Contents

1 Basic

1.1 debug list

```
模板要記得 init
priority_queue 要清空
把邊界條件都加入測資
邊界條件 (過程溢位, 題目數據範圍), 會不會爆 long long
是否讀錯題目, 想不到時可以自己讀一次題目
環狀or凸包問題一定要每種都算n次
比較容易有問題的地方換人寫
注意公式有沒有推錯或抄錯
精度誤差 sqrt(大大的東西) + EPS
測試 %lld or %I64d
喇分 random_suffle 隨機演算法
用long long int記得要算MLE
```

2 DP

2.1 9 talk

```
//01
for i=1..N
 for v=V..0
 f[v]=max{f[v],f[v-c[i]]+w[i]};
//01優化
 for i=1..n
 bound=max{V-sum{w[i..n]},c[i]}
 for v=V..bound
//完全背包：每種物品有無限件
for i=1..N
 for v=0..V
 f[v]=max{f[v],f[v-cost]+weight}
//多重背包：每種物品最多n[i]件
const int N = 100, W = 100000;
int cost[N], weight[N], number[N];
int c[W + 1];

void knapsack(int n, int w)
{
    for (int i = 0; i < n; ++i)
    {
        int num = min(number[i], w / weight[i]);
        for (int k = 1; num > 0; k *= 2)
        {
            if (k > num) k = num;
            num -= k;
            for (int j = w; j >= weight[i] * k; --j)
                c[j] = max(c[j], c[j - weight[i] * k] +
                     cost[i] * k);
        }
    }
    cout << "最高的價值為" << c[w];
}
//多重背包可行性
F[0, 1 . . . V ] ← −1
F[0, 0] ← 0
for i ← 1 to N
    for j ← 0 to V
        if F[i − 1][j] ≥ 0
            F[i][j] = Mi
        else
            F[i][j] = −1
    for j ← 0 to V − Ci
        if F[i][j] > 0
            F[i][j + Ci] ← max{F[i][j + Ci], F[i][j] −
                1}
//湊零錢問題
int price[5] = {5, 2, 6, 11, 17};
int c[1000+1];

void change(int m)
{
    memset(c, 0, sizeof(c));
    c[0] = 1;

    for (int i = 0; i < 5; ++i)
        for (int j = price[i]; j <= m; ++j)
            c[j] += c[j-price[i]];

    cout << "湊得價位" << m;
    cout << "湊法總共" << c[m] << "種";
}
//找零錢問題
int price[5] = {50, 20, 10, 4, 2};

void cashier(int n)
{
    int c = 0;
    for (int i=0; i<5; ++i)
    {
        c += n / price[i];
```

```
        n %= price[i];
    }

    if (n != 0)
        cout << "找不出來";
    else
        cout << "找了" << c << "個錢幣";
}
//混合背包
for i ← 1 to N
    if 第 i 件物品属于 01 背包
        ZeroOnePack(F,Ci,Wi)
    else if 第 i 件物品属于完全背包
        CompletePack(F,Ci,Wi)
    else if 第 i 件物品属于多重背包
        MultiplePack(F,Ci,Wi,Ni)
//二維背包：：对于每件物品，具有两种不同的费用，选择这件物品必
//须同时付出这两种费用。对于每种费用都有一个可付出的最大值（背包容量）。
F[i, v, u] = max{F[i − 1, v, u], F[i − 1, v − Ci, u − Di] + Wi}
//分组的背包问题：划分为 K 组，每组中的物品互相冲突，最多选一件。
for k ← 1 to K
    for v ← V to 0
        for all item i in group k
            F[v] ← max{F[v], F[v − Ci] + Wi}
```

2.2 1033C GAME

```
#include <bits/stdc++.h>
using namespace std;
string ans="";
int n;
vector<int> vec;
int dp[100005][2]={0};
char ansNow;
bool dfs(int now,int turn){
    if(dp[now][turn])return dp[now][turn];
    if(turn){
        //Alice
        bool can=false;
        for(int i=now+vec[now];i<n;i+=vec[now]){
            if(vec[i]>vec[now])
                can=can||dfs(i,turn^1);
        }
        for(int i=now-vec[now];i>=0;i-=vec[now]){
            if(vec[i]>vec[now])
                can=can||dfs(i,turn^1);
        }
        return dp[now][turn]=can;
    }else{
        //Bob
        bool can=true;
        for(int i=now+vec[now];i<n;i+=vec[now]){
            if(vec[i]>vec[now])
                can=can&&dfs(i,turn^1);
        }
        for(int i=now-vec[now];i>=0;i-=vec[now]){
            if(vec[i]>vec[now])
                can=can&&dfs(i,turn^1);
        }
        return dp[now][turn]=can;
    }
}
int main() {
    cin>>n;
    for(int i=0;i<n;i++){
        int d;
        cin>>d;
        vec.emplace_back(d);
    }
    for(int i=0;i<n;i++){
        cout<<(dfs(i,1)?'A':'B');
    }
}
```

2.3 a 夾 b

```
//找subsequence a中間有b 幾種找法
//abbaa -> 5
//[1], [4], [5], [1,4], [1,5].
//Firstly, let's erase all symbols different from 'a'
    and 'b'. Then let's split string on blocks of
    consecutive symbols 'a'.
//Now we need to multiply all sizes of blocks increased
     by 1.
//It is an answer which also includes one empty
    subsequence, so we should just decrease it by one.
#include <bits/stdc++.h>
using namespace std;
int64_t i,k,r,z=1e9+7;
string s;
int main(){
  for(cin>>s,s+='b';i<s.size();i++){
    if(s[i]=='a')k++;
    if(s[i]=='b')r=(r*k+r+k)%z,k=0;
  }
  cout<<r;
}
```

2.4 LCS

```
//「最長共同子序列」。出現於每一個序列、而且是最長的子
    序列。可能有許多個。
//
//s1: 2 5 7 9 3 1 2
//s2: 3 5 3 2 8
//
//LCS(s1, s2) = 5 3 2
//s1: a b c d b c e e a
//s2: c a b d e f g a
//s3: d c e a
//
//LCS(s1, s2, s3) =  {c e a, d e a}
//LCS(s1, s2) =
// { max( LCS(sub1, s2), LCS(s1, sub2) ) , when e1 !=
    e2
// { LCS(sub1, sub2) + e1                , when e1 ==
    e2
const int n1 = 7, n2 = 5;
// 為了實作方便，從陣列的第1格開始存入序列。
int s1[7+1] = {0, 2, 5, 7, 9, 3, 1, 2};
int s2[5+1] = {0, 3, 5, 3, 2, 8};
int length[7+1][5+1];   // DP表格

void LCS()
{
    // 初始化：當s1或s2是空集合，則LCS是空集合。
    // length[x][0] = length[0][x] = 0;
    for (int i=0; i<=n1; i++) length[i][0] = 0;
    for (int j=0; j<=n2; j++) length[0][j] = 0;

    // 填表格：依照遞迴公式
    for (int i=1; i<=n1; i++)
        for (int j=1; j<=n2; j++)
            if (s1[i] == s2[j])
                // +1是因為e1的長度為1
                length[i][j] = length[i-1][j-1] + 1;
            else
                length[i][j] = max(length[i-1][j],
                                   length[i][j-1]);

    cout << "LCS的長度是" << length[n1][n2];
}
```

