Miscellaneous

Ordered statistic multi-set

```
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds ;
using namespace std ;
typedef tree <pair<int, int> , null_type , less<pair<int, int>>, rb_tree_tag ,
tree_order_statistics_node_update> ordered_set ;

const int inf = 1e9;

struct multi_orderset {
    ordered_set st;
    map<int, int> mp;
    void insert(int val) {
        st.insert(make_pair(val, mp[val] + 1));
        mp[val]++;
    }

    void remove(int val) {
        if (mp[val] > 0) {
            st.erase(make_pair(val, mp[val]));
            mp[val]--;
        }
    }

    int count(int val) {
        return mp[val];
    }

    bool contains(int val) {
        if (mp[val] != 0) return true;
        return false;
    }

    // return how many numbers are strictly smaller than val
    int order_of_key(int val) {
        return st.order_of_key(make_pair(val, -1));
    }
    // 0-indexed
    int findkth(int k) {
        if ((int) st.size() - 1 < k) return inf;
        return st.find_by_order(k)->first;
    }
};

multi_orderset myset;
```

System of different constraints

```
#pragma GCC optimize(3)
#pragma GCC optimize(2)
#include <bits/stdc++.h>
#define MAX_SIZE 1011
using namespace std;

const int inf = 1e7;

struct edge {
    int from, to, w;
};

pair<int, int> oedg[MAX_SIZE << 3];
int ans[MAX_SIZE << 3];
edge edg[MAX_SIZE << 3];
// the graph and reverse graph
vector<int> g[MAX_SIZE];
vector<int> rg[MAX_SIZE];
// on the path from 1 to n, on the reverse path from n to 1
int onpath1[MAX_SIZE], onpathn[MAX_SIZE];
// useful iff onpath1[i] and onpathn[i]
int useful[MAX_SIZE];
int n, m;
int tol = 0;
int dist[MAX_SIZE];

void dfs1(int v) {
    onpath1[v] = 1;
    for (auto nv : g[v]) {
        if (!onpath1[nv]) {
            dfs1(nv);
        }
    }
}

void dfs2(int v) {
    onpathn[v] = 1;
    for (auto nv : rg[v]) {
```

```cpp
            if (!onpathn[nv]) {
                dfs2(nv);
            }
        }
}

bool bellman_ford() {
    dist[0] = 0;
    int i, j;
    for (i = 1; i <= n; ++i) {
        dist[i] = inf;
    }

    for (i = 0; i < n; ++i) {
        for (j = 1; j <= tol; ++j) {
            edge &e = edg[j];
            dist[e.to] = min(dist[e.from] + e.w, dist[e.to]);
        }
    }

    for (j = 1; j <= tol; ++j) {
        edge &e = edg[j];
        if (dist[e.to] > dist[e.from] + e.w) {
            return false;
        }
    }

    return true;
}

int main() {
    int i;
    scanf("%d%d", &n, &m);
    for (i = 1; i <= m; ++i) {
        scanf("%d%d", &oedg[i].first, &oedg[i].second);
        g[oedg[i].first].push_back(oedg[i].second);
        rg[oedg[i].second].push_back(oedg[i].first);
    }

    dfs1(1);
    dfs2(n);

    for (i = 1; i <= n; ++i) {
        useful[i] = onpath1[i] && onpathn[i];
    }

    for (i = 1; i <= m; ++i) {
        if (useful[oedg[i].first] && useful[oedg[i].second]) {
            // here add edge for system of different constraints
            int from = oedg[i].first, to = oedg[i].second;
            // we know that 1 <= dist[to] - dist[from] <= 2
            // it is  dist[to] - dist[from] <= 2 and dist[from] - dist[to] <=
-1
            edg[++tol] = edge{from, to, 2};
            edg[++tol] = edge{to, from, -1};
        }
    }

    for (i = 1; i <= n; ++i) {
        if (useful[i]) {
            edg[++tol] = edge{i, 0, 0};
        }
    }

    if (!bellman_ford()) {
        printf("No\n");
        return 0;
    }

    printf("Yes\n");
    for (i = 1; i <= m; ++i) {
        if (useful[oedg[i].first] && useful[oedg[i].second]) {
            printf("%d\n", dist[oedg[i].second] - dist[oedg[i].first]);
        } else {
            printf("1\n");
        }
    }
    return 0;
}
```

