



Gean Dev

Pre TOI21 Editorial

Written by
Gean Dev Team



Problemset

Day No.	No.	ID	Problem Name	Time Limit	Memory Limit
0	1	saygeandev	Say, "GeanDev"!	1.0 second	256 megabytes
	2	hackslex	hackslex	1.0 second	256 megabytes
	3	aplusb	aplusb	1.0 second	256 megabytes
1	1	pinkslex	pinkslex	2.0 seconds	256 megabytes
	2	penguin	penguin	1.0 second	256 megabytes
	3	sttamx	sttamx	1.0 second	256 megabytes
2	1	relazioneproibita	ความสัมพันธ์ต้องห้ามในป่าแห่งจินตนาการ	1.0 second	32 megabytes
	2	bouquet	bouquet	1.0 second	256 megabytes
	3	liesplayingtruth	Lies Playing Truth	1.5 second	256 megabytes



Say, "GeanDev"!

1.0 second, 256 megabytes

Author : Marszspace

Prerequisite : I/O

Official Solution

ส่ง 2 รอบ รอบแรกเขียน "gean" รอบสองเขียน "dev"



Hackslex

1.0 second, 256 megabytes

Author : icy

Co-author : blackslex

Prerequisite : implementation

Official Solution

เนื่องจากปัญหานี้ต้องการหารหัสผ่านการเรียกฟังก์ชัน `try_hack` จึงสามารถลองเรียกใช้ฟังก์ชันนั้นได้เลย

Time Complexity : $\mathcal{O}(1)$

Solution Code

```
#include "hackslex.h"

long long hack_password() {
    long long password = try_hack();
    return password;
}
```



A plus B

1.0 second, 256 megabytes

Author : ttamx

Co-author : Marszspace

Prerequisite : Binary search

Official Solution

Sort the array beforehand. Then binary search on answer and count the number of pairs that have sum at most mid .

Time Complexity : $\mathcal{O}((N + M) \log(\max(A) + \max(B)) + N \log(N) + M \log(M))$

Solution Code

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

using namespace std;
using ll = long long;

int main(){
    cin.tie(nullptr)->sync_with_stdio(false);
    int n,m;
    ll k;
    cin >> n >> m >> k;
    vector<int> a(n),b(m);
    for(auto &x:a){
        cin >> x;
    }
    for(auto &x:b){
        cin >> x;
    }
    sort(a.begin(),a.end());
    sort(b.rbegin(),b.rend());
    int l=0,r=2E9;
    while(l<r){
        int mid=l+(r-l)/2;
        ll cnt=0;
        int pos=0;
        for(auto x:b){
            while(pos<n&&a[pos]+x<=mid){
                pos++;
            }
            cnt+=pos;
        }
        if(cnt>=k)r=mid;
        else l=mid+1;
    }
    cout << l;
}
```



Pinkslex

2.0 second, 256 megabytes

Author : ttamx

Co-author : blackslex, sleepntsheep, omsincoconut

Prerequisite : Binary Search, Dynamic Programming, Data Structure

Subtask 3, 4, 5, 6

เนื่องจากในปัญหาย่อยนี้จะแก้จุดเดียวของการ binary search โดยในการ binary search จะตรวจว่าคำตอบมีค่าอย่างน้อย mid “ได้หรือไม่” ซึ่งสามารถทำได้โดยการดูว่าแบ่งให้ได้อย่างน้อย K ช่วง โดยแต่ละช่วงมีผลรวมอย่างน้อย mid “ได้หรือไม่” ในการหาผลรวมของ subarray จะใช้ prefix sum ในการหา โดยจากนั้นนิยาม $psum[i] = \sum_{1 \leq j \leq i} A[j]$

ในการตรวจสอบจะใช้การหาจำนวนมากสุดที่แบ่งได้ ว่าสามารถแบ่งได้มากกว่า K ช่วงหรือไม่ ซึ่งจะใช้ dynamic programming ในแก้ไขนิยาม $dp[i]$ แทนจำนวนช่วงที่มากที่สุดที่สามารถแบ่งได้และจบที่ตำแหน่งที่ i จะได้ว่า

$$dp[0] = 0$$

$$dp[i] = \max\{dp[j] + 1 \mid 0 \leq j < i \wedge psum[i] - psum[j] \geq mid\}$$

โดยหากไม่มีค่า j ที่สอดคล้องกับเงื่อนไขข้างต้นจะให้ $dp[i] = -\infty$

คำตอบเราจะมีค่าอย่างน้อย mid “ได้” ถ้าเมื่อ $dp[N] \geq K$

สำหรับในปัญหาย่อยนี้ จะสังเกตได้ว่าค่า N มีค่าค่อนข้างน้อย ทำให้สามารถใช้วิธีการอื่นในการแก้ปัญหาได้ ซึ่งสามารถทำได้โดยการใช้ dynamic programming มาช่วยในการหาคำตอบ

Time Complexity : $\mathcal{O}(N^2 \log(\sum |A|))$

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

using namespace std;
using ll = long long;

int main(){
    cin.tie(nullptr)->sync_with_stdio(false);
    int n,k;
    cin >> n >> k;
    vector<ll> a(n+1);
    for(int i=1;i<n;i++){
        cin >> a[i];
        a[i]+=a[i-1];
```



```
    }
    auto check=[&](ll mid)->int {
        vector<int> dp(n+1);
        dp[0]=0;
        for(int i=1;i<=n;i++){
            dp[i]=-1e9;
            for(int j=0;j<i;j++){
                if(a[i]-a[j]>=mid){
                    dp[i]=max(dp[i],dp[j]+1);
                }
            }
        }
        return dp[n];
    };
    ll l=-1e9,r=1e14;
    while(l<r){
        ll m=l+(r-l+1)/2;
        if(check(m)>=k)l=m;
        else r=m-1;
    }
    cout << l << "\n";
}
```