2.5 Longest Non-Increasing Subsequence

```
// UVA 231: Testing the CATCHER
// 題目：有一个导弹拦截系统，当它拦截一个导弹后，下次拦截就不能超过这个高度，问拦截的最多导弹数。
// 大意: 题目大意：输出最长递减子序列长度，LIS简单变形。


//  【解題想法】DP: Longest Non-Increasing Subsequence (LDS) 非嚴格遞減
#include <bits/stdc++.h>
using namespace std;

vector <int> v;
int Case, n;

int main(){
    while (1){
        cin >> n;
        if (n == -1) break;
        v.clear();
        v.push_back(n);
        while (1){
            cin >> n;
            if (n == -1) break;
            if (n <= v[v.size()-1]) v.push_back(n);
            else {
                for (int i = 0; i < v.size(); i++){
                    if (n > v[i]){
                        v[i] = n;
                        break;
                    }
                }
            }
        }
        if (Case) cout << "\n";
        Case++;
        cout << "Test #" << Case << ":\n";
        cout << "  maximum possible interceptions: " <<
             v.size() << "\n";
    }
}
```

2.6 MATRIX 壓縮

```
//11100111
//11100111
//11100111
//00000000
//00000000
//11100111
//11100111
//11100111
//可不可以壓縮
//x-壓縮 => A[i][j]=B[□i/x□][□j/x□]
//題解 行列連續1 or 0的gcd = ans
#include <bits/stdc++.h>
using namespace std;
const int N = 5200;
int n;
bool a[N][N];
void parse_char(int x, int y, char c) {
  int num = -1;
  if (isdigit(c)) {
    num = c - '0';
  } else {
    num = c - 'A' + 10;
  }
  for (int i = 0; i < 4; ++i) {
    a[x][y + 3 - i] = num & 1;
    num >>= 1;
  }
}
int main() {
  scanf("%d", &n);
  char buf[N];
  for (int i = 0; i < n; ++i) {
    scanf("%s", buf);
    for (int j = 0; j < n / 4; ++j) {
      parse_char(i, j * 4, buf[j]);
    }
  }
  int g = n;
  for (int i = 0; i < n; ++i) {
    for (int j = 0; j < n; ++j) {
      int k = j;
      while (k < n && a[i][k] == a[i][j]) ++k;
      g = __gcd(g, k - j);
      j = k - 1;
    }
  }
  for (int j = 0; j < n; ++j) {
    for (int i = 0; i < n; ++i) {
      int k = i;
      while (k < n && a[k][j] == a[i][j]) ++k;
      g = __gcd(g, k - i);
      i = k - 1;
    }
  }
  cout << g << endl;
}
```

2.7 Stacking Boxes

```
//題目
//在數學或電腦科學裡，有些概念在一維或二維時還蠻簡單的，但到 N 維就會顯得非常複雜。試想一個 n 維的「盒子」：在二維空間裡
//，盒子 ( 2 , 3 ) 可代表一個長為 2 個單位，寬為 3 個單位的盒子；在三維空間裡，盒子 ( 4 , 8 , 9 ) 則是一個 4*8*9
//（長、寬、高）的盒子。至於在六維空間裡，也許我們不清楚 ( 4 , 5 , 6 , 7 , 8 , 9 ) 長得怎樣，不過我們還是可以分析這
//些盒子的特性。在此問題裡，我們要算出一組 n 維盒子裡，它們的「最長套入串列」： b1,b2,.....,bk，其中每個盒子 bi 都
//可以「放入」盒子 bi+1 中（1≤i<k）考慮兩個盒子 D = ( d1,d2,.....,dn ) ， E = ( e1,e2,.....,en ) 。如果盒子
//D 的 n 個維，能夠存在一種重排，使得重排後， D 每一維的量度都比 E 中相對應的維的量度還要小，則我們說盒子 D 能「放入」
//盒子 E 。（用比較不嚴謹的講法，這就好像我們將盒子 D 翻來翻去，看看能不能擺到 E 裡面去。不過因為我們考慮的是任一重排，
//所以實際上盒子不只可轉來轉去，甚至還可以扭曲。）（還是看看下面的例子說明好了）。譬如說，盒子 D = ( 2 , 6 ) 能夠被放
//入盒子 E = ( 7 , 3 ) 裡，因為 D 可以重排變為 ( 6 , 2 ) ，這樣子 D 的每個維的量度都比 E 裡對應的維還要小。而盒子
//D = ( 9 , 5 , 7 , 3 ) 就沒辦法放進盒子 E = ( 2 , 10 , 6 , 8 ) ，因為就算再怎摸重排 D 裡的維，還是沒辦法符合
//「放入」的條件。不過 F = ( 9 , 5 , 7 , 1 ) 就可以放入 E 了，因為 F 可以重排成 ( 1 , 9 , 5 , 7 ) ，這樣就符合
//了放入的條件。我們今定義「放入」如下：對於任兩個盒子 D = ( d1,d2,.....,dn) 和 E = ( e1,e2,.....,en ) ，
//如果存在一種 1..n 的重排π，使得對於任何的 1≤i≤n，皆有 dπ(i)<ei，則我們說盒子 D 能「放入」盒子 E
//
```

```
//Input Format
//輸入包含多組測試資料。每組測試資料的第一列有兩個數字：第一個是盒子的數量 k ，然後是盒子的維數 n 。
//接下來有 k 列，每列有n個整數表示一個盒子的 n 個維的量度，量度之間由一個以上的空白做區隔。第一列表示第一個盒子，第二列表示第二個盒子，依此類推。
//此問題裡，盒子的維數最小是 1 ，最大是 10 ， 並且每組測試資料中盒子的個數最多為 30 個。
//
//Output Format
//對於每一組測試資料，你必須輸出兩列數字：第一列是「最長套入串列」的長度，第二列是按照內外順序，印出「最長套入串列」
//裡盒子的編號（其中編號是按照在輸入檔案的每組數列裡所出現的順序，例如第一個盒子就是 1 號 . . . 等等。）
//最裡面的盒子
//（或是最小的）擺在第一個，再來是次小的，依此類推。
//如果對於每一組的盒子，存在兩個以上的「最長套入串列」，輸出任何一個均可。
//Solution
//這題其實就是經典的LIS變形題目，首先我們要對每一個箱子的維度進行排序，可以用簡單邏輯得知由小到大排是最佳方式。
//接著我們再對各箱子進行排序，排序是利用字典大小的方法排，這樣就解決了！

#include <bits/stdc++.h>
using namespace std;

int k, n;
struct objects {
    int length[12];
    int num;
    bool operator < (objects const &b) const {
        for (int i = 0; i < n; i++)
            if (!(length[i] < b.length[i]))
                return 0;
        return 1;
    }
}input[35];
int dp[35];
int previous[31];
int ans;
void trace(int i) {
    if (previous[i] != -1) trace(previous[i]);
    cout << input[i].num;
    if (--dp[ans])
        cout << ' ';
    else
        cout << '\n';
}
bool cmp(objects a, objects b) {
    for (int i = 0; i < n; i++) {
        if (a.length[i] < b.length[i])
            return 1;
        if (a.length[i] > b.length[i])
            return 0;
    }
    return 0;
}

int main() {
    while (cin >> k >> n) {
        for (int i = 0; i < k; i++) {
            for (int j = 0; j < n; j++)
                cin >> input[i].length[j];
            input[i].num = i + 1;
            sort(input[i].length, input[i].length + n);
        }
        for (int i = 0; i < k; i++) dp[i] = 1;
        sort(input, input + k, cmp);
        for (int i = 0; i < k; i++)
            previous[i] = -1;
        for (int i = 0; i < k; i++)
            for (int j = i + 1; j < k; j++)
                if (input[i] < input[j] && dp[i] + 1 > dp[j]) {
                    dp[j] = dp[i] + 1;
                    previous[j] = i;
                }
        ans = 0;
        for (int i = 1; i < k; i++)
            if (dp[ans] < dp[i])
                ans = i;
        cout << dp[ans] << '\n';
        trace(ans);
    }
    return 0;
}
```

