện chẵn lần.

Dễ thấy là T có thể biến đổi thành palindrome khi và chỉ khi mask(T) = 0 (tất cả ký tự xuất hiện chẵn lần) hoặc mask(T) = 2^d (tức là chỉ có ký tự O[d] xuất hiện lẻ lần, đặt ký tự này ở chính giữa xâu). Điều kiện này có thể kiểm tra nhanh bằng công thức: (mask & -mask) == 0.

Quay trở lại bài toán chính.

Để cho đơn giản, ta xét luôn vị trí 1 và N vào tập các vị trí cần xét. Giờ, một xâu đẹp cần có (k + 2) vị trí như vậy, bao gồm cả vị trí 1 và N.

Giả sử ta đã cố định được i ký tự đầu của xâu kết quả, hiện đã có j vị trí thỏa mãn, mask hiện tại là x.

Đặt f(i, j, x) là số lượng xâu độ dài N thỏa mãn các điều kiện vừa nêu ra.

Dễ thấy là f(N, j, x) = 1 với mọi j >= k + 2 và x là mask thỏa mãn.

Công thức truy hồi:

f(i, j, x) = sum(f(i + 1, n_j, n_x)) với mọi d trong đoạn [0; m - 1]

Trong đó:
n_j = j + 1 nếu xâu hiện tại thêm ký tự d vào vẫn thỏa mãn.
n_j = j trong trường hợp ngược lại.
n_x = mask của xâu hiện tại khi ghép thêm ký tự d.

Để tính kết quả, ta lần lượt duyệt các vị trí từ trái qua phải của xâu kết quả rồi thử đặt từng ký tự vào, tính lại mask và cập nhật lại kết quả. Có thể tham khảo phần source code để hiểu rõ hơn (chú ý là trong code, tham số i trong hàm f() được định nghĩa là số ký tự cần thêm vào chứ không phải là số ký tự đã thêm).

Ngoài ra, dữ liệu của bài khá lớn nên việc cài số lớn là cần thiết.

5.3 Chương trình:

```
#include<bits/stdc++.h>
usingnamespace std;

typedeflonglongLL;
typedef pair<int, int>II;

typedef string BigInteger;

BigInteger Sum(BigIntegera, BigIntegerb) {
while (a.size() <b.size()) {
a = "0" + a;
    }
while (a.size() >b.size()) {
b = "0" + b;
    }
int n = (int)a.size();
int c = 0;
BigInteger ans = "";
for (int i = n - 1; i >= 0; --i) {
int d = (a[i] - 48) + (b[i] - 48) + c;
        c = d / 10;
ans = char(d % 10 + 48) + ans;
    }
if (c > 0) {
        ans = char(c + 48) + ans;
    }
return ans;
}

BigInteger Subtract(BigIntegera, BigIntegerb) {
while (a.size() >b.size()) {
b = "0" + b;
    }
int n = (int)a.size();
int c = 0;
BigInteger ans = "";
for (int i = n - 1; i >= 0; --i) {
int d = (a[i] - 48) - (b[i] - 48) - c;
if (d < 0) {
d += 10;
            c = 1;
        } else {
            c = 0;
        }
        ans = char(d + 48) + ans;
}
```

```
while (ans.size() > 1 && ans[0] == '0') {
        ans.erase(0, 1);
    }
return ans;
}

constint MAXN = 50 + 10;
constint MAXM = 5;
BigInteger dp[MAXN][MAXN][1 << MAXM];

bool Check(intmask) { return (mask& -mask) == mask; }

bool Compare(BigIntegera, BigIntegerb) {
if (a.size() != b.size()) {
returna.size() <b.size();
    }
returna<= b;
}

int main() {
    freopen("QPALIN.inp", "r", stdin);
    freopen("QPALIN.out", "w", stdout);

int n, m, k;
    string S;
BigInteger t;

    cin >> n >> k;
cin >> S;
    cin >> t;

    sort(S.begin(), S.end());
    S.resize(unique(S.begin(), S.end()) - S.begin());
    m = (int)S.size();

for (int i = 0; i <= n; ++i) {
for (int j = 0; j <= n; ++j) {
for (int x = 0; x < 1 << m; ++x) {
                dp[i][j][x] = "0";
            }
        }
    }
for (int x = 0; x < 1 << m; ++x) {
if (Check(x) == false) {
continue;
        }
for (int j = k + 2; j <= n; ++j) {
            dp[0][j][x] = "1";
        }
```

```
    }
for (int i = 1; i <= n; ++i) {
for (int j = 0; j <= n; ++j) {
for (int x = 0; x < 1 << m; ++x) {
                dp[i][j][x] = "0";
for (int d = 0; d < m; ++d) {
int nx = x ^ (1 << d);
int nj = j + Check(nx);
                    dp[i][j][x] = Sum(dp[i][j][x], dp[i - 1][nj][nx]);
                }
            }
        }
    }

int mask = 0;
int cnt = 0;
for (int i = 1; i <= n; ++i) {
for (int d = 0; d < m; ++d) {
int tmask = mask ^ (1 << d);
int tcnt = cnt + Check(tmask);
BigInteger s = dp[n - i][tcnt][tmask];
if (Compare(t, s)) {
                putchar(S[d]);
                mask = tmask;
                cnt = tcnt;
break;
            } else {
                t = Subtract(t, s);
            }
        }
    }
    putchar('\n');
return 0;
}
```

