SOLUTION 19/10

Bài 1: Mê cung tình ái

Thuật toán: Toán

1	2	9	10	25
4	3	8	11	24
5	6	7	12	23
16	15	14	13	22
17	18	19	20	21

- Có mê cung như hình bên. Đầu tiên, ta cần xác định giá trị của ô thuộc đường chéo (có $x = y$). Dễ dàng nhận ra công thức để tính ô này: $x \times (x - 1) + 1$.
- Đặt $m = \max(x, y)$. Ta tính ô chéo thứ m dựa trên công thức trên. Gọi giá trị ấy là z.
- Xét hai trường hợp:
    - Nếu m chẵn: Ô cần tìm có giá trị là $z + (x - y)$.
    - Nếu m lẻ: Ô cần tìm có giá trị là $z - (x - y)$.

Bài 2: Lật bánh

Subtask 1: Duyệt 2^{2N}.

Subtask 2: Kiểm tra trường hợp kết quả $= 1$, nếu thời điểm N có khoảng lật, với kết quả $= 2$ thì tìm 2 đoạn mà có chênh lệch trong khoảng N bằng cách duyệt K^2.

Subtask 3: Quy hoạch động $dp[i][j]$ là số lần lật ít nhất đến thời điểm i và mặt trên đã được rán j giây:

– Đến thời điểm i nếu không lật bánh thì $dp[i][j] = dp[i-1][j]$

– Nếu có thể lật bánh thì $dp[i][j] = dp[i-1][i-j] + 1$

Subtaks 4, 5: Ta có nhận xét rằng trong một khoảng thời gian có thể lật từ $[l_i, r_i]$ thì tối đa ta chỉ lật 2 lần, do đó ta chỉ cần quan tâm đến các thời điểm cuối cùng của mỗi khoảng.

Gọi $dp[i][j]$ là số lần lật ít nhất đến thời điểm r_i và mặt trên đã được rán j giây:

– $dp[i][j] = dp[i-1][j]$ nếu không lật,

– $dp[i][j] = min_{l_i \le t \le r_i}(dp[i-1][t-j]) + 1$ khi dùng 1 lật.

– $dp[i][j] = min_{j-(r_i-l_i) \le j' < j}(dp[i-1][j']) + 2$ khi dùng 2 lật.

Ta sẽ dùng deque để tính nhanh được $dp[i][j]$ với độ phức tạp $O(NK)$. Một số cách dp khác có thể sẽ cần dùng cấu trúc cây để tính nhanh thì thuật toán là $O(NK\ log\ N)$ thì chỉ qua được subtask 4.

Code:

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

using namespace std;

int n, k;
int dp[2][500005];
int l[105], r[105];

int main() {
    cin >> n >> k;
    for (int i = 1; i <= k; i++) {
        cin >> l[i] >> r[i];
    }
    memset(dp[0], 63, sizeof dp[0]);
    int x = 0, y = 1;
    dp[0][0] = 0;
    for (int i = 1; i <= k; i++, x ^= 1, y ^= 1) {
        memset(dp[y], 63, sizeof dp[y]);
        // 0 lật
        for (int j = 0; j <= min(n, r[i]); j++) {
            dp[y][j] = min(dp[y][j], dp[x][j]);
        }

        // 1 lật
        deque<int> st;
        for (int j = min(n, r[i]), u = 0; j > 0; j--) {
            while (u <= r[i] - j && u <= n) {
                while (!st.empty() && dp[x][st.back()] >= dp[x][u]) {
                    st.pop_back();
                }
                st.push_back(u);
                u++;
            }
            while (!st.empty() && st.front() < l[i] - j) {
                st.pop_front();
            }
            if (!st.empty()) {
                dp[y][j] = min(dp[y][j], dp[x][st.front()] + 1);
            }
        }

        // 2 lật
        st.clear();
        int d = r[i] - l[i];
        for (int j = 1, u = 0; j <= min(n, r[i]); j++) {
            while (u < j) {
                while (!st.empty() && dp[x][st.back()] >= dp[x][u]) {
                    st.pop_back();
```