Subtask 7,8,9

เนื่องจากในปัญหาอยู่นี่คำตอบจะมีค่าไม่เป็นลบเสมอ นั่นคือ $mid \geq 0$

ในการตรวจสอบค่า mid จึงสามารถทำได้ด้วยการหาลำดับย่อของ A ที่มี $A[i]$ และ $A[N]$ ที่ยาวที่สุด โดยที่ค่าตัวหลังจะต้องมีค่านากกว่าค่าตัวหน้าอย่างน้อย mid ซึ่งสามารถทำได้ด้วยการทำ Longest Increasing Subsequence แบบมีผลต่างใน $\mathcal{O}(N \log N)$

Time Complexity : $\mathcal{O}(N \log(N) \log(\sum |A|))$

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

using namespace std;

using ll = long long;

int main(){
    cin.tie(nullptr)->sync_with_stdio(false);
    int n,k;
    cin >> n >> k;
    vector<ll> a(n+1);
    for(int i=1;i<=n;i++){
        cin >> a[i];
        a[i]+=a[i-1];
    }
    auto check=[&](ll mid)->int {
        vector<ll> lis{0LL};
        for(int i=1;i<=n;i++){
            auto it=upper_bound(lis.begin(),lis.end(),a[i]-mid);
            if(i==n) return it-lis.begin();
            if(it==lis.end()) lis.emplace_back(a[i]);
            else if(it!=lis.begin())*it=min(*it,a[i]);
        }
        return 0;
    };
}
```



```
ll l=0,r=1e14;
while(l<r){
    ll m=l+(r-l+1)/2;
    if(check(m)>=k)l=m;
    else r=m-1;
}
cout << l << "\n";
```

Official Solution

จาก solution ในปัญหาอยู่อย่างที่แล้ว หาก mid มีค่าเป็นลบจะทำให้ค่าที่เก็บใน lis เปลี่ยนแบบไม่ monotone ทำให้ไม่สามารถทำ upperbound ได้ ซึ่งเราสามารถแก้ได้โดยใช้ std::set หรือ std::map แทนการใช้ std::vector โดยเมื่อใส่ค่าใหม่เข้าไปจะทำการลบค่าที่ทำให้มิ่ง monotone ออกทั้งหมด

Time Complexity : $\mathcal{O}(N \log(N) \log(\sum |A|))$

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

using namespace std;

using ll = long long;

int main(){
    cin.tie(nullptr)->sync_with_stdio(false);
    int n,k;
    cin >> n >> k;
    vector<ll> a(n+1);
    for(int i=1;i<=n;i++){
        cin >> a[i];
        a[i]+=a[i-1];
    }
    auto check=[&](ll mid)->int {
        map<ll,int> dat;
        dat[0LL]=0;
        for(int i=1;i<=n;i++){
            auto it=dat.upper_bound(a[i]-mid);
            if(it==dat.begin())continue;
            int cur=prev(it)->second+1;
            if(i==n)return cur;
            it=dat.upper_bound(a[i]);
            while(it!=dat.end()&&it->second<=cur)it=dat.erase(it);
            if(it==dat.begin()||prev(it)->second<cur)dat[a[i]]=cur;
        }
        return 0;
    };
    ll l=-1e9,r=1e14;
    while(l<r){
        ll m=l+(r-l+1)/2;
        if(check(m)>=k)l=m;
        else r=m-1;
    }
    cout << l << "\n";
}
```



Penguin

1 second, 256 megabytes

Author : kunzaZa183

Co-author : ttamx, omsincoconut, pimofthelims, spycoderyt

Prerequisite : Binary Search

Subtask 1

เนื่องจาก $N^2 = 2500 \leq 3000$ ทำให้เราสามารถเรียกฟังก์ชัน ask ระหว่างเพนกวินทุกคู่ที่เป็นไปได้เพื่อหาว่าเป็นคู่กันหรือไม่

จำนวนครั้งที่เรียกฟังก์ชัน : 2500 ครั้ง

```
#include <bits/stdc++.h>
#include "penguin.h"

using namespace std;

vector<pair<int, int>> find_couples(int n) {
    vector<pair<int, int>> res;
    for (int i = 0; i < n; i++) {
        for (int j = i + 1; j < n; j++) {
            if (ask(i, i, j, j)) {
                res.emplace_back(i, j);
            }
        }
    }
    return res;
}
```