2.8 HistoryGrading

```
// 题目：历史上有一些事件发生的先后顺序，现在有很多学生写了不同的顺序表，

//            判断每个学生的最大的前后顺序正确的序列。

// 分析：dp，LIS，最大上升子序列。

//            注意本题的数据格式，串里的每个元素对应于：对应下标编号的事件在表中的位置；

//            状态：F（n）记录以第n个元素为结束元素的序列的最长上升子序列，有转移方程：

//             F（n）= max（F（i）+1）  { 其中 0 < i < n 且 data[i] < data[n] }
#include <iostream>
#include <cstdlib>

using namespace std;

int A[22],B[22],L[22];

int main()
{
  int n,t;
  cin >> n;
  for (int i = 1 ; i <= n ; ++ i)
    cin >> A[i];
  while (cin >>t) {
    B[t] = 1;
    for (int i = 2 ; i <= n ; ++ i) {
      cin >> t;
      B[t] = i;
    }

    for (int i = 1 ; i <= n ; ++ i) {
      L[i] = 1;
      for (int j = 1 ; j < i ; ++ j)
        if (A[B[j]] < A[B[i]] && L[j] >= L[i])
          L[i] = L[j]+1;
    }

    int max = 0;
    for (int i = 1 ; i <= n ; ++ i)
      if (max < L[i])
        max = L[i];

    cout << max << endl;
  }
  return 0;
}
```

2.9 Compromise

```
// 题目大意：输入两段文章，每段文章以#字符为终止符，输出最长公共子序列（单词组成）
```

```
// 解題策略：DP+LCS输出，算法网上比比皆是。注意输入和单词分离，输出时注意当输出最后一个单词，须输出换行

#include <iostream>
#include <sstream>
#include <cstdio>
#include <cstring>
#include <string>
using namespace std;
const int maxn = 1010;
string buf1[3030], buf2[3030];
string s1[3030], s2[3030];
int dp[1010][1010], mark[1010][1010], wordnum_buf1,
    wordnum_buf2;
int tc = 0;

//定义字符流，分离单词
int input(string s[], string buf[], int ff){
    int i=0, k=0;
    while(ff)
    {
       getline(cin, s[++i]);
       if((s[i])[0] == '#'){
           k=0;
           for(int j=1; j<i; j++){
              stringstream ss(s[j]);
              string temp;
              while(ss >> temp)
              {

                 buf[++k] = temp;
              }
           }
           ff = 0;
       }
    }
    return k;
}

void LCS(){
   memset(dp, 0, sizeof(dp));
   memset(mark, 0, sizeof(mark));
   for(int i=1; i<=wordnum_buf1; i++){
      for(int j=1; j<=wordnum_buf2; j++){
         if(buf1[i] == buf2[j]){
            dp[i][j] = dp[i-1][j-1]+1;
            mark[i][j] = 1;
         }
         else if(dp[i-1][j] >=  dp[i][j-1]){
             dp[i][j] = dp[i-1][j];
             mark[i][j] = 2;
         }
         else{
            dp[i][j] = dp[i][j-1];
            mark[i][j] = 3;
         }
      }
   }
}

void print(int i, int j){
    if(i == 0 || j == 0)   return;
    if(mark[i][j] == 1){
       print(i-1, j-1);
       //定义全局变量，控制输出
       if(tc == 0)  cout << buf1[i];
       else cout << " " << buf1[i];
       tc++;
    }
    else if(mark[i][j] == 2)   print(i-1, j);
    else if(mark[i][j] == 3)   print(i, j-1);
}

bool read(){
    //返回文章单词数量
    wordnum_buf1 = input(s1, buf1, 1);
    wordnum_buf2 = input(s2, buf2, 1);
    return !cin.eof();
}
////////////////////////////////////////
int main(){
    while(read()){
       LCS();
       tc = 0;
       print(wordnum_buf1, wordnum_buf2);
       cout << endl;
    }
    return 0;
}
```

2.10 TheTwinTowers

```
// input
// 牆磚: 1 ≤ N1, N2 ≤ 100
// N1 個牆磚半徑
// N2 個牆磚半徑

// 題解
// 在一個古帝國，有兩座形狀不一樣的塔，它們是由不同半徑的圓形牆磚疊合而成。
// 數千年後，皇帝要求工匠移除某些牆磚，使得兩座塔變得一樣，當然牆磚的順序必須和原本的塔一樣
// 請問若要使塔的高度最高，它有幾塊牆磚?

// 作法
// 要使塔最高，就代表相同的牆磚要愈多愈好，而且順序不能變
// 那就是 LCS 解了！
// 輸出記得空一行


#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define max(a,b) ((a)>(b)?(a):(b))

int N1[105],N2[105];
int F[105][105];

int main()
{
  int n1,n2,Case = 1;
  while ( scanf("%d%d",&n1,&n2) && n1+n2 ) {
    for ( int i = 1 ; i <= n1 ; ++ i )
      scanf("%d",&N1[i]);
    for ( int i = 1 ; i <= n2 ; ++ i )
      scanf("%d",&N2[i]);

    memset( F, 0, sizeof(F) );
    for ( int i = 1 ; i <= n1 ; ++ i )
    for ( int j = 1 ; j <= n2 ; ++ j )
      if ( N1[i] == N2[j] )
        F[i][j] = F[i-1][j-1]+1;
      else F[i][j] = max( F[i-1][j], F[i][j-1] );

    printf("Twin Towers #%d\n",Case ++);
    printf("Number of Tiles : %d\n\n",F[n1][n2]);
  }
  return 0;
}
```

2.11 LongestMatch

```
// 给定两行字符串序列，输出它们之间最大公共子单词的个数
```

```
// 对于给的两个序列X 和 Y，用i 和 j分别作为它们的前缀指针，f[i][j]表示序列X的前缀Xi 和 序列Y的前缀Yi 的最长公共子序列的长度，在这道题中，可把每一个单词当作一个字符来进行比较。

// 当 i || j 为0时 ，此 f[i][j] = 0;

// 当 i!=0 && j!=0 && Xi==Yi 时 f[i][j] = f[i-1][j-1] + 1;

// 当 i!=0 && j!=0 && Xi!=Yi 时 f[i][j] = max ( f[i-1][j] + f[i][j-1] );

#include <iostream>
#include <cstdio>
#include <cstring>
#include <string>
using namespace std;

struct node
{
    int num;          //记录单词的个数
    string w[1010];   //单词序列
};
int f[1010][1010];    //Xi 中前i个单词 与 Yj 中前j个单词 最大公共子序列个数为f[i][j]
void devide(string s,node &t)
{
    int len=s.size();
    t.num=1;
    for(int i=0; i<len; i++)
    {
        if( (s[i]>='a' && s[i]<='z') || (s[i]>='A' && s[i]<='Z') ||(s[i]>='0' && s[i]<='9') )
        {
            t.w[t.num]+=s[i];
        }
        else t.num++;
    }
    int n=0;
    for(int i=1; i<=t.num ; i++)
    {
        if(!t.w[i].empty())
            t.w[++n]=t.w[i];
    }
    t.num=n;
}
int main()
{
    int ca=1;
    while(!cin.eof())
    {
        string X,Y;
        node t1,t2;
        getline(cin,X);
        devide(X,t1);      //把x中的单词取出来
        getline(cin,Y);
        devide(Y,t2);       //同上
        printf("%2d. ",ca++);
        if(X.empty() || Y.empty())
        {
            printf("Blank!\n");
            continue;
        }
        memset(f,0,sizeof(f));
        for(int i=1; i<=t1.num; i++)
            for(int j=1; j<=t2.num; j++)
            {                                                //
                计算前i个单词 和 前j个单词最大匹配数
                f[i][j]=max(f[i-1][j],f[i][j-1]);
                if(t1.w[i] == t2.w[j])
                    f[i][j]=max(f[i][j] , f[i-1][j
                        -1]+1);
            }
        printf("Length of longest match: %d\n",f[t1.num
            ][t2.num]);
    }
    return 0;
}
```

