A-最大子段和

注意数据范围,最大和会爆int

代码

```
#include <iostream>
#include <cstdio>
#include <string>
#include <cstring>
#include <cmath>
#include <cctype>
#include <cstdlib>
#include <algorithm>
#include <queue>
#include <stack>
#include <map>
#include <vector>
#include <list>
#include <set>
using namespace std;
#define CL(a) memset(a,0,sizeof(a))
#define Cl(a,b) memset(a,b,sizeof(a))
#define INF 0x7fffffff
#define LL long long
const int MOD = 1e9 + 7;
const int maxn = 1e6 + 100;
LL sum[maxn];
int main()
{
    int n,a;
    LL sum,ans;
    while (scanf("%d",&n) != EOF){
        ans = sum = 0;
        for (int i = 1;i <= n;i++){
            scanf("%d",&a);
            sum = max(sum+a,0ll);
            ans = max(sum,ans);
        }
        printf("%lld\n",ans);
    }
    return 0;
}
```

B - 循环数组最大子段和

思路

如果最大和在不循环的情况下得到,我们可以按照最大的连续子序列和的方法找到一个最大的和maxSum;

如果最大和的在循环的情况下得到,我们我们可以求一个最小连续子序列的和minSum,用数组总和tot-minSum就是在循环的情况下最大和;

最终答案取两者最大值ans = max{minSum,tot-maxSum}

代码

```
#include <iostream>
#include <cstdio>
#include <cstring>
#include <string>
#include <algorithm>
#include <set>
#include <stack>
#include <queue>
#include <vector>
#include <cctype>
using namespace std;
#define CL(a) memset(a,0,sizeof(a));
#define LL long long
const int INF= 0x7fffffff;
const int maxn = 1e6 + 100;
LL a[maxn];
int main()
{
    int n;
    LL maxSum,minSum,ans,sum,tot;
    while (scanf("%d",&n) != EOF){
        tot = 0;
        for (int i = 0;i < n;i++){
            scanf("%lld",&a[i]);
            tot += a[i];
        }
        maxSum = sum = 0;
        minSum = INF;
        for (int i = 0;i < n;i++){
            sum = max(sum + a[i],0ll);
            maxSum = max(maxSum,sum);
        }
        for (int i = 0;i < n;i++){
            sum = min(sum + a[i],0ll);
            minSum = min(minSum,sum);
```

```
        }
        ans = max(maxSum,tot - minSum);
        printf("%lld\n",ans);
    }
    return 0;
}
```

C - Common Subsequence

题意

求两个字符串最长公共子序列长度

代码

```
#include <iostream>
#include <cstdio>
#include <string>
#include <cstring>
#include <cmath>
#include <cctype>
#include <cstdlib>
#include <algorithm>
#include <queue>
#include <stack>
#include <map>
#include <vector>
#include <list>
#include <set>
using namespace std;
#define CL(a) memset(a,0,sizeof(a))
#define Cl(a,b) memset(a,b,sizeof(a))
#define INF 0x7fffffff
#define LL long long
const int MOD = 1e9 + 7;
const int maxn = 1e3 + 100;
char a[maxn];
char b[maxn];
int dp[maxn][maxn];
int main()
{
    while (scanf("%s%s",a,b) != EOF){
        CL(dp);
        int lena = strlen(a);
        int lenb = strlen(b);
        for (int i = 1;i <= lena;i++){
```

```
            for (int j = 1;j <= lenb;j++){
                dp[i][j] = max(dp[i-1][j-1] + (a[i-1]==b[j-1]),max(dp[i-1
            }
        }
        printf("%d\n",dp[lena][lenb]);
    }
    return 0;
}
```

D - 最长公共子序列Lcs

思路

先求最长公共子序列,然后从最后一位倒着找回去.

代码

```
#include <iostream>
#include <cstdio>
#include <string>
#include <cstring>
#include <cmath>
#include <cctype>
#include <cstdlib>
#include <algorithm>
#include <queue>
#include <stack>
#include <map>
#include <vector>
#include <list>
#include <set>
using namespace std;
#define CL(a) memset(a,0,sizeof(a))
#define Cl(a,b) memset(a,b,sizeof(a))
#define INF 0x7fffffff
#define LL long long
const int MOD = 1e9 + 7;
const int maxn = 1e3 + 100;
char a[maxn];
char b[maxn];
int dp[maxn][maxn];
void print(int x,int y)
{
    if (x == 0 || y == 0)
        return;
```

```
    if (a[x-1] == b[y-1]){
        print(x-1,y-1);
        printf("%c",a[x-1]);
    }
    else if (dp[x-1][y] >= dp[x][y-1]){
        print(x-1,y);
    }
    else{
        print(x,y-1);
    }
}
int main()
{
    while (scanf("%s%s",a,b) != EOF){
        int lena = strlen(a);
        int lenb = strlen(b);
        for (int i = 1;i <= lena;i++){
            for (int j = 1;j <= lenb;j++){
                dp[i][j] = max(dp[i-1][j-1]+(a[i-1]==b[j-1]),max(dp[i-1][
            }
        }
        print(lena,lenb);
        printf("\n");
    }
    return 0;
}
```