SG function

```
#pragma GCC optimize(3)
#pragma GCC optimize(2)
#include <bits/stdc++.h>
#define DEBUG(x) std::cerr << #x << '=' << x << std::endl
using namespace std;
typedef long long ll;
typedef pair<int , int> ii;

int T , R , C;
int DP[20][20][20][20];
int SG[20][20][20][20];
char g[20][20];

/*
```

```
  when placing either H or V , we can divide our current board into 2 seperate
 boards
  thus, creating 2 individual games
  To check if we can win both these 2 individual games, we just need to take t
he xor of
  the SG value of these 2 games. We can win iff the xor sum is 0.
  The SG value of current game state is mex of SG values of all possible reach
able game states
*/

int mex(unordered_map<int , int>& vis){
  int ret = 0;
  for(int i = 0; i <= vis.size(); ++i){
    if(vis[i] == 0){
      ret = i;
      break;
    }
  }
  return ret;
}

int check(int x1 , int y1 , int x2 , int y2){
  for(int i = x1; i <= x2; ++i){
    for(int j = y1; j <= y2; ++j){
      if(g[i][j] == '#')return 0;
    }
  }
  return 1;
}

int solve(int x1 , int y1 , int x2 , int y2){
  if(x1 > x2 || y1 > y2 || x1 < 1 || y1 < 1 || x2 > R || y2 > C)return 0; //lo
st
  int& ret = SG[x1][y1][x2][y2];
  if(~ret)return ret; // already computed
  unordered_map<int , int> vis;
  for(int i = x1; i <= x2; ++i){
    for(int j = y1; j <= y2; ++j){
      if(g[i][j] == '#')continue;
      if(check(i , y1 , i , y2)){ // can place H at (i , j)
        int sg1 = solve(x1 , y1 , i - 1 , y2);
        int sg2 = solve(i + 1 , y1 , x2 , y2);
        if((sg1 ^ sg2) == 0)++DP[x1][y1][x2][y2]; // means this is a winning m
ove
        ++vis[sg1 ^ sg2];
      }
      if(check(x1 , j , x2 , j)){ // can place V at (i , j)
        int sg1 = solve(x1 , y1 , x2 , j - 1);
        int sg2 = solve(x1 , j + 1 , x2 , y2);
        if((sg1 ^ sg2) == 0)++DP[x1][y1][x2][y2];
        ++vis[sg1 ^ sg2];
      }
    }
  }
  return ret = mex(vis);
}

void answer(){
  for(int i = 1; i <= R; ++i)
    for(int j = 1; j <= C; ++j)
      cin >> g[i][j];
  memset(DP , 0 , sizeof(DP));
  memset(SG , -1 , sizeof(SG));
  solve(1 , 1 , R , C);
  cout << DP[1][1][R][C] << endl;
}

int main(){
  ios::sync_with_stdio(false);
  cin.tie(nullptr);
  cin >> T;
  for(int t = 1; t <= T; ++t){
    cin >> R >> C;
    cout << "Case #" << t << ": ";
    answer();
  }
}
```

