D - 食塩水

$$\frac{\sum w_i p_i}{\sum w_i} \geq x$$
$$\sum w_i (p_i - x) \geq 0$$

```
from collections import *
from itertools import *
from functools import *


def LI():
    return list(map(int, input().split()))


def I():
    return int(input())

def solve():
    N,K=LI()
    W,P=[0]*N,[0]*N
    for i in range(N):
        W[i],P[i]=LI()

    def check(x)->bool:
        v=[W[i]*(P[i]-x) for i in range(N)]
        v.sort(reverse=True)
        return sum(v[:K])>=0

    L,R=0,10**9
    for _ in range(100):
        mid=(L+R)/2
        if check(mid):
            L=mid
        else:
            R=mid
    print("{0:.6f}".format(L))



solve()
```

D - 道を直すお仕事

$$\frac{\sum c_i}{\sum t_i} \leq x$$
$$\sum c_i - xt_i \leq 0$$

```
from collections import *
from functools import *
from itertools import *


class Dsu:
    def __init__(self, n):
        self.f = list(range(n))
        self.sz = [1] * n

    def find(self, x):
        xcopy = x
        while x != self.f[x]:
            x = self.f[x]
        while xcopy != x:
            self.f[xcopy], xcopy = x, self.f[xcopy]
        return x

    def same(self, u, v):
        return self.find(u) == self.find(v)

    def merge(self, u, v) -> bool:
        u = self.find(u)
        v = self.find(v)
        if u == v:
            return False
        self.sz[u] += self.sz[v]
        self.f[v] = u
        return True

    def size(self, x):
        return self.sz[self.find(x)]


def LI():
    return list(map(int, input().split()))


def I():
    return int(input())


def solve():
```

```
    N, M = LI()
    edges = []
    for _ in range(M):
        a, b, c, t = LI()
        edges.append((a, b, c, t))

    def check(x: float) -> bool:
        dsu = Dsu(N)
        es = []
        for a, b, c, t in edges:
            es.append((a, b, c-x*t))
        es.sort(key=lambda v: v[2])
        cost = 0
        for i in range(M):
            a, b, w = es[i]
            if w <= 0:
                cost += w
                dsu.merge(a, b)
            elif not dsu.same(a, b):
                cost += w
                dsu.merge(a, b)

        return cost <= 0

    L, R = 0.0, 10**9
    for _ in range(100):
        mid = (L+R)/2
        if check(mid):
            R = mid
        else:
            L = mid
    print("{0:.2f}".format(L))


solve()
```

D - Gathering Children

给定字符串 S ,最初每个单元格都有一个人
若 $S_i = R$ ， 则说明 i 位置的人走到右边的格子,
若 $S_i = L$ ， 则说明 i 位置的人走到左边的格子。
走 10^{100} 次后,描述每个单元格的人数

解法:

可以发现 10^{100} 是一个很大的偶数
单元格的状态会在一定的次数内稳定下来，且周期为2
而我们只需要偶数次的单元格状态
一次移动的花费是: $|S| \leq 10^5$
显然我们不能直接迭代移动回合
由于每次我们去往的地方是固定的，不妨考虑倍增
记录对于 i 位置，他第 p 次移动的地方

$$dp[0][i] = \begin{cases} i+1, s_i = R \\ i-1, s_i = L \end{cases}$$

已知第p次，i移动的位置为$j = dp[p][i]$
那么第$p+1$次, i移动的位置即为第p次j移动的位置
$$dp[p+1][i] = dp[p][dp[p][i]]$$
$$dp[p+1][i] = dp[p][j]$$

```
from collections import *
from itertools import *
from functools import *


def LI():
    return list(map(int, input().split()))


def I():
    return int(input())


def solve():
    S=input()
    N=len(S)
    M=32
    dp=[[0]*N for _ in range(M+1)]

    for i,ch in enumerate(S):
        dp[0][i]=i+1 if ch=='R' else i-1
    for p in range(M):
        for i in range(N):
            dp[p+1][i] = dp[p][dp[p][i]]
    ans=[0]*N
    res=''
    for i in range(N):
        ans[dp[M][i]]+=1
    for i,x in enumerate(ans):
        res+=str(x)
        if i!=N-1:
            res+=' '
    print(res)


solve()
```

A - Range Flip Find Route

```
from collections import *
from itertools import *
from functools import *


def LI():
    return list(map(int, input().split()))


def I():
    return int(input())


def solve():
    H,W=LI()
    g=[input() for _ in range(H)]

    INF=10**9
    dis=[[INF]*W for _ in range(H)]
    vis=[[False]*W for _ in range(H)]
    dis[0][0] = 1 if g[0][0]=='#' else 0

    dq=deque()
    dq.append((0,0))

    dx,dy=[0,1],[1,0]

    while dq:
        x,y=dq.popleft()
        if vis[x][y]:
            continue
        vis[x][y] = True

        for i in range(2):
            nx,ny=x+dx[i],y+dy[i]
            if 0<=nx<H and 0<=ny<W:
                if g[x][y]=='.' and g[nx][ny]=='#':
                    if dis[nx][ny]>dis[x][y]+1:
                        dis[nx][ny]=dis[x][y]+1
                        dq.append((nx,ny))
                else:
                    if dis[nx][ny]>dis[x][y]:
                        dis[nx][ny]=dis[x][y]
```

```
                        dq.appendleft((nx,ny))
    print(dis[H-1][W-1])


solve()
```

E - Smooth Subsequence

从数组A中选出最长子序列，此子序列满足相邻两项的差的绝对值不超过D

朴素做法

$$dp_i：以下标i为结尾的最长子序列$$
$$更新时遍历[0,i),若|a_i-a_j|\leq D$$
$$则dp_i=max(dp_i,dp_j+1)$$

```cpp
#include <bits/stdc++.h>
using i64 = long long;
void solve()
{
    int n,d;
    std::cin>>n>>d;
    std::vector<int>a(n);
    for(auto&i:a){
        std::cin >> i;
    }
    std::vector<int>dp(n);
    for(int i=0;i<n;i++){
        int m=0;
        for(int j=0;j<i;j++){
            if(std::abs(a[i]-a[j])<=d){
                m=std::max(m,dp[j]);
            }
        }
        dp[i]=m+1;
    }
    std::cout<<*std::max_element(dp.begin(),dp.end())<<'\n';
}
int main()
{
    std::cin.tie(nullptr)->sync_with_stdio(false);
    solve();
    return 0;
}
```

优化做法

dp_x：以数字x为结尾的最大子序列长度
当数字x计算贡献时
它会对区间$[x-d, x+d]$的每一个长度$+1$
对于这样的区间操作，我们使用线段树维护即可

```
int op(int l,int r){
    return std::max(l,r);
}
int e(){
    return 0;
}

void solve()
{
    int n,d;
    std::cin>>n>>d;
    std::vector<int>a(n);
    for(auto&i:a)std::cin>>i;
    int M=*std::max_element(a.begin(),a.end());
    atcoder::segtree<int,op,e>seg(M+1);

    for(int x:a){
        int l=std::max(0,x-d);
        int r=std::min(M,x+d);
        seg.set(x,seg.prod(l,r+1)+1);
    }
    std::cout<<seg.all_prod()<<'\n';
}
```