Subtask 2

สามารถทำการ binary search เพื่อหาเพนกวินตัวแรกที่มีคู่ ด้วยการให้ $l_2 = 0$ และ $r_2 = N - 1$ และทำการเรียกฟังก์ชัน $ask(l_1, r_1, l_2, r_2)$ แบบนี้จนกว่าจะได้เพนกวินตัวแรกที่มีคู่แล้วจึงให้ l_1 และ r_1 เป็นหมายเลขของเพนกวินตัวนั้นแล้ว binary search หาเพนกวินอีกด้วย

จำนวนครั้งที่เรียกฟังก์ชันโดยประมาณ : $2 \lceil \log_2 N \rceil = 28$ ครั้ง

```
#include <bits/stdc++.h>
#include "penguin.h"

using namespace std;

vector<pair<int, int>> find_couples(int n) {
    int l = 0, r = n - 1;
```



```
int mid = (l + r) / 2;
if (ask(0, n - 1, l, mid))
    r = mid;
else
    l = mid + 1;
}
int cur = l;
l = 0, r = n - 1;
while (l < r) {
    int mid = (l + r) / 2;
    if (ask(cur, cur, l, mid))
        r = mid;
    else
        l = mid + 1;
}
return vector<pair<int, int>>({make_pair(cur, l)});
```

Subtask 3

ทำการ binary search โดยให้ $l_1 = r_1 = 0$ และหาเพนกวินตัวแรกสุดในช่วง (l, r) ที่เป็นคู่ของเพนกวินตัวแรก โดยที่ตอนแรก $l = 1$ และ $r = N - 1$

แล้วสมมติว่าเพนกวินที่ได้จากการ binary search คือเพนกวินตัวที่ x ก็ทำการ binary search ต่อไปด้วย $l = x + 1$ และ $r = N - 1$ ทำแบบนี้ซ้ำ M รอบก็จะได้คู่หันหมด

จำนวนครั้งที่เรียกฟังก์ชันโดยประมาณ : $M \lceil \log_2 N \rceil = 1400$ ครั้ง

```
#include <bits/stdc++.h>
#include "penguin.h"

using namespace std;

vector<pair<int, int>> find_couples(int n) {
    vector<pair<int, int>> vpii;
    for (int cur = 0; cur < n; cur++) {
        int l = cur, r = n - 1;
        while (l < r) {
            int mid = (l + r) / 2;
            if (ask(0, 0, l, mid))
                r = mid;
            else
                l = mid + 1;
        }
        if (ask(0, 0, l, l)) vpii.emplace_back(l, 0);
        cur = l;
    }
    return vpii;
}
```



Subtask 4

สำหรับทุกคู่เพนกวิน (u, v) ที่ $u < v$ เราจะทำการ binary search หา u ทั้งหมดที่เป็นไปได้ โดยวิธีนึงที่ทำได้คือการหาเพนกวินตัวแรกที่มีคู่แล้วขับ $l = x + 1$ คล้ายกับ subtask ที่ 3 โดยขั้นตอนนี้จะเรียกฟังก์ชันโดยประมาณ $(M + 1)\lceil \log_2 N \rceil$ ครั้ง

โดยทุกครั้งที่เราได้เพนกวิน u ใน (u, v) เราจะทำการ binary search หา v โดยทุกครั้งที่หา v จะต้องใช้ $\lceil \log_2 N \rceil$ ครั้ง ทำ M รอบก็เป็น $M\lceil \log_2 N \rceil$ ครั้ง

จำนวนครั้งที่เรียกฟังก์ชันโดยประมาณ : $(2M + 1)\lceil \log_2 N \rceil = 2814$ ครั้ง

```
#include <bits/stdc++.h>
#include "penguin.h"

using namespace std;

vector<pair<int, int>> find_couples(int n) {
    vector<pair<int, int>> ans;
    for (int cur = 0; cur < n - 1; cur++) {
        int l = cur, r = n - 1;
        while (l < r) {
            int mid = (l + r) / 2;
            if (ask(cur, mid, cur + 1, n - 1))
                r = mid;
            else
                l = mid + 1;
        }
        cur = l;
        if (cur >= n - 1) break;
        assert(cur < n - 1);
        l = cur + 1, r = n - 1;
        while (l < r) {
            int mid = (l + r) / 2;
            if (ask(cur, cur, l, mid))
                r = mid;
            else
                l = mid + 1;
        }
        ans.emplace_back(cur, l);
    }
    return ans;
}
```

Official Solution

ทำการล้ายกับ subtask 4 แต่ว่าสำหรับทุกเพนกวิน u ต้องเช็คด้วยว่ามี v ที่ยังไม่ถูกคำนึงถึงที่มี u เดียวกันอยู่หรือไม่