Range tree graph construction

```
#include <bits/stdc++.h>
#define LEFT (index<<1)
#define RIGHT (index<<1|1)
#define MID ((l+r)>>1)
#define MAX_SIZE 524292
using namespace std;

int f[MAX_SIZE];
int idx[MAX_SIZE >> 1];
int visited[MAX_SIZE];

int find(int x) {
    if (x == f[x]) return x;
    return f[x] = find(f[x]);
}

void unionset(int x, int y) {
```

```cpp
    int a = find(x), b = find(y);
    f[a] = b;
}

void build(int l, int r, int index) {
    if (l > r) return;
    f[index] = index;
    if (l == r) {
        idx[l] = index;
        return;
    }

    build(l, MID, LEFT);
    build(MID + 1, r, RIGHT);
}

void pushdown(int l, int r, int index) {
    if (l > r || visited[index]) return;
    visited[index] = true;
    if (l == r) return;
    unionset(index, LEFT);
    unionset(index, RIGHT);

    pushdown(l, MID, LEFT);
    pushdown(MID + 1, r, RIGHT);
}

vector<int> g[MAX_SIZE >> 1];

void update(int start, int end, int l, int r, int index, int pt) {
    if (start > end || l > r) return;
    if (start <= l && r <= end) {
        if (find(idx[pt]) != find(index)) {
            for (int i = l; i <= r; ++i) {
                if (find(idx[pt]) != find(idx[i])) {
                    g[pt].push_back(i);
                    g[i].push_back(pt);
                    unionset(idx[pt], idx[i]);
                }
            }
        }
        unionset(idx[pt], index);
        pushdown(l, r, index);
        return;
    }

    if (end <= MID) {
        update(start, end, l, MID, LEFT, pt);
    } else if (start >= MID + 1) {
        update(start, end, MID + 1, r, RIGHT, pt);
    } else {
        update(start, end, l, MID, LEFT, pt);
        update(start, end, MID + 1, r, RIGHT, pt);
    }
}

void init() {
    int i;
    for (i = 0 ; i < MAX_SIZE; ++i) {
        f[i] = i, visited[i] = 0;
    }
}

int N;
vector<int> add[MAX_SIZE >> 1], del[MAX_SIZE >> 1];
char ch[4];
int color[MAX_SIZE];

void dfs(int v, int pre, int d) {
    color[v] = d % 2 == 0 ? 2 : 1;
    for (auto nv : g[v]) {
        if (nv != pre) {
            dfs(nv, v, d + 1);
        }
    }
}

int main() {
    int i, j;
    init();
    scanf("%d", &N);
    for (i = 1; i <= N; ++i) {
        int d;
        scanf("%s%d", ch, &d);
        if (ch[0] == 'C') {
            while (d-- > 0) {
                int v;
                scanf("%d", &v);
                add[i].push_back(v);
            }
        } else {
            while (d-- > 0) {
                int v;
                scanf("%d", &v);
                del[i].push_back(v);
            }
            del[i].push_back(i);
        }
    }

    build(1, N, 1);
    for (i = 1; i <= N; ++i) {
        bool hasone = false;
        sort(add[i].begin(), add[i].end());
        sort(del[i].begin(), del[i].end());
        if (del[i].empty()) {
            for (auto v : add[i]) {
                hasone = true;
                update(v, v, 1, N, 1, i);
            }
        } else {
            for (j = 0 ; j < (int) del[i].size(); ++j) {
                if (j == 0) {
                    if (del[i][j] - 1 >= 1) {
                        update(1, del[i][j] - 1, 1, N, 1, i);
                        hasone = true;
                    }
                } else {
                    if (del[i][j] - 1 >= del[i][j-1] + 1) {
                        update(del[i][j-1] + 1, del[i][j] - 1, 1, N, 1, i);
                        hasone = true;
                    }
                }
            }

            if (del[i].back() + 1 <= N) {
                update(del[i].back() + 1, N, 1, N, 1, i);
                hasone = true;
            }
        }

        if (!hasone) {
            printf("Impossible");
            return 0;
        }
    }

    for (i = 1; i <= N; ++i) {
        if (color[i] == 0) {
            dfs(i, 0, 0);
        }
    }

    for (i = 1; i <= N; ++i) {
        if (color[i] == 1) {
            printf("S");
        } else {
            printf("V");
        }
    }
    return 0;
}
```