2.12 CommonPermutation

```
// 題意:
// 此題輸出入都是小寫字母

// 每兩行為一組測資，分別代表兩序列 (A 和 B)
// 輸出共同存在 A 和 B的字母
// 如果某字母出現不只一次，就要印出 A 和 B 之間較少的次數

// 如: A = abccc , B = accd，則輸出 acc

// 以字母 a ~ z 的順序印出

// 解法:
// 以兩陣列分別統計兩序列所有元素出現次數
// 輸出時，從 a 開始，依序印出兩陣列次數大於0且較小次數的字母

// 其中，小寫 a~z 的 ASCII 為從 97~122

#include <iostream>
#include <string>
#include <algorithm>
using namespace std;

int main()
{
    string a,b;//輸入用字串
    int i,t;//方便用

    while(getline(cin, a))//輸入資料
    {
        getline(cin, b);//輸入資料

        string re[1000]={""};//記錄用陣列

        for(i=0;i<a.length();i++)//使用雙重迴圈比對
        {
            for(t=0;t<b.length();t++)
            {
                if(a[i]==b[t])//找到重複的字
                {
                    re[i]=a[i];//紀錄
                    b.erase(t,1);//擦掉該字避免重複找查
                    break;//直接結束第二個迴圈， 理由同上
                }
            }
        }

        sort(re,re+1000);//排序

        for(i=0;i<1000;i++)//輸出
        {
            cout<<re[i];
        }

        cout<<endl;//記得換行
    }
    return 0;

}
```

2.13 XOR 特殊條件 + 前綴和

```
//題解：這個題只要知道異或滿足
//
//    if A ^ B == C then A == B ^ C 這個性質
//
//就會很簡單，首先處理出異或前綴和是很自然的，之后對於滿足條件的pair，必定有f(r) == f(l - 1)，
//f即異或前綴和，這是必要條件，同時也是充分的，充分性同樣利用這個性質
//
//    若有a[l] ^ a[l+1] ^ ... ^ a[k] = a[k+1] ^ a[k + 2] ^ ... ^ a[r]， 那么就可以利用上述性質使得k == mid，因為
//如果k > mid，那么,等式兩邊同時異或a[k]即可將a[k]從等式左邊變到右邊，繼續進行這種操作知道k == mid，對於k < mid，同理，
//因此現在就是找滿足異或前綴和相等的pair，並且要使得r - l + 1是偶數，也就是r 和 l - 1同奇偶，這個問題很好解決，統計每種異
//或前綴和的個數（對每個值按下標奇偶性分類）即可計算出最終結果，由a數組數據的范圍可知異或前綴和 < 2^21，復雜度是完全可以接受的
//，別忘了開 long long。
#include <bits/stdc++.h>

using namespace std;

#define REP(i, n) for (int i = 1; i <= (n); i++)
#define sqr(x) ((x) * (x))

const int maxn = 2000000 + 100;
const int maxm = 200000 + 100;
const int maxs = 10000 + 10;

typedef long long LL;
typedef pair<int, int> pii;
typedef pair<double, double> pdd;

const LL unit = 1LL;
const int INF = 0x3f3f3f3f;
const double eps = 1e-14;
const double inf = 1e15;
const double pi = acos(-1.0);
const int SIZE = 100 + 5;
const LL MOD = 1000000007;

LL cnt[maxn][2];
int n;

int main()
{
    ios::sync_with_stdio(false);
    cin.tie(0);
    //freopen("input.txt", "r", stdin);
    //freopen("output.txt", "w", stdout);
    cin >> n;
    int pre = 0, x;
    for (int i = 1; i <= n; i++)
    {
        cin >> x;
        pre ^= x;
        //cout << pre << endl;
        cnt[pre][i % 2]++;
    }
    LL ans = 0;
    cnt[0][0]++;
    for (int i = 0; i < maxn; i++)
    {
        ans += cnt[i][0] * (cnt[i][0] - 1) / 2;
        ans += cnt[i][1] * (cnt[i][1] - 1) / 2;
    }
    cout << ans << endl;
    return 0;
}
```

2.14 倒水問題

```
//有三個桶子，它們各能裝 a ， b ， 和 c 公升。（a , b , c 都為正整數而且不超過200。）第一，第二個桶子最剛開始是空的，
//但是第三個桶子卻是裝滿水的。你可以從 X 桶子把水倒入 Y 桶子裡並且把 Y 桶子裝滿，要不然就是把 X 桶子倒乾。倒水的步驟可以執行很多次。
//
//你要寫一個程式去計算整個過程中至少要倒多少水才能達成目標，即這三個桶子中有一個桶子剩 d 公升的水。（d 為正整數而且不超過200。）
//但是如果你沒有辦法達成目標，也就是沒有辦法讓任何一個桶子剩下 d 公升的水，那麼請達成 d’ 公升，d’ 比 d 小但是最接近 d。當 d’ 找到了，
//請你輸出整個過程至少要倒多少水才能達成 d’ 公升。
//BFS
#include<stdio.h>
#include<queue>
#include<iostream>
#include<string.h>
#include<stdlib.h>
#include<algorithm>
using namespace std;
struct state {
  int a[3], cost;
  bool operator < (const state& a) const {
    return this -> cost > a.cost;
  }
};
state temp;
int jugs[3], d, vis[205][205], ans[205], head, tail;
void update(state s) {
  for (int i = 0; i < 3; i++) {
    int t2 = s.a[i];
    if (ans[t2] < 0 || s.cost < ans[t2]) { //记录到达t2状态(出现装有t2水量的水杯)所需的最小水量
      ans[t2] = s.cost;
    }
  }
}
void BFS() {
  priority_queue<state> q;
  state start;
  start.cost = 0;
  start.a[0] = 0;
  start.a[1] = 0;
  start.a[2] = jugs[2];
  q.push(start);
  vis[0][0] = 1;

  while(!q.empty()) {
    temp = q.top();
    q.pop();
    update(temp);  //更新记录
    if(ans[d] >= 0) break;

    for(int i = 0; i < 3; i++) {
      if (temp.a[i] == 0) continue;

      for(int j = 0; j < 3; j++) {
        if (i == j) continue;
        if (temp.a[j] == jugs[j]) continue;

        int m = min(temp.a[i], jugs[j] - temp.a[j]);
        state temp2 = temp;
        temp2.cost += m;
        temp2.a[i] -= m;
        temp2.a[j] += m;

        if(!vis[temp2.a[0]][temp2.a[1]]) {
          vis[temp2.a[0]][temp2.a[1]] = 1;
          q.push(temp2);
        }
      }
```

```
    }
  }

}
int main() {
  int T;
  scanf("%d", &T);
  while (T--) {
    scanf("%d %d %d %d", &jugs[0], &jugs[1], &jugs[2],
        &d);
    memset(vis, 0, sizeof(vis));
    memset(ans, -1, sizeof(ans));
    BFS();
    while (d >= 0) {
      if (ans[d] >= 0) { //若不能到达目标状态，就寻找最
          接近并且可达状态
        printf("%d %d\n", ans[d], d);
        break;
      }
      d--;
    }
  }
}
```