5.4 Test:

Đường dẫn:

https://drive.google.com/file/d/1duP6F4bPx7--i87u_oInM4SZ3_xTiYqu/view?usp=sharing

5.5 Cảm nhận:

Nhìn chung, nếu đã biết cách kiểm tra nhanh xâu palindrome cấp 0 thì đây chỉ là bài toán mức trung bình với phần cài đặt đơn giản. Tuy rằng giới hạn kết quả của bài toán yêu cầu thí sinh phải cài đặt số nguyên lớn, độ phức tạp của thuật toán này đủ nhỏ để cách cài số lớn "trâu" (sử dụng string để mô tả số nguyên) vượt được TL.

Độ phức tạp: gọi M là số lượng ký tự phân biệt trong xâu đề bài cho.

- O(N^2 M 2^M) cho phần khởi tạo.
- O(NM) cho phần tính kết quả.

(Ở đây vẫn chưa tính độ phức tạp của các phép tính trên số nguyên lớn).

Bài toán 6: Ngài đáng kính

6.1 Đề bài:

Nguồn:VO17BACH (Đề VNOI Online 2017, phiên bản 1 test ở group Vietnam Competitive Programming trên Codeforces).

Để hack não đội tuyển TPC (Turbo Pascal chuyên), hôm nay ngài Xuân Bách Ferguson ra cho các học trò một bài toán sau đây.

Ngài Bách có các số nguyên dương từ 1 tới N. Ngài sắp xếp các số này theo thứ tự thỏa mãn điều kiện sau:

- Nếu tổng các chữ số của X nhỏ hơn tổng các chữ số của Y, X đứng trước Y.
- Nếu tổng các chữ số của X bằng tổng các chữ số của Y và X có thứ tự từ điển nhỏ hơn Y, X đứng trước Y.

Ví dụ:

- 227 đứng trước 97 vì tổng các chữ số của 227 nhỏ hơn tổng các chữ số của 97.
- 11 đứng trước 3 vì tổng các chữ số của 11 nhỏ hơn tổng các chữ số của 3.
- 9230 đứng sau 914 vì 914 có thứ tự từ điển nhỏ hơn 9230.
- 20 đứng trước 200 vì 20 có thứ tự từ điểm nhỏ hơn 200.

Ngài Bách yêu cầu học sinh giải hai bài tập sau:

- Tìm số đứng thứ K trong dãy đã được sắp xếp theo quy tắc trên.
- Tìm số thứ tự của số K trong dãy đã được sắp xếp theo quy tắc trên.

Dữ liệu vào: Cho trong file **VO17BACH.inp**:

- Gồm một dòng chứa hai số nguyên không âm N và K

Dữ liệu ra: In ra file **VO17BACH.out**:

- Dòng đầu tiên chứa một số nguyên là thứ tự của số K.
- Dòng thứ hai chứa một số nguyên là số đứnng ở vị trí của số K.

Giới hạn:

- T < 2, Q < 200 và N <= 1e18.

VO17BACH.inp	VO17BACH.out
10 2	3 10

6.2 Hướng dẫn giải thuật:

Ta sẽ giải bài toán tìm số thứ K trong dãy. Bài toán tìm vị trí của số K trong dãy có thể áp dụng bài toán trước kết hợp tìm kiếm nhị phân.