Maxflow with double

```
#include <bits/stdc++.h>
#define MAX_SIZE 3011
typedef long long ll;
#define INF 1000000000000000
using namespace std;
const ll NOFLOW = 0;

struct Maxflow {
    struct edge {
        int from, to;
        ll flow, capacity;
    };

    // start and end point
    int s, t;

    // list array
    vector<edge> edg;

    // g reference to the ith vertex's edges
    vector<int> g[MAX_SIZE];

    // distance array and visited array
    int dist[MAX_SIZE], visited[MAX_SIZE];
    int cur[MAX_SIZE];

    void init() {
        edg.clear();
        int i;
        for (i = 0 ; i < MAX_SIZE; i++) {
            g[i].clear();
        }
    }

    void addedge(int from, int to, ll capacity) {
        edge e1 = edge{from, to, 0ll, capacity};
        edge e2 = edge{to, from, 0ll, 0ll};
        edg.push_back(e1), edg.push_back(e2);
        g[from].push_back((int) edg.size() - 2);
        g[to].push_back((int) edg.size() - 1);
    }

    // construct the level graph
    bool bfs() {
        memset(visited,0,sizeof(visited));
        memset(dist,0,sizeof(dist));
        queue<int> q;
        q.push(s);
        visited[s] = 1;
        dist[s] = 0;
        while (!q.empty()) {
            int v = q.front();
            q.pop();
            for (int i = 0 ; i < (int) g[v].size(); i++) {
                edge &e = edg[g[v][i]];
                int nxt = e.to;
                if (!visited[nxt] && e.capacity > e.flow + NOFLOW) {
                    dist[nxt] = dist[v] + 1;
                    q.push(nxt);
                    visited[nxt] = 1;
                }
            }
        }

        return visited[t];
    }

    ll dfs(int x, ll cp) {
        if (x == t || cp == 0) {
            return cp;
        }

        ll flow = 0, newflow;
        for (int &y = cur[x]; y < (int) g[x].size(); y++) {
            edge &e = edg[g[x][y]];
            if (dist[x] + 1 == dist[e.to]) {
                ll minn = min(cp, e.capacity - e.flow);
                newflow = dfs(e.to, minn);
                if (newflow > NOFLOW) {
                    e.flow += newflow;
                    edg[g[x][y] ^1].flow -= newflow;
                    flow += newflow;
                    cp -= newflow;

                    if (cp <= NOFLOW) {
                        break;
                    }
                }
            }
        }

        return flow;
    }

    ll Dinic(){
        ll flow=0;
        while(bfs()){
            memset(cur,0,sizeof(cur));
            flow += dfs(s,INF);
        }
        return flow;
    }

    set<int> nd;

    void reachs(int src) {
        visited[src] = 1;
        nd.insert(src);
        for (int i = 0; i < (int) g[src].size(); i++) {
            edge e = edg[g[src][i]];
            if (!visited[e.to] && e.flow + NOFLOW < e.capacity) {
                reachs(e.to);
            }
        }
    }

    void couldt(int src) {
        visited[src] = 1;
        nd.insert(src);
        // cout << src << endl;
        for (int i = 0; i < (int) g[src].size(); i++) {
            edge e = edg[g[src][i]];
            if (!visited[e.to] && edg[g[src][i]^1].flow + NOFLOW < edg[g[src][
i]^1].capacity) {
                couldt(e.to);
            }
        }
    }
};

Maxflow mf;
```

All edges which are from a reachable vertex to non-reachable vertex are minimum **cut** edges.

Permutation

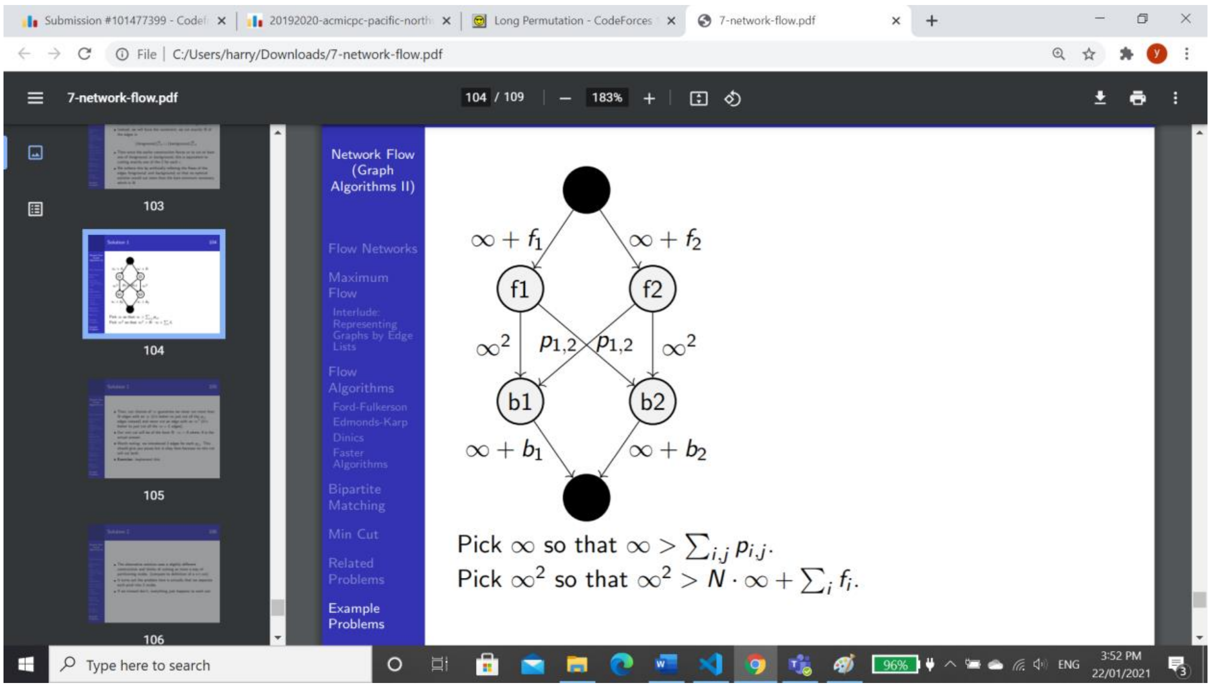
Find the ord-th permutation.

```
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
#define MAX_SIZE 200011
typedef long long ll;
using namespace std;
using namespace __gnu_pbds ;
typedef tree <int , null_type , less <int>, rb_tree_tag ,
tree_order_statistics_node_update> ordered_set ;

ordered_set st;
ll pre[MAX_SIZE], ord = 0, fac[17];
int a[MAX_SIZE], p[MAX_SIZE], used[17];
int N, M, Q;

void permute() {
    int i;
    ll x = ord;
    st.clear();
    for (i = 1; i <= M; ++i) st.insert(i);
    for(i = 1; i <= M; ++i) {
        int t = x / fac[M-i] + 1;
        // cout << t << endl;
        int k = *st.find_by_order(t - 1);
        st.erase(k);
        p[i] = k;
        x %= fac[M-i];
    }
}



void permute() {
    int i, j;
    ll x = ord;
    memset(used, 0, sizeof(used));
    for(i = 1; i <= M; ++i) {
        int t = x / fac[M-i] + 1, k = 0;
        for(j = 1; j <= M; ++j) {
            if(!used[j] && ++k==t) {
                k = j;
                break;
            }
        }
        used[k] = 1;
        p[i] = k;
        x %= fac[M-i];
    }
}
```