2.15 分 k 組

```
//題意
//
//給你n個數，分成k組，要求每組內最大值與最小值的差值不
    超過5。求k組最多可以放多少個數。
//
//1 ≤ n , k ≤ 50001 \leq n,k \leq 50001≤n ,k≤5 0 0 0
//做法
//
//首先對數組排序，我們可以預處理每個數最多可以向左擴展
    的長度。
//之後我們用dp [ i ] [ j ]dp[i][j]d p [ i ] [ j ]表示前
    i個數分為j組最多可以放多少個數。
//對於每個i，一定是從之前預處理的位置轉移過來，因為一段
    應該放盡量多的值。
//我們設位置iii能夠擴展的最左位置為l[i]l[i]l [ i ]。那
    麼轉移方程為:d
//p [ i ] [ j ] = m a x ( dp [ i − 1 ] [ j ] , dp [ i ]
     [ j − 1 ] , dp [ l [ i ] − 1 ] [ j − 1 ] + i − l [
     i ] + 1 )dp[i][j]=max(dp[i-1][j],dp[i][j-1],dp[l[i
    ]-1][j-1]+il[i]+ 1)d p [ i ] [ j ]=m a x ( d p [ i
    −1 ] [ j ] ,d p [ i ] [ j−1 ] ,d p [ l [ i ]−1 ] [
    j−1 ]+i−l [ i ]+1 )
//最後dp [ n ] [ k ]dp[n][k]d p [ n ] [ k ]便是答案。
#include<stdio.h>
#include<iostream>
#include<algorithm>
using namespace std;
const int maxn = 5005;
int a[maxn],l[maxn];
int dp[maxn][maxn];
int main()
{
    int n,k;
    scanf("%d%d",&n,&k);
    for(int i=1;i<=n;i++) scanf("%d",&a[i]);
    sort(a+1,a+1+n);
    dp[0][0]=0;
    for(int i=1;i<=n;i++) l[i]=lower_bound(a+1,a+1+n,a[
        i]-5)-a;
    for(int i=1;i<=n;i++)
    {
        for(int j=1;j<=i;j++)
        {
            dp[i][j]=max(dp[i-1][j],dp[i][j-1]);
            dp[i][j]=max(dp[i][j],dp[l[i]-1][j-1]+i-l[i
                ]+1);
        }
    }
    printf("%d\n",dp[n][k]);
    return 0;
}
```

2.16 印出最後出現的 LIS 即可得解

```
/*****************************************************
    */
/* UVa 481 What Goes Up
    */
/* Author: Maplewing [at] knightzone.studio
    */
// 經典的 LIS (Longest Increasing Subsequence) ，印出最
    後出現的LIS即可得解。
/*****************************************************
    */
#include <iostream>
#include <cstdio>
#include <algorithm>
#include <vector>
#include <stack>
using namespace std;

int getLIS(const vector<int> &sequence, vector<int> &
    position){
  if(sequence.size() == 0) return 0;

  vector<int> subsequence;
  for(int i = 0 ; i < sequence.size() ; ++i ){
    vector<int>::iterator lowerBound = lower_bound(
        subsequence.begin(), subsequence.end(),
        sequence[i]);
    if(lowerBound == subsequence.end()){
      position[i] = subsequence.size();
      subsequence.push_back(sequence[i]);
    }
    else{
      position[i] = lowerBound - subsequence.begin();
      *lowerBound = sequence[i];
    }
  }

  return subsequence.size();
}


int main(){
  vector<int> sequence;
  int n;
  while(scanf("%d", &n) != EOF){
    sequence.push_back(n);
  }

  vector<int> position(sequence.size(), 0);
  int length = getLIS(sequence, position);
  printf("%d\n", length);
  printf("-\n");

  stack<int> increasingSubsequence;
  int currentPosition = length - 1;
  for(int i = sequence.size() - 1 ; i >= 0 ; --i){
    if(currentPosition == position[i]){
      increasingSubsequence.push(sequence[i]);
      --currentPosition;
    }
  }

  while(!increasingSubsequence.empty()){
    printf("%d\n", increasingSubsequence.top());
    increasingSubsequence.pop();
  }

  return 0;
}
```

2.17 奇數回文

```
//大致題意：給你一個序列，數字有-1或者1到k構成。其中的-1需要用1到k中任意一個數字替代，現在定義一個壞序列的標準是把-1替代完畢後，
//沒有長度大於1的奇回文串。問你在所有的替代方案中，最後能得到不是壞序列的方案數是多少。
//
//這裡的一個奇回文串，看似很複雜不知道應該怎麼做，但是其實很容易發現，所有的奇回文串必定包含一個長度為3的回文串。換句話說，
//含有長度為3的回文串的就是壞序列。再進一步，發現長度為3的回文串，第一個字符和第三個字符是一樣的。也就意味著，如果我們把原串
//按照奇偶取拆成兩個串的話，只要其中有一個串有連續兩個相同的字符，那麼這個串就是一個壞序列，反之就是好序列。
//
//繼續分析，最後的答案就是奇數串好序列個數*偶數串好序列個數。那麼，現在問題的關鍵就是怎麼求這個好序列個數。我們考慮dp[i][0]表示
//長度為i的一串-1且兩端夾著兩個不相等的數字的好序列方案數。同理，dp[i][1]表示長度為i的一串-1且兩端夾著兩個相等的數字的好序列的
//方案數。那麼，我們可以很簡單的得到轉移方程：
//
//
//dp[i][0] = dp[i-1][0] + (k-2)*d[i-1][1]
//dp[i][1] = (k-1)*d[i-1][0]
//dp[0][0] = 0, dp[1][0] = 1
//
//根據這個轉移方程，我們就可以把序列按照已知的數字進行分段，把每一段的方案數乘起來就是我們最後的答案。最後要特殊處理
//一下兩端為-1的情況。具體見代碼：
#include <bits/stdc++.h>
#define INF 0x3f3f3f3f
#define LL long long
#define sc(x) scanf("%d",&x)
#define scc(x,y) scanf("%d%d",&x,&y)
#define sccc(x,y,z) scanf("%d%d%d",&x,&y,&z)
#define file(x) freopen(#x".in","r",stdin);

using namespace std;
const int N = 2e5 + 10;
const int mod = 998244353;

int a[N],b[N];
int n,k,l1,l2;

LL solve(int *a,int len)
{
    if (len==1)
        if (a[1]==-1) return k;
                else return 1;
    LL sam=1,dif=0,res=1;
    int ls=a[1];
    for(int i=2;i<=len;i++)
    {
        LL nxt_sam=dif*(k-1)%mod;
        LL nxt_dif=(sam+(k-2)*dif%mod)%mod;
        if (a[i]==-1) sam=nxt_sam,dif=nxt_dif;
        else
        {
            if (ls==-1) res=res*(nxt_sam+nxt_dif*(k-1)%mod)%mod;
            else res=res*(a[i]==ls?nxt_sam:nxt_dif)%mod;
            ls=a[i]; sam=1; dif=0;
        }
    }
    res=res*(sam+dif*(k-1)%mod)%mod;
    if (ls==-1) return res*k%mod;
    return res;
}

int main()
{
    scc(n,k);
    for(int i=1;i<=n;i++)
    {
        int x; sc(x);
        if (i&1) a[++l1]=x;
            else b[++l2]=x;
    }
    printf("%lld\n",solve(a,l1)*solve(b,l2)%mod);
}
```