Ta chuẩn bị trước các mảng sau:

- dp(len, sum) = số lượng số có len chữ số và tổng chữ số là sum.
- cntSum(s) = số lượng số trong đoạn [1; N] có tổng chữ số là s. Để tính mảng này, ta sử dụng mảng dp() và QHĐ chữ số đơn giản, sẽ không trình bày thêm.

Từ mảng cntSum(), ta dễ dàng tìm ra số thứ K có tổng chữ số là bao nhiêu, gọi tổng đó là sum. Giờ ta chỉ cần tìm theo thứ tự từ điển các số có tổng chữ số là sum.

Xét biểu diễn của (N + 1) (ta xét N + 1 để chỉ cần phải xét quan hệ <, chứ không phải <=):
$a_1\ a_2\ a_3\ \ldots\ a_n$ với n là số chữ số của (N + 1).

Giả sử ta đã tìm được (p - 1) chữ số đầu tiên của kết quả. Đặt:

- flag: số nguyên mô tả thứ tự từ điển giữa (p - 1) chữ số vừa điền với (p - 1) chữ số đầu của (N + 1) - (-1) nếu nhỏ hơn, 0 nếu bằng và 1 nếu lớn hơn.
- sum: tổng các chữ số cần điền thêm.

Giờ ta sẽ thử từng chữ số d từ 0 tới 9 để điền vào vị trí p. Khi đã điền d, các số có p chữ số đầu như đã điền sẽ chia ra 2 loại:

- Loại 1: các số có ít hơn n chữ số. Để tính số lượng ta chỉ cần thử độ dài và sử dụng mảng dp() đã có để tính.
- Loại 2: các số có đúng n chữ số. Trong loại này, ta xét 3 trường hợp:
  - flag == 0 và a[p] == d: tức là tiền tố của số vừa điền trùng với tiền tố của (N + 1). Trường hợp này quy về bài toán QHĐ chữ số cơ bản.
  - flag == -1 hoặc (flag == 0 và d < a[p]): tức là tiền tố vừa điền nhỏ hơn tiền tố tương ứng của (N + 1). Vì nó đã nhỏ hơn nên số lượng của nó chính là dp(n - p, sum - d).
  - flag == 1 hoặc (flag == 0 và d > a[p]): không cần quan tâm vì chắc chắn nó không thể thỏa mãn điều kiện nằm trong đoạn [1; N].

Chú ý là nếu đặt điều kiện lặp của p là sum > 0 thì khi thoát khỏi vòng lặp, K > 1, ta phải nhân kết quả với 10^{K-1} (ví dụ như kết quả là 1, cần điền thêm 1 chữ số 0 nhưng khi đó tổng chữ số cần là sum = 0 và đã thoát khỏi vòng lặp).