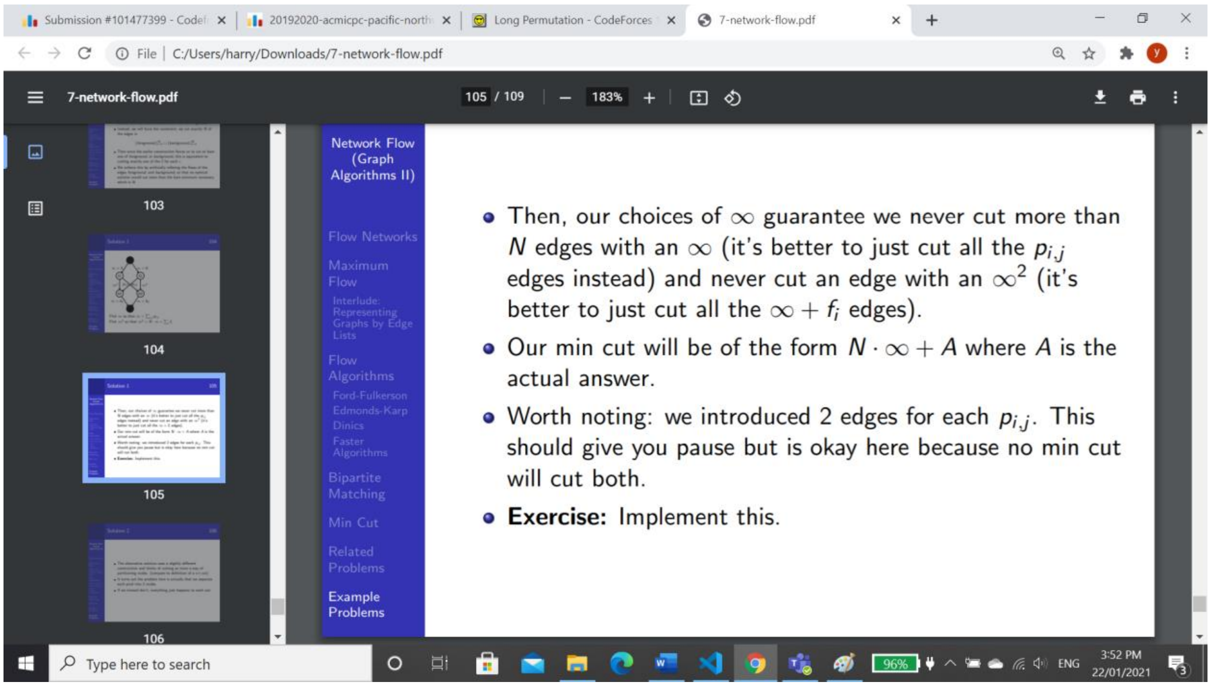
```
static const int FAC[] = {1, 1, 2, 6, 24, 120, 720, 5040, 40320, 362880};   // 阶乘
int cantor(int *a, int n)
{
    int x = 0;
    for (int i = 0; i < n; ++i) {
        int smaller = 0;  // 在当前位之后小于其的个数
        for (int j = i + 1; j < n; ++j) {
            if (a[j] < a[i])
                smaller++;
        }
        x += FAC[n - i - 1] * smaller; // 康托展开累加
    }
    return x;  // 康托展开值
}
```