2.18 字串前後放區間 dp

```
//題目大意：
//給定一個字符串S和T。有兩種操作：
//1.將S的最前面的字符移到A字符串的最前面，然後將S中的這個最前面的字符刪掉。
//2.將S的最前面的字符移到A字符串的最後面，然後將S中的這個最前面的字符刪掉。
//用不超過n個操作，問有多少個操作序列能夠使得生成的字符串的前綴為T。
//
//解題思路：
//區間dp。首先將T想像成S一樣長的字符，其中長度大於T的部分就是對任何字符都可以匹配。
//dp[i][j]表示T中[i…j]被匹配的方案數。
//然後最外層枚舉區間長度len，內層枚舉i。
//將S的第len個字符插入T的最前面，則將S[len]和T[i]進行比較，如果相同則dp[i][i+len-1]+ = dp[i+1][ i+len-1]。
//將S的第len個字符插入T的最後面，則將S[len]和T[i+len-1]進行比較，如果相同則dp[i][i+len-1]+=dp[i ][i+len-2]。
//因為操作不超過n次，但為了有T的前綴，最少要操作m次，則將dp[1][m…n]的答案都加起來，最後*2就是答案了。
#import<bits/stdc++.h>
#define ll long long
#define mod 998244353
using namespace std;
string s,t;
ll dp[3002][3002],n,m,ans=0,i,l,r;
int main(){
    cin>>s>>t;
    n=s.size();m=t.size();
    s="*"+s;t="*"+t;
    for(i=1;i<=n+1;++i)dp[i][i-1]=1;
    for(i=1;i<=n;++i)
    for(l=1,r=l+i-1;r<=n;++l,++r){
        if(l>m||s[i]==t[l])dp[l][r]+=dp[l+1][r]%mod;
        if(r>m||s[i]==t[r])dp[l][r]+=dp[l][r-1]%mod;
    }
    for(i=m;i<=n;++i)
    ans=(ans+dp[1][i])%mod;
    cout<<ans;
}
```

2.19 序列平均分割

```
//題目大意：将一个个数为n的序列分割成m份，要求这m份中的每份中值（该份中的元素和）最大值最小， 输出切割方式，
//有多种情况输出使得越前面越小的情况。
//
//解题思路：二分法求解f(x), f(x) < 0 说明不能满足， f(x) >= 0说明可以满足，f(x) 就是当前最大值为x的情况最少需要
//划分多少份-要求份数（如果f(x ) >= 0 说明符合要求而且还过于满足，即x还可以更小）。
```

```
//
//注意用long long .
#include <stdio.h>
#include <string.h>
int max(const int &a, const int &b) { return a > b ? a
    : b;}

const int N = 1005;
int n, T;
long long num[N], sum[N], rec[N];

bool judge(int Max) {
    int cnt = 0, first = 0, end = 1;
    while (end <= n) {
  if (sum[end] - sum[first] > Max) {
      cnt++;
      first = end - 1;
  }
  end++;
    }
    return cnt <= T - 1;
}

int main () {
    long long  cas, lift, right, cur;
    scanf("%lld", &cas);
    while (cas--) {
  // Init;
  memset(num, 0, sizeof(num));
  memset(sum, 0, sizeof(sum));
  memset(rec, 0, sizeof(rec));
  lift = right = 0;

  // Read.
  scanf("%d%d", &n, &T);
  for (int i = 1; i <= n; i++) {
      scanf("%lld", &num[i]);
      sum[i] = sum[i - 1] + num[i];
      lift = max(lift, num[i]);
  }
  right = sum[n];

  // Count;
  while (lift != right) {
      cur = (right + lift) / 2;
      if (judge(cur)) {
    right = cur;
      }
      else
    lift = cur + 1;
  }

  // Draw;
  long long now = 0, cnt = 0;
  for (int i = n; i > 0; i--) {
      if (now + num[i] > lift || i < T - cnt) {
    rec[i] = 1;
    cnt++;
    now = num[i];
      }
      else
    now += num[i];
  }

  // Printf;
  for (int i = 1; i < n; i++) {
      printf("%lld ", num[i]);
      if (rec[i])     printf("/ ");
  }
  printf("%lld\n", num[n]);
    }
    return 0;
}
```

題意：有n本书，m个人，每本书有相应的时间花费。每个人只能抄序列中连续的书，并且每人至少抄一本书。求使得最大花费时间最小的分配方式。如果有多解尽量使前面的人的工作量小。

思路：dp[i][j]表示i个人抄前j本书的最大花费时间的最小值。dp[i][j]=min(dp[i][j],max(dp[i-1][k],sum[j]-sum[k]));然后求得这个值之后，从后往前贪心地尽量让后面的人多做。

```
for(i=2;i<=m;i++)
           for(j=i;j<=n;j++)
           {
               dp[i][j]=INF;
               for(k=i-1;k<j;k++)
                  dp[i][j]=min(dp[i][j],max(dp[i-1][k],
                      sum[j]-sum[k]));
```

2.20 往後走路

```
//給你一個n*m(n<=10,m<=100)的矩陣，
//矩陣每個點上有一個權值ai<=2^30
//你可以選擇一條路徑並選擇這條路徑每個點的的權值，
//使得這條路徑的權值之和最小
//每個點只能走到與它相鄰的右上角右下角和右邊三個點
//特別地，這個矩陣第一行和最後一行是相連的
//相同權值時，要求路徑的字典序最小，
//輸出這條由行的值構成的路徑，並輸出這個權值之和
#include<iostream>
#include<cstdio>
#include<algorithm>
#include<cstring>
using namespace std;
typedef long long ll;
int n,m;
ll ans,a[15][105],dp[15][105];
int head,Next[15][105];
int main()
{
  while(~scanf("%d%d",&n,&m))
  {
    for(int i=1;i<=n;++i)
    {
      for(int j=1;j<=m;++j)
      {
        scanf("%lld",&a[i][j]);
        dp[i][j]=1e18;
      }
    }
    ans=1e18;
    for(int j=m;j>=1;--j)//從右到左枚舉列
    {
      for(int i=1;i<=n;++i)//增序枚舉行確保head的字典序
          最小
      {
        if(j==m)dp[i][j]=a[i][j];
        else
        {
          int row[3]={i,i+1,i-1};
          if(i==n)row[1]=1;//第一行的下一行
          if(i==1)row[2]=n;//第一行的上一行
          sort(row,row+3);//滿足行的字典序最小
          for(int k=0;k<3;++k)
          {
            if(dp[i][j]>dp[row[k]][j+1]+a[i][j])
            {
              dp[i][j]=dp[row[k]][j+1]+a[i][j];
              Next[i][j]=row[k];//(i,j)->(row[k],j+1)
            }
          }
          }
        if(j==1)
        {
          if(ans>dp[i][j])
          {
            ans=dp[i][j];
            head=i;
          }
        }
```

```
      }
    }
    for(int row=head,col=1;col<=m;row=Next[row][col],
        col++)
    printf("%d%c",row,col==m?'\n':' ');
    printf("%lld\n",ans);
  }
  return 0;
}
```