จำนวนครั้งที่เรียกฟังก์ชันโดยประมาณ : $2M\lceil \log_2 N \rceil = 2914$ ครั้ง



Solution Code

```
#include <bits/stdc++.h>
#include "penguin.h"

using namespace std;

vector<pair<int, int>> find_couples(int n) {
    vector<pair<int, int>> ans;
    for (int cur = 0; cur < n - 1; cur++) {
        int l = cur, r = n - 1;
        while (l < r) {
            int mid = (l + r) / 2;
            if (ask(cur, mid, cur + 1, n - 1))
                r = mid;
            else
                l = mid + 1;
        }
        cur = l;
        if (cur >= n - 1) break;
        assert(cur < n - 1);
        for (int pos = cur + 1; pos < n && ask(cur, cur, pos, n-1); pos++) {
            l = pos, r = n - 1;
            while (l < r) {
                int mid = (l + r) / 2;
                if (ask(cur, cur, pos, mid))
                    r = mid;
                else
                    l = mid + 1;
            }
            pos = l;
            ans.emplace_back(cur, pos);
        }
    }
    return ans;
}
```



I'm sttamx

1.0 second, 256 megabytes

Author : NortGLG

Co-author : ttamx, omsincoconut, blackslex

Prerequisite : Dynamic Programming, Observation

Subtask 1

เราสามารถสังเกตได้ว่าเราไม่จำเป็นต้องวางแผนบล็อกเพิ่มเลย เนื่องจากทุก ๆ บล็อกในด่านเริ่มต้นมีความสูงเท่ากันทั้งหมดแล้ว คำตอบจึงจะมีค่าเท่ากับ 0 เสมอ

Time Complexity: $\mathcal{O}(1)$

Subtask 2

ในปัญหาย่อยนี้เนื่องจากมีจำนวนบล็อกน้อยมาก เราเลยสามารถทำการ Brute force โดยการเลือก ตำแหน่งทั้งหมดที่เราจะทำการ Super Jump ได้แต่ละครั้งโดยใช้DFS ซึ่งจะมีรูปแบบทั้งหมด 2^{N-1} แบบ ที่เป็นไปได้ และหากเราพิจารณาระยะห่างระหว่างจุดที่ทำการ Super Jump สองบล็อกใด ๆ จะต้องห่างกันอย่างน้อย $K + 1$ บล็อก เสมอ เนื่องจากหลังจากใช้ Super Jump แล้ว จะต้องเดินอีกอย่างน้อย K ครั้ง และ การเดินอย่างน้อย K ครั้ง จะผ่านบล็อกอย่างน้อย $K + 1$ บล็อกนั่นเอง

โดยการเดินแบบไม่ Super Jump ระหว่างบล็อกที่ l ถึงบล็อกที่ r สังเกตว่าก่อนที่จะเดินในช่วงนี้ เราจะต้องไปยืนอยู่ที่ความสูงอย่างน้อยที่สุดคือ h_i ที่มากที่สุดในช่วง $[l, r]$ อย่างแน่นอน (ไม่เช่นนั้นจะไม่สามารถเดินโดยผ่านบล็อกที่มีความสูงเท่ากันหมดได้) ซึ่งสังเกตว่ายิ่งเราไปเดินที่ความสูงมากเท่าไหร่จะยิ่งต้องวางแผนบล็อกจำนวนเพิ่มขึ้น

ดังนั้นความสูงที่เราควรไปยืนก็คือความสูงที่น้อยที่สุดที่เป็นไปได้นั่นคือ $H = \max(h_l, h_{l+1}, h_{l+2}, \dots, h_r)$ เพื่อทำให้สามารถเดินจากบล็อกที่ l ไปยังบล็อกที่ r ด้วยจำนวนบล็อกจำนวนน้อยที่สุด ดังนั้นสำหรับทุก ๆ บล็อกในช่วง $[l, r]$ จะต้องเพิ่มความสูงให้มีค่าเท่ากับ H หน่วยพอดี ซึ่งจะต้องเติมบล็อกเป็นจำนวน $H - h_i$ บล็อก สำหรับทุก $l \leq i \leq r$ นั่นเอง

Time Complexity: $\mathcal{O}(N \times 2^N)$

```
#include<bits/stdc++.h>
using namespace std;
using ll = long long;
const int N = 16;
ll h[N];
int main() {
    cin.tie(nullptr)->sync_with_stdio(false);
    int n, k;
    cin >> n >> k;
    for (int i = 0; i < n; i++) {
        cin >> h[i];
```



```
}

ll ans = LLONG_MAX;
for (int i = 0; i < (1 << (n - 1)); i++) {
    int last = -1e9;
    bool can = 1;
    vector<int> jump;
    jump.push_back(0);
    for (int j = 1; j < n; j++) {
        if (i & (1 << (j - 1))) {
            if (j - last <= k) can = 0;
            last = j;
            jump.push_back(j);
        }
    }
    jump.push_back(n);
    if (!can) continue;
    int sz = jump.size();
    ll res = 0;
    for (int i = 0; i < sz - 1; i++) {
        ll H = 0;
        for (int j = jump[i]; j < jump[i + 1]; j++) {
            H = max(H, h[j]);
        }
        for (int j = jump[i]; j < jump[i + 1]; j++) {
            res += H - h[j];
        }
    }
    ans = min(ans, res);
}
cout << ans << '\n';
return 0;
}
```