Image segmentation

Problem Statement: I have an image made of N pixels. Each can be either in the background or foreground. For the ith pixel, you get fi points if it is in the foreground, bi points if it is in the background. Furthermore, there are M pairs of pixels that you would prefer to have the same assignment. For the kth pair, you pay a penalty of pk if pixel ak has a different assignment to pixel bk . Maximize points - penalties. Input Format: First line, 2 integers N M. 1 ≤ N, M ≤ 1000. Next line contains N integers, the values fi . Next line contains N integers, the values bi . Next M lines each contain a triplet, ak bk pk, describing one of the penalty pairs.



Pick ∞ so that $\infty > \sum_{i,j} p_{i,j}$.
Pick ∞^2 so that $\infty^2 > N \cdot \infty + \sum_i f_i$.



- Then, our choices of ∞ guarantee we never cut more than N edges with an ∞ (it's better to just cut all the $p_{i,j}$ edges instead) and never cut an edge with an ∞^2 (it's better to just cut all the $\infty + f_i$ edges).
- Our min cut will be of the form $N \cdot \infty + A$ where A is the actual answer.
- Worth noting: we introduced 2 edges for each $p_{i,j}$. This should give you pause but is okay here because no min cut will cut both.
- **Exercise:** Implement this.

MCMF model

The good times at Heidi's library are over. Marmots finally got their internet connections and stopped coming to the library altogether. Not only that, but the bookstore has begun charging extortionate prices for some books. Namely, whereas in the previous versions each book could be bought for 1 CHF, now the price of book *i* is c_i CHF.

I had some initial observations on this problem, we can firstly purchase all the books in and minus the merging cost (i.e. the book of the same type that is required on day d1 and d2 can be calculated only once). But what happened next to this problem is stunning. The flow graph works like this, each vertex is splitted into 2 nodes i and i'. Then we add edges (1, cost) from src to i and edges (1, 0) from i' to sink. Then, we add edges from i to i' (1, 0). To make sure each day there's only k books in hand, we can add edges (k-1, 0) from i to i+1. And to deal with the merging cost condition, we can add edges from i-1 to j' (1, -cost[tp[i]]). Here j' is the outgoing vertices such that tp[i] = tp[j] and j is the maximum of those nodes (j < i). The MCMF would be the answer. The key lesson for this problem is how to model the k books in hand condition.

```
MCMF mf;
int tp[maxn], cost[maxn];
int pre[maxn];

int main() {
    int n, k, i;
    mf.init();
    mf.src = 0, mf.target = maxn - 1;
    scanf("%d%d", &n, &k);
    for (i = 1; i <= n; ++i) {
        scanf("%d", &tp[i]);
    }

    for (i = 1; i <= n; ++i) {
        scanf("%d", &cost[i]);
    }

    for (i = 1; i <= n; ++i) {
        mf.addedge(mf.src, i, 1, cost[tp[i]]);
        mf.addedge(i, i + n, 1, 0);
        mf.addedge(i + n, mf.target, 1, 0);
        if (i < n) mf.addedge(i, i + 1, k - 1, 0);
    }

    for (i = 1; i <= n; ++i) {
        if (pre[tp[i]]) {
            mf.addedge(i - 1, pre[tp[i]] + n, 1, -cost[tp[i]]);
        }
        pre[tp[i]] = i;
    }

    ll cost = 0;
    mf.mincostMaxflow(cost);
    cout << cost << endl;
    return 0;
}
```