E - 编辑距离

思路

编辑距离，又称Levenshtein距离（也叫做Edit Distance），是指两个字串之间，由一个转成另一个所需的最少编辑操作次数。许可的编辑操作包括将一个字符替换成另一个字符，插入一个字符，删除一个字符。

给定字符串S和T，我们可以用一种特殊字符促成两个字符串的对齐。我们加的特殊字符是“-”，我们允许在S和T中任意添加这种特殊字符使得它长度相同，然后让这两个串“对齐”，最终两个串相同位置出现了不同字符，编辑距离就+1。

对于例子 我们实际上采取了这样的对齐方式：

12345
ABCF-

DB-FG

1.子问题: 用dp[x][y]表示表示编辑a串前x位和b串前y位的编辑距离

2.状态转移：

(1)当第x位和第y位对齐时

(1.1)若a[x]==b[y]时,此时不需要编辑，dp[x][y] = dp[x-1][y-1]

(2)当a[x] != b[y]时,此时需要编辑，dp[x][y] = dp[x-1][y-1] + 1

(2)当第x位和第y位不对齐时

(2.1)当第x位和第y-1位对齐时，此时需要编辑,dp[x][y] = dp[x][y-1] + 1 (2.2)

当第x-1位和第y位对其时。此时需要编辑,dp[x][y] = dp[x-1][y] + 1

3.自底向上求解：

初始化dp

```
    for (int i = 1;i <= lena;i++){
        dp[i][0] = i;
    }
    for (int i = 0;i <= lenb;i++){
        dp[0][i] = i;
    }
```

求解

```
    for (int i = 1;i <= lena;i++){
        for (int j = 1;j <= lenb;j++){
            dp[i][j] = min(dp[x][y] + (a[i-1] != b[j-1]),min(dp[x-1][y],d
        }
    }
```

代码

```
#include <iostream>
#include <cstdio>
#include <string>
#include <cstring>
#include <cmath>
#include <cctype>
#include <cstdlib>
#include <algorithm>
#include <queue>
#include <stack>
#include <map>
#include <vector>
#include <list>
```

```
#include <set>
using namespace std;
#define CL(a) memset(a,0,sizeof(a))
#define Cl(a,b) memset(a,b,sizeof(a))
#define INF 0x7fffffff
#define LL long long
const int MOD = 1e9 + 7;
const int maxn = 1e3 + 100;
int dp[maxn][maxn];
char a[maxn];
char b[maxn];
int main()
{
    while (scanf("%s%s",a,b) != EOF){
        int lena = strlen(a);
        int lenb = strlen(b);
        for (int i = 1;i <= lena;i++){
            dp[i][0] = i;
        }
        for (int i = 1;i <= lenb;i++){
            dp[0][i] = i;
        }
        for (int i = 1;i <= lena;i++){
            for (int j = 1;j <= lenb;j++){
                dp[i][j] = min(dp[i-1][j-1] + (a[i-1]!=b[j-1]),min(dp[i-1
            }
        }
        printf("%d\n",dp[lena][lenb]);
    }


    return 0;
}
```

F - 最长递增子序列

思路

要用nlogn的做法

lower_bound()函数

代码

```
#include <iostream>
```

```
#include <cstdio>
#include <string>
#include <cstring>
#include <cmath>
#include <cctype>
#include <cstdlib>
#include <algorithm>
#include <queue>
#include <stack>
#include <map>
#include <vector>
#include <list>
#include <set>
using namespace std;
#define CL(a) memset(a,0,sizeof(a))
#define Cl(a,b) memset(a,b,sizeof(a))
#define INF 0x7fffffff
#define LL long long
const int MOD = 1e9 + 7;
const int maxn = 5e4 + 100;
int a[maxn];
int dp[maxn];
int main()
{
    int n,t,ans;
    while (scanf("%d",&n) != EOF){
        Cl(dp,0x7f);
        dp[0] = -INF;
        ans = 0;
        for (int i = 1;i <= n;i++){
            scanf("%d",&a[i]);
        }
        for (int i = 1 ;i <= n;i++){
            t = lower_bound(dp+1,dp+n+1,a[i]) - dp;
            ans = max(ans,t);
            dp[t] = min(dp[t],a[i]);
        }
        printf("%d\n",ans);
    }
    return 0;
}
```