Subtask 3

เราสามารถนิยาม Dynamic Programming ในข้อนี้ได้ โดยจะนิยาม $dp[i]$ เป็นจำนวนบล็อกที่ต้องวางน้อยที่สุด เพื่อที่จะเดินมาจบที่ตำแหน่งที่ i และสามารถใช้ Super Jump ต่อได้ทันที โดยเราจะกำหนดให้ $H = \max(h_{j+1}, h_{j+2}, \dots, h_i)$ ทำให้สามารถเขียนสมการ Dynamic Programming ได้ดังนี้

- $dp[i] = \min(dp[j] + (H - h_{j+1}) + (H - h_{j+2}) + \dots + (H - h_i))$

ซึ่งตำแหน่งของ j จะต้องห่างจาก i อย่างน้อย $K + 1$ หรือ $j = 0$ ก็ได้เนื่องจากเราจะเริ่มที่ความสูงเท่าใดก็ได้ ณ ตอนเริ่มต้น ดังนั้น เงื่อนไขของค่า j สำหรับ สมการข้างบนก็คือ $j < i - K$ หรือ $j = 0$ นั่นเอง

โดยในขั้นตอนการ Transition สามารถคำนวณค่า H สำหรับแต่ละ j โดยการ loop ใน $h_{j+1}, h_{j+2}, \dots, h_i$ และค่าของ $(H - h_{j+1}) + (H - h_{j+2}) + \dots + (H - h_i)$ ได้ใน $\mathcal{O}(N)$ ได้เลย

ดังนั้นเงื่อนใจเรามีคู่ (i, j) ที่ต้องคำนวณทั้งหมดประมาณ N^2 ครับแล้วเราจึงสามารถคำนวณ $dp[i]$ ได้ ณ ได้ใน $\mathcal{O}(N^3)$

และในส่วนของการเดินครั้งสุดท้าย (เดินมาจบที่บล็อกที่ N) เราจะต้องคำนวณแยกทีหลังเอาร่อง เนื่องจากคำตอบในข้อนี้ไม่สามารถตอบจากนิยามของ $dp[N]$ เพียงอย่างเดียวได้ โดยจำนวนบล็อกที่ต้องใช้หั้งหมดจะเท่ากับ $dp[j] + (H - h_{j+1}) + (H - h_{j+2}) + \dots + (H - h_N)$ เมื่อ $H = \max(h_{j+1}, h_{j+2}, \dots, h_N)$ เราจึงสามารถหา j ที่ทำให้ค่าดังกล่าวน้อยที่สุดได้เลย



Time Complexity: $\mathcal{O}(N^3)$

```
#include<bits/stdc++.h>
using namespace std;
using ll = long long;
const int N = 202;
ll dp[N];
ll h[N];
ll cal(int j, int i) {
    ll H = 0;
    ll sum = 0;
    for (int k = j + 1; k <= i; k++) H = max(H, h[k]);
    for (int k = j + 1; k <= i; k++) sum += H - h[k];
    return dp[j] + sum;
}
int main() {
    cin.tie(nullptr)->sync_with_stdio(false);
    int n, k;
    cin >> n >> k;
    ll sum = 0;
    for (int i = 1; i <= n; i++) {
        cin >> h[i];
    }
    dp[0] = 0;
    for (int i = 1; i <= n; i++) {
        dp[i] = cal(0, i);
        for (int j = 0; j < i - k; j++) {
            dp[i] = min(dp[i], cal(j, i));
        }
    }
    ll ans = LLONG_MAX;
    for (int j = 0; j <= n; j++) {
        ans = min(ans, cal(j, n));
    }
    cout << ans << '\n';
    return 0;
}
```

Subtask 4

จาก Subtask 3 เราสังเกตว่าหากเราเขียน Transition ของ dp ในรูปของผลรวมจะได้

- $dp[i] = \min(dp[j] + \sum_{k=j+1}^i H - h_k)$ สำหรับ $j < i - K$ หรือ $j = 0$

ซึ่งสังเกตว่าในแต่ละ k ค่าของ H จะเป็นค่าเดิมเสมอ เราเลยสามารถดึงออกมาราบบกการ และเขียนในรูปของ $(i - j) \times H$ ได้เลยก็คือ

- $dp[i] = \min(dp[j] + H \times (i - j) - \sum_{k=j+1}^i h_k)$ สำหรับ $j < i - K$ หรือ $j = 0$



โดยความสามารถสังเกตได้อีกว่า การที่เราทำนิยาม dp นี้ไปจนถึงบล็อกที่ N ในส่วนของ $\sum_{k=j+1}^i h_k$ เมื่อนำพจน์นี้จากทุก ๆ Transition ของ

dp มีรวมกันทั้งหมดแล้วจะได้เท่ากับ $\sum_{k=1}^N h_k$ หมายความว่าความสามารถดึงพจน์ของผลรวม h_k มาคำนวณในภายหลังได้เลย