```
                }
                st.push_back(u);
                u++;
            }
            while (!st.empty() && st.front() < j - d) {
                st.pop_front();
            }
            if (!st.empty()) {
                dp[y][j] = min(dp[y][j], dp[x][st.front()] + 2);
            }
        }
    }
    if (dp[x][n] > 1e9) {
        dp[x][n] = -1;
    }
    cout << dp[x][n];
}
```

Bài 3: Truy bắt tội phạm

Mấu chốt của bài toán này là phải tìm được 2 đỉnh xuất phát và kết thúc của k đỉnh đã cho, từ đó ta có thể suy ra được các đỉnh thỏa mãn sẽ là từ một trong 2 đỉnh tìm được đi qua k đỉnh đã cho và đi đến các đỉnh này.

Subtask 1: Đồ thị cho là dạng đường thẳng, do đó ta chỉ cần tìm 2 đỉnh lá sau đó tiến dần vào trong để tìm hai đỉnh xuất phát và kết thúc trong k đỉnh đã cho, kết quả là những đỉnh từ 2 đỉnh tìm được đi đến 2 lá tương ứng.

Subtask 2: Đồ thị dạng cây, ta coi một trong k đỉnh đã là cây có gốc và dfs từ gốc xuống, khi dfs đến đỉnh u ta sẽ đếm xem nó có bao nhiêu nhánh chứa các đỉnh trong k đỉnh đã cho gọi là c_u, nếu u không phải là gốc thì $c_u \leq 1$ thì mới tồn tại đường đi nhỏ nhất qua k đỉnh đã cho, ngược lại thì vô nghiệm. Nếu u là 1 đỉnh trong k đỉnh đã cho và $c_u = 0$ có nghĩa u chính là 1 trong 2 đỉnh xuất phát và kết thúc. Nếu ta tìm đc 2 đỉnh có $c_u = 0$ thì đấy chính là 2 đỉnh cần tìm, ngược lại thì đỉnh gốc chính là đỉnh còn lại. Từ 2 đỉnh tìm được ta loang ra các đỉnh có $c_u = 0$, kết quả chính là số lượng đỉnh tìm được khi loang ra.

Subtask 3, 4: Duyệt từng đỉnh y, ta có $d[y][u]$ là khoảng cách nhỏ nhất từ y đến u. Dựa vào $d[y][u]$ ta có thể biết thứ tự cần đi qua k đỉnh đã cho như thế nào (đỉnh u đi qua trước v trên đường đi từ x đến y nếu $d[y][u] > d[y][v]$). Sau khi có thứ tự $u_1, u_2, \dots, u_k$ ta có thể kiểm tra được y có phải đỉnh thỏa mãn hay không dựa vào điều kiện $d[y][u_i] = d[u_i][u_{i+1}] + d[y][u_{i+1}]$. Với $n \leq 100$ ta có thể sử dụng luôn floyd để xây dựng mảng d, với $n \leq 1000$ thì cần dùng Dijkstra và hàng đợi ưu tiên.

Subtask 5: Với subtask này ta tìm được 2 đỉnh xuất phát và kết thúc trong 2 lần Dijkstra, lần đầu là Dijkstra từ 1 đỉnh bất kì trong tập k đỉnh và tìm đỉnh xa nhất s, và Dijkstra lần 2 để tìm đỉnh t còn lại. Sau khi tìm được 2 đỉnh này, ta Dijkstra trạng thái từ 2 đỉnh này đến các đỉnh trong đồ thị với mỗi trạng thái là $(u, state)$ với $state$ ở đây mô tả những đỉnh trong tập k đỉnh đã đi qua. Những đỉnh thỏa mãn sẽ có $d[u][state] = ds[u]$ ($d[u][state]$ là đường đi ngắn nhất đến u và trạng thái qua các đỉnh trong tập k đỉnh là $state = 2^k - 1$, $ds[u]$ là đường đi ngắn nhất từ s đến u).

Độ phức tạp sẽ là $O(n * 2^5 * (logn + 5))$.

Subtask 6: Giống subtask 5 ở tìm s và t, tuy nhiên ta sẽ quy hoạch động $c[u]$ là số đỉnh nhiều nhất trong tập k đỉnh đã cho trong các đường đi ngắn nhất từ s đến u. Như vậy những đỉnh u thỏa mãn sẽ có $c[u] = k$. Độ phức tạp chỉ là $O((n + m) \, log \, n)$.

Code:

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

#define rep(i, a, b) for(int i = a; i < (b); ++i)
#define trav(a, x) for(auto& a : x)
#define all(x) x.begin(), x.end()
#define sz(x) (int)(x).size()
#define hash dhsjakhd
typedef long long ll;
typedef pair<int, int> pii;
typedef pair<ll, ll> pll;
typedef vector<int> vi;
typedef vector<ll> vl;
typedef long double ld;

ll n,m,T,k;
const ll big = 1000000007;
const ll big2 = 998244353;

const int MAXN = 200000;

vector<vl> C(MAXN, vl());
vector<vl> CW(MAXN, vl());
vl special;
bool is_special[MAXN] = {0};

vl ind;

// to check connectedness
bool connected(){
```

```
    bool mark[MAXN] = {0};
    int vis = 0;
    queue<ll> Q;
    Q.push(0);
    mark[0] = 1;
    vis++;
    while(!Q.empty()){
        ll i = Q.front();
        Q.pop();
        for(int c1 = 0; c1 < sz(C[i]); c1++){
            ll j = C[i][c1];
            if(!mark[j]){
                mark[j] = 1;
                vis++;
                Q.push(j);
            }
        }
    }
    return (vis == n);
}
//

ll upd[MAXN] = {0};
ll counter = 0;
ll DIST[MAXN] = {0};

void dijkstra(ll start){
    counter++;
    for(int c1 = 0; c1 < n; c1++){
        DIST[c1] = -1;
    }
    priority_queue<pll> pq;
    pq.push({0,start});
    DIST[start] = 0;
    while(!pq.empty()){
        ll i = pq.top().second;
        pq.pop();
        if(upd[i] != counter){
            upd[i] = counter;
            for(int c1 = 0; c1 < sz(C[i]); c1++){
                ll j = C[i][c1];
                ll w = CW[i][c1];
                if(DIST[j] == -1 || DIST[j] > DIST[i]+w){
                    DIST[j] = DIST[i]+w;
                    pq.push({-DIST[j],j});
                }
            }
        }
    }
}
```