6.3 Chương trình:

```
#include<bits/stdc++.h>
usingnamespace std;

typedeflonglongLL;
typedef pair<int, int>II;

constint MAXN = 20;
```

```
LL dp[MAXN][MAXN * 9];

void Init() {
    dp[0][0] = 1;
for (int len = 1; len <= 19; ++len) {
for (int sum = 0; sum <= len * 9; ++sum) {
for (int x = 0; x < 10 && x <= sum; ++x) {
                dp[len][sum] += dp[len - 1][sum - x];
            }
        }
    }
}

int Convert(inta[], LLN) {
int n = 0;
while (N) {
a[++n] = N % 10;
N /= 10;
    }
    reverse(a + 1, a + 1 + n);
return n;
}

LL KthElement(LLN, LLK) {
LL cntSum[MAXN * 9] = {0};
int a[MAXN + 10], n = Convert(a, N + 1);
for (int i = 1, s = 0; i <= n; ++i) {
for (int d = 0; d < a[i]; ++d) {
for (int s1 = 0; s1 <= (n - i) * 9; ++s1) {
                cntSum[s + d + s1] += dp[n - i][s1];
            }
        }
        s += a[i];
    }
int sum = -1;
    cntSum[0] = 0;
for (int s = 1; s <= MAXN * 9; ++s) {
if (K<= cntSum[s]) {
            sum = s;
break;
        }
K -= cntSum[s];
    }

LL ans = 0;
int flag = 0; // x[i] < N[i]
for (int p = 1; p <= n && sum > 0; ++p) {
for (int d = (p == 1); d < 10 && d <= sum; ++d) {
LL s = 0;
```

```
for (int len = p; len <= n - 1; ++len) {
                s += dp[len - p][sum - d];
            }
if (flag == 0 && d == a[p]) {
int s1 = a[p];
for (int i = p + 1; i <= n; ++i) {
for (int x = 0; x < a[i] && x + s1 <= sum; ++x) {
                        s += dp[n - i][sum - s1 - x];
                    }
                    s1 += a[i];
                }
            } elseif (flag == -1 || (flag == 0 && d < a[p])) {
                s += dp[n - p][sum - d];
            }
if (K> s) {
K -= s;
            } else {
                ans = ans * 10 + d;
                sum -= d;
if (flag == 0) {
                    flag = (d < a[p]) ? -1 : (d == a[p] ? 0 : 1);
                }
break;
            }
        }
    }
for (int i = 1; i <= K - 1; ++i) {
        ans *= 10;
    }
return ans;
}

int SumDigits(LLn) {
int s = 0;
while (n> 0) {
        s += n % 10;
n /= 10;
    }
return s;
}
string ToString(LLn) {
    stringstream is;
    is <<n;
return is.str();
}
bool Compare(LLa, LLb) {
if (SumDigits(a) != SumDigits(b)) {
return SumDigits(a) < SumDigits(b);
    }
```

```
return ToString(a) <= ToString(b);
}

LL Position(LLN, LLK) {
LL l = 1, r = N, f = -1;
while (l <= r) {
LL m = (l + r) >> 1;
if (Compare(KthElement(N, m), K)) {
            f = m;
            l = m + 1;
        } else {
            r = m - 1;
        }
    }
return f;
}

int main() {
    freopen("VO17BACH.inp", "r", stdin);
    freopen("VO17BACH.out", "w", stdout);

LL N, K;
    cin >> N >> K;

    Init();
cout << Position(N, K) <<"\n";
cout << KthElement(N, K) <<"\n";
return 0;
}
```

6.4 Test:

Đường dẫn:

https://drive.google.com/file/d/1R83_c5atLM6xYfxwWjgZ0FyjiyNcK2qG/view?usp=sharing

6.5 Cảm nhận:

Tư tưởng của bài toán này khá đơn giản: trước tiên tìm tổng chữ số rồi tìm theo thứ tự từ điển. Tuy nhiên việc cài đặt tương đối phức tạp, vì thứ tự từ điển ở đây không cố định độ dài, nên rất dễ sai một số lỗi như:

- Quên các chữ số 0 ở cuối kết quả.
- Xét thừa các số có chữ số 0 bắt đầu.

Ngoài ra, với yêu cầu tìm vị trí của số K, việc cài đặt trực tiếp là rất phức tạp. Nhìn chung, ở dạng bài này, nên tránh các cách làm mà phải quản lý quan hệ lớn nhỏ của kết quả so với nhiều hơn 1 số, vì nếu không có kinh nghiệm cài đặt thì sẽ rất dễ rối và bị sai.

Thay vào đó có thể sử dụng bài toán tìm số thứ K kết hợp tìm kiếm nhị phân để giảm chi phí cài đặt.

Độ phức tạp: O(N^2 * 10^2).

PHẦN III. KẾT LUẬN

Qua nội dung của chuyên đề trên, ta nhận thấy rằng các bài toán về cấu hình tập hợp thường có hai yêu cầu:

- Cho cấu hình S, hỏi S nằm ở vị trí nào trong tập hợp cho trước.
- Cho vị trí K, tìm cấu hình S ứng với vị trí K trong tập hợp cho trước.

Tư tưởng chính để giải yêu cầu thứ nhất là tìm số lượng cấu hình có vị trí nằm trước S trong tập hợp từ đó suy ra vị trí của S.

Tư tưởng chính để giải yêu cầu thứ hai là thu hẹp dần phạm vi của tập đi (lần lượt bỏ đi những cấu hình S có vị trí bé hơn K tạo thành một tập hợp mới đồng thời giảm chỉ số K sao cho phù hợp với tập hợp mới).

Với những lý thuyết và bài tập đã được xây dựng ở nội dung chuyên đề trên, tôi hy vọng sẽ giúp một phần nhỏ vào công tác bồi dưỡng học sinh giỏi.

Do trình độ chuyên môn và kinh nghiệm bồi dưỡng còn nhiều hạn chế nên chắc chắn chuyên đề sẽ còn rất nhiều thiếu sót. Rất mong nhận được ý kiến đóng góp từ các đồng nghiệp.