เราเลยสามารถลดรูปของ $dp[i]$ ได้เป็น

- $dp[i] = \min(dp[j] + H \times (i - j))$ สำหรับ $j < i - K$ หรือ $j = 0$

ดังนั้นในแต่ละการ Transition เราจึงจำเป็นต้องคำนวณค่าของ $H = \max(h_{j+1}, h_{j+2}, \dots, h_k)$ เพียงเท่านั้น โดยสังเกตว่าเราไม่จำเป็นต้องคำนวณค่า H ใหม่สำหรับทุก ๆ j แต่ความสามารถคำนวณ H สำหรับ j ได้จาก H' ซึ่งเป็น H ของ $j + 1$ ได้ทันทีเลย เนื่องจากเรารู้ว่า $H' = \max(h_{j+2}, h_{j+3}, \dots, h_i)$ และเราต้องการ $H = \max(h_{j+1}, h_{j+2}, \dots, h_i)$ เราเลยสามารถคำนวณ $H = \max(h_{j+1}, \max(h_{j+2}, h_{j+3}, \dots, h_i)) = \max(h_{j+1}, H')$ ได้เลย ซึ่งเรายังสามารถใช้วิธีการเดียวกันเพื่อคำนวณจำนวนบล็อกที่ต้องใช้ในการเดินໄไปยบบล็อกที่ N ได้อีกด้วย

Time Complexity: $\mathcal{O}(N^2)$

```
#include<bits/stdc++.h>
using namespace std;
using ll = long long;
const int N = 5005;
ll dp[N];
ll h[N];
int main() {
    cin.tie(nullptr)->sync_with_stdio(false);
    int n, k;
    cin >> n >> k;
    ll sum = 0;
    for (int i = 1; i <= n; i++) {
        cin >> h[i];
        sum += h[i];
    }
    dp[0] = 0;
    for (int i = 1; i <= n; i++) {
        dp[i] = LLONG_MAX;
        ll mx = 0;
        for (int j = i - 1; j >= 0; j--) {
            mx = max(mx, h[j + 1]);
            if (j == 0 || j < i - k)
                dp[i] = min(dp[i], dp[j] + mx * (i - j));
        }
    }
    ll mn = dp[n];
    ll mx = 0;
    for (int j = n - 1; j >= 0; j--) {
        mx = max(mx, h[j + 1]);
        mn = min(mn, mx * (n - j) + dp[j]);
    }
    cout << mn - sum << '\n';
    return 0;
}
```



Official Solution

เราสามารถนำ Idea ของ Subtask 4 มาทำการ Observe สมบัติบางอย่างของ dp เพื่อทำให้สามารถคำนวณ dp ได้ไวขึ้นได้

เราจะสังเกตว่าสำหรับแต่ละ i หากค่า j น้อยกว่า i มากจะไม่ทำให้ $dp[i]$ น้อยลงอย่างแน่นอน โดยช่วงของค่า j ที่มีโอกาสเป็นคำตอบได้ก็คือ $i - 2K - 1 \leq j < i - K$ (หากพิจารณากรณี $i - 2K - 1 \geq 0$)

ซึ่ง Observation ดังกล่าวสามารถพิสูจน์ได้โดยการ Proof by Contradiction โดยบทพิสูจน์ถือว่าเป็นแบบฝึกหัดสำหรับผู้อ่าน

หลังจากเราได้ Observation ข้างต้นมาแล้ว สังเกตว่าสำหรับแต่ละ i เราจะต้องไปวนหาค่า j รวมแล้วจำนวนไม่เกินประมาณ $2K$ ตัวเท่านั้น ซึ่งเนื่องจากข้อจำกัดของ $K \leq \min(500, N - 1)$ ที่ค่อนข้างน้อย เราเลยสามารถใช้วิธีดังกล่าวเพื่อทำข้อนี้ได้เลย

Time Complexity: $\mathcal{O}(NK)$

Solution Code

```
#include<bits/stdc++.h>
using namespace std;
using ll = long long;
const int N = 100100;
ll dp[N];
ll h[N];
int main() {
    cin.tie(nullptr)->sync_with_stdio(false);
    int n, k;
    cin >> n >> k;
    ll sum = 0;
    for (int i = 1; i <= n; i++) {
        cin >> h[i];
        sum += h[i];
    }
    dp[0] = 0;
    for (int i = 1; i <= n; i++) {
        dp[i] = LLONG_MAX;
        ll mx = 0;
        for (int j = i - 1; j >= 0; j--) {
            mx = max(mx, h[j + 1]);
            if (j == 0 || j < i - k)
                dp[i] = min(dp[i], dp[j] + mx * (i - j));
            if (j < i - 2 * k - 1) break;
        }
    }
    ll mn = dp[n];
    ll mx = 0;
    for (int j = n - 1; j >= 0; j--) {
        mx = max(mx, h[j + 1]);
        mn = min(mn, mx * (n - j) + dp[j]);
    }
    cout << mn - sum << '\n';
    return 0;
}
```