G - 子序列个数

思路

对于一个集合(不包含重复元素的序列)来说，一个集合的子集(子序列)的个数为2^n。 求解的方法:

设集合的子集个数为m,则集合中增加一个元素时，
所有不包含x这个元素的子集个数为m个
所有包含x这个元素的子集个数也为m个(相当于向原来每个子集中添加x这个元素) 所以，增加元素x后子集的总数为2*m
1个元素的的子集个数为2个(包含空序列)
则两个元素的子集列个数为2∧2个
……
包含n个元素的集合的子集(含空集)的总数为2∧n

接下来我们讨论包含重复元素的序列的子序列的个数。

我们每向序列中添加一个元素x时
如果这个元素与前面的元素不重复，那么此时子序列的个数为原来的2倍(同集合);
如果这个元素与前面的元素有重复，如果也按照子序列的个数为原来的2倍的方法来计算的话，就会与前面所有以x为结尾的子序列重复，因此我们要减去这一部分(也就是加入上一个x前的子序列的个数)

1.子问题: dp[i]表示序列前i位的子序列的个数

2.状态转移：

加入第i个元素时，
如果第i个元素与前面的元素不重复dp[i] =2 * dp[i-1]
如果第i个元素与前面第j个元素重复，那么dp[i] = 2*dp[i-1] - dp[j-1]

3.自底向上求解：

pos[x]表示元素x出现的位置

```
dp[0] = 1;
for (int i = 1;i <= n;i++){
    dp[i] = dp[i-1] * 2;
    if (pos[a[i]] > 0){
        dp[i] -= dp[pos[a[i]]-1];
    }
    pos[a[i]] = i;
}
```

代码

```
#include <iostream>
```

```
#include <cstdio>
#include <string>
#include <cstring>
#include <cmath>
#include <cctype>
#include <cstdlib>
#include <algorithm>
#include <queue>
#include <stack>
#include <map>
#include <vector>
#include <list>
#include <set>
using namespace std;
#define CL(a) memset(a,0,sizeof(a))
#define Cl(a,b) memset(a,b,sizeof(a))
#define INF 0x7fffffff
#define LL long long
const int MOD = 1e9 + 7;
const int maxn = 1e6 + 100;
LL dp[maxn];
int pos[maxn];
int main()
{
    int n,now;
    dp[0]=1;
    while (scanf("%d",&n) != EOF){
        CL(pos);
        for(int i=1;i<=n;i++){
            scanf("%d",&now);
            dp[i]=(dp[i-1]*2)%MOD;
            if (pos[now]){
               dp[i]=(dp[i] -dp[pos[now]-1] + MOD) % MOD;
            }
            pos[now] = i;
        }
        printf("%lld\n",dp[n]-1);
    }
    return 0;
}
```

H - 数塔

思路

1.定义状态:dp[i][j]表示到第i行第j个元素的最大值.

2.状态转移:

dp[i][j] = max(dp[i-1][j-1],dp[i-1][j]) + a[i][j];

代码

```
#include <iostream>
#include <cstdio>
#include <string>
#include <cstring>
#include <cmath>
#include <cctype>
#include <cstdlib>
#include <algorithm>
#include <queue>
#include <stack>
#include <map>
#include <vector>
#include <list>
#include <set>
using namespace std;
#define CL(a) memset(a,0,sizeof(a))
#define Cl(a,b) memset(a,b,sizeof(a))
#define INF 0x7fffffff
#define LL long long
const int MOD = 1e9 + 7;
const int maxn = 1e2 + 100;
int dp[maxn][maxn];
int main()
{
    int T,n;
    scanf("%d",&T);
    while (T--){
        scanf("%d",&n);
        CL(dp);
        int ans = 0;
        for (int i = 1;i <= n;i++){
            for (int j = 1;j <= i;j++){
                scanf("%d",&dp[i][j]);
                dp[i][j] += max(dp[i-1][j-1],dp[i-1][j]);
                ans = max(ans,dp[i][j]);
            }
        }
        printf("%d\n",ans);
    }
    return 0;
}
```