2.21 找最大子段和

```
// 題目: 找最大子段和，如果有多个答案，那就找加一起的数
    最多的和起点最小的

#include <bits/stdc++.h>
using namespace std;
int a[20005];
fstream in,out;
int main()
{
    ios::sync_with_stdio(false);
    int n,c,s;
    in.open("data.txt");
    out.open("answer.txt");
    cin>>c;
    for(int t=1;t<=c;t++)
    {
        int st=1,en=-1,stm=1;
        cin>>s;
        memset(a,0,sizeof(a));
        for(int i=1;i<s;i++)
            cin>>a[i];
        int Max=-1,tmp=0;
        for(int i=1;i<s;i++)
        {
            if(tmp>=0)
                tmp+=a[i];
            else
            {
                tmp=a[i];
                stm=i;
            }
            if(tmp>Max||tmp==Max&&i-stm>en-st)
            {
                Max=tmp;
                st=stm;
                en=i;
            }
        }
        if(Max<=0)
            cout<<"Route "<<t<<" has no nice parts"<<
                endl;
        else
            cout<<"The nicest part of route "<<t<<" is
                between stops "<<st<<" and "<<en+1<<
                endl;
    //  if(t!=c)

    }
    in.close();
    out.close();
    return 0;
}
```

2.22 最大上升子序列

```
// Strategic Defense Initiative

// 題目：最大上升子序列，輸出一組解。

// 分析：dp，LIS。數據較小O（n^2）算法即可。

//            設以第i個數字作為最大上升子序列中的最後一
    個數的長度為f（i），則有轉移方程：

//             f（i）= max（f（j））  { 0=< j < i &&
    data[j] < data[i] }；

//            用一個數組記錄前驅，遞歸輸出即可。

// 說明：注意輸出格式有點糾結。
#include <iostream>
#include <cstdlib>
#include <cstring>
#include <cstdio>

using namespace std;

char buf[256];
int  data[10000];
int  dp[10000];
int  front[10000];

void output( int d, int s )
{
  if ( front[s] >= s )
    printf("Max hits: %d\n",d+1);
  else
    output( d+1, front[s] );
  printf("%d\n",data[s]);
}

int main()
{
  int n;
  while (~scanf("%d",&n)) {
    getchar();
    getchar();
    while ( n -- ) {
      char ch;
      int count = 0;
      while (gets(buf) && buf[0])
        data[count ++] = atoi(buf);
      for ( int i = 0 ; i < count ; ++ i ) {
        dp[i] = 1;
        front[i] = i;
        for ( int j = 0 ; j < i ; ++ j )
          if ( data[i] > data[j] && dp[i] < dp[j]+1 ) {
            dp[i] = dp[j]+1;
            front[i] = j;
          }
      }

      int max = 0;
      for ( int i = 1 ; i < count ; ++ i )
        if ( dp[i] > dp[max] )
          max = i;

      if ( count )
        output( 0, max );
      else printf("Max hits: 0\n");
      if ( n ) printf("\n");
    }
  }
  return 0;
}
```

3 Mathmatics

3.1 ax+by=gcd(a,b)

```
typedef pair<int, int> pii;
pii extgcd(int a, int b){
  if(b == 0) return make_pair(1, 0);
  else{
    int p = a / b;
```

```
    pii q = extgcd(b, a % b);
    return make_pair(q.second, q.first - q.second * p);
  }
}
```

3.2 Inverse

```
int inverse[100000];
void invTable(int b, int p) {
  inverse[1] = 1;
  for( int i = 2; i <= b; i++ ) {
    inverse[i] = (long long)inverse[p%i] * (p-p/i) % p;
  }
}

int inv(int b, int p) {
  return b == 1 ? 1 : ((long long)inv(p % b, p) * (p-p/
      b) % p);
}
```

3.3 LinearPrime

```
const int MAXP = 100; //max prime
vector<int> P;  // primes
void build_prime(){
  static bitset<MAXP> ok;
  int np=0;
  for (int i=2; i<MAXP; i++){
    if (ok[i]==0)P.push_back(i), np++;
    for (int j=0; j<np && i*P[j]<MAXP; j++){
      ok[ i*P[j] ] = 1;
      if ( i%P[j]==0 )break;
    }
  }
}
```

4 Data Structure

4.1 Sparse Table

```
const int MAXN = 200005;
const int lgN = 20;

struct SP{ //sparse table
  int Sp[MAXN][lgN];
  function<int(int,int)> opt;
  void build(int n, int *a){ // 0 base
    for (int i=0 ;i<n; i++) Sp[i][0]=a[i];

    for (int h=1; h<lgN; h++){
      int len = 1<<(h-1), i=0;
      for (; i+len<n; i++)
        Sp[i][h] = opt( Sp[i][h-1] , Sp[i+len][h-1] );
      for (; i<n; i++)
        Sp[i][h] = Sp[i][h-1];
    }
  }
  int query(int l, int r){
    int h = __lg(r-l+1);
    int len = 1<<h;
    return opt( Sp[l][h] , Sp[r-len+1][h] );
  }
};
```

4.2 Segment Tree

```
int n,m,i,a,b,c;
int ans[MAXN<<2],add[MAXN<<2],inp[MAXN<<2];

inline int ls(const int&p){
    return p<<1;
}

inline int rs(const int&p){
    return p<<1|1;
}

inline int Max(const int&x,const int&y){
    return x>y?x:y;
}

inline void push_up(const int&p,const int&tag){
    ans[p]=Max(ans[ls(p)],ans[rs(p)])+tag;
}

void build(const int l=1,const int r=n,const int p=1){
    if(l==r){
        get(ans[p]);
        inp[l]=ans[p];
        return;
    }
    int mid=(l+r)>>1;
    build(l, mid, ls(p));
    build(mid+1,r,rs(p));
    push_up(p,0);
}

inline void update(const int&x,const int&y,const int&k,
    const int&l=1,const int&r=n,const int&p=1){
    if(l>=x&&r<=y){
        add[p]+=k;
        ans[p]+=k;
        return;
    }
    int mid=(l+r)>>1;
    if(x<=mid){
        update(x,y,k,l, mid, ls(p));
    }
    if(y>mid){
        update(x,y,k,mid+1,r,rs(p));
    }
    push_up(p,add[p]);
}

inline int query(const int &x,const int &y,const int &
    tag=0,const int &l=1,const int &r=n,const int &p=1)
    {
    if(l>=x&&r<=y){
        return ans[p]+tag;
    }
    int mx=-1;
    int mid=(l+r)>>1;
    if(x<=mid){
        mx=Max(mx,query(x,y,tag+add[p],l, mid, ls(p)));
    }
    if(y>mid){
        mx=Max(mx,query(x,y,tag+add[p],mid+1,r,rs(p)));
    }
    return mx;
}
```

4.3 Djs

```
struct DisjointSet{
    int n, fa[MAXN];

      void init(int size) {
            for (int i = 0; i <= size; i++) {
                  fa[i] = i;
                    }
             }
```

```
        void find(int x) {
              return fa[x] == x ? x : find(fa[x]);
                }

          void unite(int x, int y) {
                p[find(x)] = find(y);
                  }

} djs;
```

4.4 Binary Indexed Tree

```
vector<int> bit;
int size;
int lowbit(int x){
    return x & (-x);
}
void update(int p, int val){
    while(p <= size){
        bit[p] += val;
        p += lowbit(p);
    }
}
int sum(int p){
    int ans = 0;
    while(p > 0){
        ans += bit[p];
        p -= lowbit(p);
    }
    return ans;
}
vector<int> countSmaller(vector<int>& nums) {
    if(nums.empty()){
        return vector<int>{};
    }
    size = nums.size();
    vector<int> ans(size, 0);
    bit = vector<int>(size + 1, 0);
    vector<int> tmp = nums;
    unordered_map<int, int> m;
    sort(tmp.begin(), tmp.end());
    for(int i = 0;i < size; ++i){
        m[tmp[i]] = i + 1;
    }
    for(int i = size - 1;i >= 0; --i){
        ans[i] = sum(m[nums[i]] - 1);
        update(m[nums[i]], 1);
    }
    return ans;
}
```