Relazione Proibita Nella Foresta Fantastica

1.0 second, 32 megabytes

Author : rufflogix

Co-author : ttamx

Prerequisite : Dynamic Programming on Tree

Subtask 1

เนื่องจากรับประกันว่าทุกผืนดินเป็นประเภท 0 จึงทำให้ค่าหัวรวมเป็น 0

```
cout << 0 << '\n';
```

Subtask 2

เนื่องจากรับประกันว่าเมื่อไส่สัตว์ในทุกผืนดินประเภท 1, 2, 3 จะไม่พึດกูแน่ ๆ จึงทำให้ค่าหัวรวมเกิดจากผลรวมของค่าหัวของทุกผืนดิน

```
int ans = 0;
for (int i = 1; i <= V; i++) {
    ans += c[arr[i]];
}
```

Subtask 3

กรณีที่ $N \leq 15$ สามารถแก้โดยการ brute force ไปยังทุกสถานะที่เป็นไปได้ สามารถทำได้ทั้ง recursive function หรือ iterative loop โดยเก็บข้อมูลว่า parent ของ node ปัจจุบันมีสัตว์อยู่หรือไม่ และ child node ของ node ปัจจุบันถูกบังคับให้ว่างหรือไม่

```
int solve(int u, bool par_selected, bool cur_selected, int p) {
    if (cur_selected && (arr[u] == 0 || (par_selected && (arr[u] == 1 || arr[u] == 2))))
        return -1e9;

    int res = cur_selected ? c[arr[u]] : 0;

    for (int v: adj[u]) {
        if (v == p) continue;
        if (cur_selected) {
            int tmp = solve(v, true, false, u);
            if (arr[u] == 1) tmp = max(tmp, solve(v, true, true, u));
            res += tmp;
        } else {
            res += max(solve(v, false, false, u), solve(v, false, true, u));
        }
    }
}
```



```
    }
    return res;
}
```

Subtask 4

กรณีผู้คนจะเรียงตัวกันเป็นเส้นตรง เราจึงสามารถที่จะใช้ array ในการเก็บการเรียงตัวของผู้คนได้ แล้วจึงค่อยไล่พิจารณาจากซ้ายไปขวา $1 \leq i \leq N$

Subtask 5, 6, 7

แก้คล้าย ๆ กับ subtask 3 แต่จะจงเฉพาะผู้คนในประเทศที่กำหนด

Official Solution

Time Complexity : $\mathcal{O}(N)$

Solution Code

นำ subtask 3 มาเพิ่ม memory เพื่อจำสถานะที่เคยคำนวณไว้แล้ว เพื่อลดระยะเวลาในการท่องสถานะต่าง ๆ

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

using namespace std;

int c[4];
int arr[200001];
vector<int> adj[200001];
int dp[200001][2][2];
bool visited[200001][2][2];

int solve(int u, bool par_selected, bool cur_selected, int p) {
    if (cur_selected && (arr[u] == 0 || (par_selected && (arr[u] == 1 || arr[u] == 2))))
        return -1e9;

    if (visited[u][par_selected][cur_selected])
        return dp[u][par_selected][cur_selected];

    int res = cur_selected ? c[arr[u]] : 0;

    for (int v : adj[u]) {
        if (v == p) continue;
        if (cur_selected) {
            int tmp = solve(v, true, false, u);
            if (arr[u] == 1) tmp = max(tmp, solve(v, true, true, u));
            res += tmp;
        }
    }
    return res;
}
```



```
        } else {
            res += max(solve(v, false, false, u), solve(v, false, true, u));
        }
    }

visited[u][par_selected][cur_selected] = true;
return dp[u][par_selected][cur_selected] = res;
}

int main() {
ios_base::sync_with_stdio(0), cin.tie(0);

int V;
cin >> V;

for (int i = 1; i <= 3; i++) cin >> c[i];
for (int i = 1; i <= V; i++) cin >> arr[i];

for (int i = 1; i <= V - 1; i++) {
    int u, v;
    cin >> u >> v;

    adj[u].push_back(v);
    adj[v].push_back(u);
}

cout << max(solve(1, false, false, -1), solve(1, false, true, -1)) << '\n';