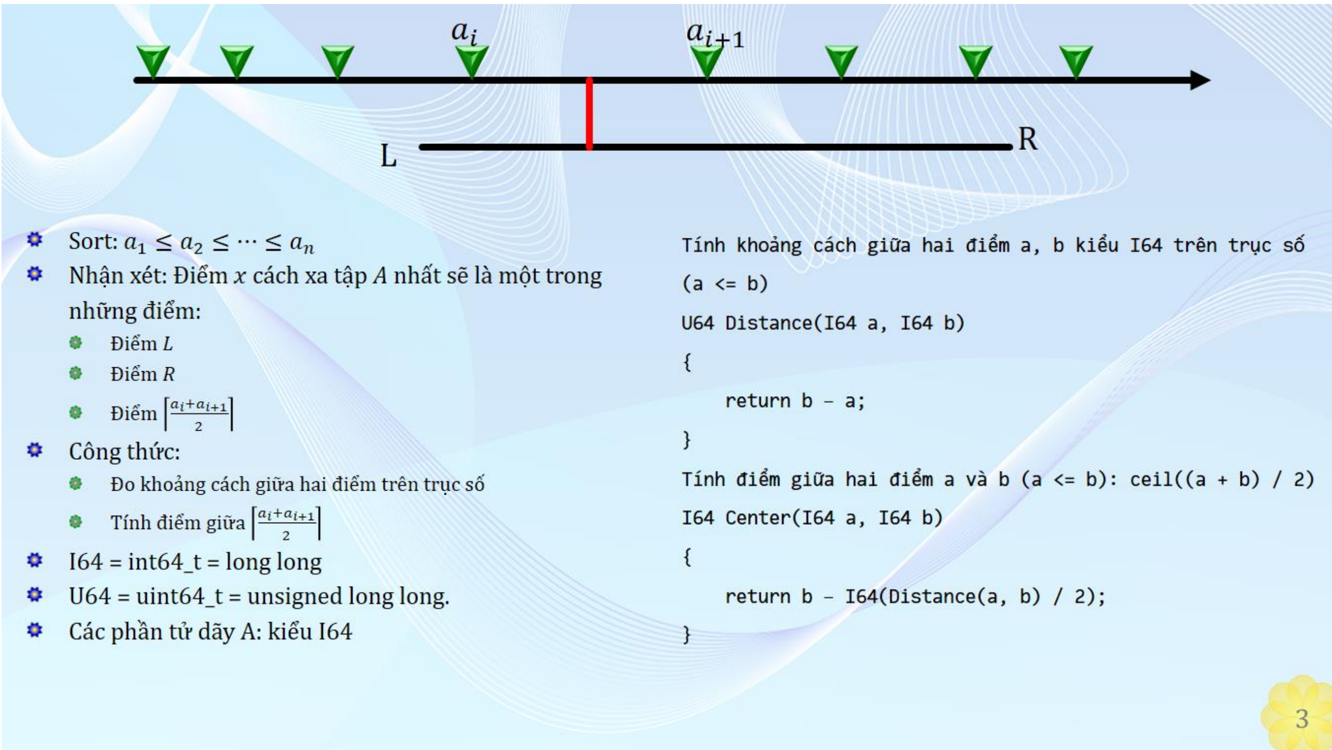
```
ll start1, start2;

bool comp(ll i, ll j){
    return DIST[i] < DIST[j];
}

bool ANS[MAXN] = {0};
ll specials[MAXN] = {0};

void solve(ll start){
    dijkstra(start);
    sort(all(ind),comp);
    for(int c1 = 0; c1 < n; c1++){
        specials[c1] = 0;
    }
    for(int c1 = 0; c1 < n; c1++){
        ll a = ind[c1];
        specials[a] += is_special[a];
        if(specials[a] == k)ANS[a] = 1;
        for(int c2 = 0; c2 < sz(C[a]); c2++){
            ll j = C[a][c2];
            ll w = CW[a][c2];
            if(DIST[j] == DIST[a]+w)specials[j] =
max(specials[a],specials[j]);
        }
    }
}

int main() {
    ios_base::sync_with_stdio(0);
    cin.tie(0);
    //freopen("input.txt","r",stdin);
    //freopen("autput.txt","w",stdout);
    ll a,b,c;

    cin >> n >> m >> k;
    for(int c1 = 0; c1 < k; c1++){
        cin >> a;
        a--;
        special.push_back(a);
        is_special[a] = 1;
    }
    for(int c1 = 0; c1 < n; c1++){
        ind.push_back(c1);
    }
    for(int c1 = 0; c1 < m; c1++){
        cin >> a >> b >> c;
        a--;
        b--;
```

```
        C[a].push_back(b);
        C[b].push_back(a);
        CW[a].push_back(c);
        CW[b].push_back(c);
    }
    assert(connected());

    if(k == 1){
        cout << n << "\n";
        for(int c1 = 0; c1 < n; c1++){
            cout << c1+1 << " ";
        }cout << "\n";
        return 0;
    }

    dijkstra(special[0]);
    start1 = 0;
    ll dmax = 0;
    for(int c1 = 0; c1 < sz(special); c1++){
        if(DIST[special[c1]] > dmax){
            dmax = DIST[special[c1]];
            start1 = special[c1];
        }
    }
    solve(start1);
    for(int c1 = 0; c1 < n; c1++){
        if(is_special[ind[c1]])start2 = ind[c1];
    }
    solve(start2);

    ll ans = 0;
    for(int c1 = 0; c1 < n; c1++){
        ans += ANS[c1];
    }
    cout << ans << "\n";
    int counter = 0;
    for(int c1 = 0; c1 < n; c1++){
        if(ANS[c1]){
            if (counter++) cout.put(' ');
            cout << c1+1;
        }
    }
    cout << "\n";
    return 0;
}
```