5 String

5.1 KMP

```
template<typename T>
void build_KMP(int n, T *s, int *f){ // 1 base
  f[0]=-1, f[1]=0;
  for (int i=2; i<=n; i++){
    int w = f[i-1];
    while (w>=0 && s[w+1]!=s[i])w = f[w];
    f[i]=w+1;
  }
}

template<typename T>
int KMP(int n, T *a, int m, T *b){
  build_KMP(m,b,f);
  int ans=0;

  for (int i=1, w=0; i<=n; i++){
    while ( w>=0 && b[w+1]!=a[i] )w = f[w];
    w++;
    if (w==m){
      ans++;
      w=f[w];
    }
  }
  return ans;
}
```

6 Dark Code

6.1 PBDS

```
#include<ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
#include <ext/pb_ds/detail/standard_policies.hpp>
typedef tree<int,null_type,less<int>,rb_tree_tag,
    tree_order_statistics_node_update> ordered_set;
```

7 Search

7.1 LIS

```
int LIS(vector<int>& s)
{
    if (s.size() == 0) return 0;

    vector<int> v;
    v.push_back(s[0]);

    for (int i = 1; i < s.size(); ++i)
    {
        int n = s[i];

        if (n > v.back())
            v.push_back(n);
        else
            *lower_bound(v.begin(), v.end(), n) = n;
    }

    return v.size();
}
```

7.2 Merge sort

```
void merge(int *vec, int start, int end)
{
    if (start >= end) return;
    int mid = start + ((end - start) >> 1);
    merge(vec, start, mid);
    merge(vec, mid + 1, end);
    for (int i = start; i <= mid; i++)
    {
        //ans += distance(lower_bound(vec + mid + 1,
            vec + end + 1, vec[i] + lower),
        //                upper_bound(vec + mid + 1,
            vec + end + 1, vec[i] + upper));
        //Do some cool stuffs
    }
    inplace_merge(vec + start, vec + mid + 1, vec + end
         + 1);
}
```

7.3 離散化

```cpp
int main()
{
    int n, in;
    vector<int> data, lib;
    cin >> n;
    for (int i = 0; i < n; i++)
    {
        cin >> in;
        data.push_back(in);
    }
    lib = data;
    sort(lib.begin(), lib.end());
    lib.erase(unique(lib.begin(), lib.end()), lib.end()
        );

    for (auto i:data)
        cout << lower_bound(lib.begin(), lib.end(), i)
            - lib.begin() << " ";
    cout << endl;
}
```

8 Others

8.1 1D/1D dp 優化

```cpp
#include<bits/stdc++.h>

int t, n, L;
int p;
char s[MAXN][35];
ll sum[MAXN] = {0};
long double dp[MAXN] = {0};
int prevd[MAXN] = {0};

long double pw(long double a, int n) {
    if ( n == 1 ) return a;
    long double b = pw(a, n/2);
    if ( n & 1 ) return b*b*a;
    else return b*b;
}
long double f(int i, int j) {
//    cout << (sum[i] - sum[j]+i-j-1-L) << endl;
    return pw(abs(sum[i] - sum[j]+i-j-1-L), p) + dp[j];
}
struct INV {
    int L, R, pos;
};
INV stk[MAXN*10];
int top = 1, bot = 1;
void update(int i) {
    while ( top > bot && i < stk[top].L && f(stk[top].L
        , i) < f(stk[top].L, stk[top].pos) ) {
        stk[top - 1].R = stk[top].R;
        top--;
    }
    int lo = stk[top].L, hi = stk[top].R, mid, pos =
        stk[top].pos;
    //if ( i >= lo ) lo = i + 1;
    while ( lo != hi ) {
        mid = lo + (hi - lo) / 2;
        if ( f(mid, i) < f(mid, pos) ) hi = mid;
        else lo = mid + 1;
    }
    if ( hi < stk[top].R ) {
        stk[top + 1] = (INV) { hi, stk[top].R, i };
        stk[top++].R = hi;
    }
}

int main() {
    cin >> t;
    while ( t-- ) {
        cin >> n >> L >> p;
        dp[0] = sum[0] = 0;
        for ( int i = 1 ; i <= n ; i++ ) {
            cin >> s[i];
            sum[i] = sum[i-1] + strlen(s[i]);
            dp[i] = numeric_limits<long double>::max();
        }
        stk[top] = (INV) {1, n + 1, 0};
        for ( int i = 1 ; i <= n ; i++ ) {
            if ( i >= stk[bot].R ) bot++;
            dp[i] = f(i, stk[bot].pos);
            update(i);
//            cout << (ll) f(i, stk[bot].pos) << endl;
        }
        if ( dp[n] > 1e18 ) {
            cout << "Too hard to arrange" << endl;
        } else {
            vector<PI> as;
            cout << (ll)dp[n] << endl;
        }
    }
    return 0;
}
```

8.2 Theorm - DP optimization

```
Monotonicity & 1D/1D DP & 2D/1D DP
-----------------------------------------------------
Definition xD/yD
1D/1D DP[j] = min(0≤i<j) { DP[i] + w(i, j) }; DP[0] = k
2D/1D DP[i][j] = min(i<k≤j) { DP[i][k - 1] + DP[k][j] }
     + w(i, j); DP[i][i] = 0
-----------------------------------------------------
Monotonicity
       c       d
  -----------------
a | w(a, c) w(a, d)
b | w(b, c) w(b, d)

Monge Condition
Concave(凹四邊形不等式): w(a, c) + w(b, d) >= w(a, d) +
     w(b, c)
Convex (凸四邊形不等式): w(a, c) + w(b, d) <= w(a, d) +
     w(b, c)

Totally Monotone
Concave(凹單調): w(a, c) <= w(b, d) -----> w(a, d) <= w
    (b, c)
Convex (凸單調): w(a, c) >= w(b, d) -----> w(a, d) >= w
    (b, c)
-----------------------------------------------------
1D/1D DP O(n^2) -> O(nlgn)
**CONSIDER THE TRANSITION POINT**
Solve 1D/1D Concave by Stack
Solve 1D/1D Convex by Deque
-----------------------------------------------------
2D/1D Convex DP (Totally Monotone) O(n^3) -> O(n^2)
h(i, j − 1) ≤ h(i, j) ≤ h(i + 1, j)
```

8.3 Mo's algorithm

```cpp
int l = 0, r = 0, nowAns = 0, BLOCK_SIZE, n, m;
int ans[];
struct QUE{
    int l, r, id;
    friend bool operator < (QUE a, QUE b){
        if(a.l / BLOCK_SIZE != b.l / BLOCK_SIZE)
            return a.l / BLOCK_SIZE < b.l / BLOCK_SIZE;
        return a.r < b.r;
    }
}querys[];

inline void move(int pos, int sign) {
    // update nowAns
```

```
}

void solve() {
    BLOCK_SIZE = int(ceil(pow(n, 0.5)));
    sort(querys, querys + m);
    for (int i = 0; i < m; ++i) {
        const QUE &q = querys[i];
        while (l > q.l) move(--l, 1);
        while (r < q.r) move(r++, 1);
        while (l < q.l) move(l++, -1);
        while (r > q.r) move(--r, -1);
        ans[q.id] = nowAns;
    }
}
```