I - 逆序排列

思路

1.定义状态:dp[i][j]表示1~i所有排列中逆序数为j的数量
2.状态转移:

先看一个例子,
序列{3,2,1}的逆序数为3,
如果把4放在最后{3,2,1,4},逆序数仍为3;
如果把4放在倒数第二位,{3,2,4,1},逆序数为4;
如果把4放在倒数第三位,{3,4,2,1},逆序数为5;
如果把4放在倒数第四位,{4,3,2,1},逆序数为6;
我们发现4每向前移动一位,逆序数加1;

处理倒i,求长度为j的逆序数时,
要构成逆序数为j的序列,
我们可以把i放在逆序数为j-1的序列的最后一位; 我们可以把i放在逆序数为j-2的序列的倒数第二位; 我们可以把i放在逆序数为j-3的序列的倒数第三位;

我们可以的到下面一段代码

```
    for (int i = 1;i <= maxi;i++){
        dp[i][0] = 1;
    }
    for (int i = 1;i <= maxi;i++){
        for (int j = 1;j <= maxj;j++){
            for (int k = 0;k <= j && k < i;k++){
                dp[i][j] += dp[i-1][j-k];
            }
        }
    }
```

时间复杂度有点高,我们看看可不可以简化一下,我们发现计算dp[i][j]和dp[i][j-1]时似乎有重复计算的部分

j < i时;这个时候i有j+1种放法

长度为i-1,逆序数为j的第i-1位后面

长度为i-1,逆序数为j-1的第i-2位后面

长度为i-1,逆序数为j-2的第i-3位后面

长度为i-1,逆序数为j-3的第i-4位后面

....

长度为i-1,逆序数为0的第i-j位后面

例如:

```
dp[5][3] = dp[4][3] + dp[4][2] + dp[4][1] + dp[4][0]
dp[5][4] = dp[4][4] + dp[4][3] + dp[4][2] + dp[4][1] + dp[4][0]
```

dp[i][j] = dp[i][j-1] + dp[i-1][j]

j >= i时,这时候i有i种放法,

长度为i-1,逆序数为j的第i-1位后面

长度为i-1,逆序数为j-1的第i-1位前面

长度为i-1,逆序数为j-2的第i-2位前面

长度为i-1,逆序数为j-3的第i-3位前面

.....

长度为i-1,逆序数为j-(i-1)的第1位前面

例如:

```
dp[3][4] = dp[2][4] + dp[2][3] + dp[2][2]
dp[3][5] = dp[2][5] + dp[2][4] + dp[2][3]
```

`dp[i][j] = dp[i][j-1] + dp[i-1][j] - dp[i-1][j-i];

所以可以把代码优化成

```
    for(int i = 1;i <= maxi;i++){
        for (int j = 1;j <= maxj;j++){
            if (j >= i){
                dp[i][j] = dp[i][j-1] + dp[i-1][j] - dp[i-1][i-j];
            }
            else {
                dp[i][j] = dp[i][j-1] + dp[i-1][j];
            }
        }
    }
```

代码

```
#include <iostream>
#include <cstdio>
#include <string>
#include <cstring>
#include <cmath>
#include <cctype>
```

```
#include <cstdlib>
#include <algorithm>
#include <queue>
#include <stack>
#include <map>
#include <vector>
#include <list>
#include <set>
using namespace std;
#define CL(a) memset(a,0,sizeof(a))
#define Cl(a,b) memset(a,b,sizeof(a))
#define INF 0x7fffffff
#define LL long long
const int MOD = 1e9 + 7;
const int maxn = 1e3 + 10;
const int maxm = 2e4 + 100;
int dp[maxn][maxm];
void Init()
{
    for (int i = 0;i <= 1000;i++){
        dp[i][0] = 1;
    }
    dp[1][0] = 1;
    for (int i = 1;i <= 1000;i++){
        for (int j = 1;j <= 20000;j++){
            if (j>=i)
                dp[i][j] = (0ll + dp[i][j-1] + dp[i-1][j] - dp[i-1][j-i]+
            else
                dp[i][j] = (0ll + dp[i][j-1] + dp[i-1][j])%MOD;
        }
    }

}
int main()
{
    int n,k,T;
    Init();
    scanf("%d",&T);
    while (T--){
        scanf("%d%d",&n,&k);
        if (k > n*(n-1)/2){
            printf("0\n");
        }
        else{
            printf("%d\n",dp[n][k]);
        }
    }
    return 0;
}
```