Bài 4. Tập con không chia hết cho K

Tính chất đồng dư modunlo, đếm phân phối.

Bài 5. Khoảng cách lớn nhất



Code tham khảo:

```
#include <iostream>
#include <cstdio>
#include <limits>
#include <algorithm>
using namespace std;
#define taskname "MAXDIS"
typedef long long lli;
typedef unsigned long long llu;
const int maxN = 100000;

int n;
lli L, R, a[maxN];

inline llu Distance(lli a, lli b)
{
   return b - a;
```

```
}

inline lli Center(lli a, lli b)
{
    llu d = Distance(a, b);
    return b - (d >> 1);
}

inline bool Maximize(llu &target, llu value)
{
    if (target >= value) return false;
    target = value;
    return true;
}

void Enter()
{
    cin >> n >> L >> R;
    for (int i = 0; i < n; i++)
        cin >> a[i];
    sort(a, a + n);
}

lli Solve()
{
    llu resL = 0;
    lli x;
    if (a[n - 1] <= R)
    {
        x = R;
        resL = Distance(a[n - 1], R);
    }
    for (int i = n - 1; i > 0; i--)
    {
        if (a[i - 1] > R) continue;
        if (a[i] < L) break;
        lli y = Center(a[i - 1], a[i]);
        if (y < L) y = L;
        if (y > R) y = R;
        llu dis1 = Distance(a[i - 1], y);
        llu dis2 = Distance(y, a[i]);
        if (dis2 < dis1) dis1 = dis2;
        if (Maximize(resL, dis1))
```

```
            x = y;
        }
        if (L <= a[0] && Maximize(resL, Distance(L, a[0]))) x = L;
        return x;
    }

    int main()
    {
        ios_base::sync_with_stdio(false);
        cin.tie(NULL);
        Enter();
        cout << Solve();
    }
```

Bài 6. Vượt đèo