return 0;
}
```



Bouquet

1.0 second, 256 megabytes

Author : ttamx

Co-author : blackslex

Prerequisite : Greedy, Dijkstra's Algorithm

Official Solution

วิธีการเลือกที่ดีที่สุดคือการเลือก K ช่อที่ถูกที่สุด

 **ttamx** 9/5/2568 1:48
key idea:
1. เราจะค่อย ๆ visit ทีละ combination จากถูกสุดไปแพงสุด
2. sort ตาม p จากน้อยไปมาก
3. state ของแต่ละ combination จะมี i แทนชนิดสุดท้ายที่เลือก j แทนจำนวนดอกของชนิดนั้นที่เลือก
4. transition ที่ต้องพิจารณาจะมี หยิบดอกชนิดปัจจุบันเพิ่ม 1 กับ เลิกหยิบชนิดนั้นแล้วไปหยิบดอกชนิดถัดไป 1 ดอก (ขาดเคลื่อนไหว?)
5. จะเห็นว่าเรายังไม่พิจารณาเคสที่มีชนิดที่ไม่ถูกหยิบเลย (ทำไงดี?)
6. เราอาจจะเพิ่มอีก transition คือ ถ้าดอกปัจจุบันมี 1 ดอก เราจะต้องถอนน้ำไป และวิ่งหยิบชนิดถัดไป 1 ดอกแทน เมื่อจากเรา sort ตาม p มาแล้ว มันแพงขึ้นแน่นอน!
done!

โปรดอ่าน code เพื่อความเข้าใจ

Time Complexity : $\mathcal{O}(N \log(N) + K \log(K))$

Solution Code

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

using namespace std;

using ll = long long;
using T = tuple<ll,int,int>;

const int N=500005;

int n,k;
int p[N];
vector<pair<int,int>> a;
ll sum,ans;
priority_queue<T,vector<T>,greater<T>> pq;

int main(){
    cin.tie(nullptr)->sync_with_stdio(false);
    cin >> n >> k;
    for(int i=1;i<=n;i++){
        cin >> p[i];
        sum+=p[i];
    }
}
```



```
for(int i=1;i<=n;i++){
    int x;
    cin >> x;
    if(x>1)a.emplace_back(p[i],x-1);
}
ans=sum*k;
k--;
n=a.size();
sort(a.begin(),a.end());
if(k>0)pq.emplace(a[0].first,0,1);
while(k--){
    assert(!pq.empty());
    auto [v,i,j]=pq.top();
    pq.pop();
    ans+=v;
    if(j<a[i].second)pq.emplace(v+a[i].first,i,j+1);
    if(i+1<n){
        pq.emplace(v+a[i+1].first,i+1,1);
        if(j==1)pq.emplace(v+a[i+1].first-a[i].first,i+1,1);
    }
}
cout << ans << "\n";
}
```



Lies playing Truth

1.5 second, 256 megabytes

Author : omsincoconut

Co-author : ttamx

Prerequisite : DFS, DP on Tree

Official Solution

Main Idea: Graph จะไม่มีซึ้งจุดเดี่ยงก็ต่อเมื่อไม่มี cycle ที่ผลรวมของ X เป็นเลขคี่

-- TLDR --

Time Complexity : $\mathcal{O}(N + M)$

Solution Code

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

using namespace std;

int main(){
    cin.tie(nullptr)->sync_with_stdio(false);
    int n,m;
    cin >> n >> m;
    vector<vector<tuple<int,int,int>>> adj(n);
    for(int i=0;i<m;i++){
        int u,v,w;
        cin >> u >> v >> w;
        u--,v--;
        adj[u].emplace_back(v,w,i);
        adj[v].emplace_back(u,w,i);
    }
    vector<bool> mark(m);
    vector<int> col(n,-1),dep(n),bad(n),good(n);
    int cnt=0;
    function<void(int)> dfs=[&](int u){
        for(auto [v,w,i]:adj[u]){
            if(col[v]==-1){
                mark[i]=true;
                col[v]=col[u]^w;
                dep[v]=dep[u]+1;
                dfs(v);
                bad[u]+=bad[v];
                good[u]+=good[v];
            }else if(dep[u]-dep[v]>1){
                if(col[v]!=(col[u]^w)){
                    cnt++;
                    bad[u]++;
                    bad[v]--;
                }
            }
        }
    };
    for(int i=0;i<n;i++)
        if(mark[i])
            dfs(i);
}
```



```
        }else{
            good[u]++;
            good[v]--;
        }
    }
};

for(int i=0;i<n;i++){
    if(col[i]==-1){
        col[i]=0;
        dfs(i);
    }
}
if(cnt==0){
    cout << m << "\n";
    for(int i=1;i<=m;i++)cout << i << " \n"[i==m];
    exit(0);
}
vector<int> ans;
function<void(int)> dfs2=[&](int u){
    for(auto [v,w,i]:adj[u]){
        if(mark[i]&&dep[v]>dep[u]){
            if(bad[v]==cnt&&good[v]==0){
                ans.emplace_back(i);
            }
            dfs2(v);
        }else if(dep[u]-dep[v]>1){
            if(cnt==1&&col[v]!=(col[u]^w)){
                ans.emplace_back(i);
            }
        }
    }
};
for(int i=0;i<n;i++){
    if(dep[i]==0){
        dfs2(i);
    }
}
sort(ans.begin(),ans.end());
if(ans.empty()){
    cout << -1 << "\n";
}else{
    cout << "0\n";
    exit(0);
    cout << ans.size() << "\n";
    for(int i=0;i<ans.size();i++)cout << ans[i]+1 << " \n"[i==ans.size()-1];
}
}
```