Standard Code Library
(for Macau Regional)

Contents

1 数学

1.1 Berlekamp-Massey最小递推式

如果要求出一个次数为k的递推式, 则输入的数列需要至少有$2k$项.
返回的内容满足$\sum_{j=0}^{m-1} a_{i-j}c_j = 0$, 并且$c_0 = 1$. 称为最小递推式.
如果不加最后的处理的话, 代码返回的结果会变成$a_i = \sum_{j=0}^{m-1} c_{j-1}a_{i-j}$, 有时候这样会方便接着跑递推, 需要的话就删掉最后的处理.
(实际上Berlekamp-Massey是对每个前缀都求出了递推式, 但一般没啥用.)

```cpp
vector<int> berlekamp_massey(const vector<int> &a) {
    vector<int> v, last; // v is the answer, 0-based
    int k = -1, delta = 0;

    for (int i = 0; i < (int)a.size(); i++) {

        int tmp = 0;
        for (int j = 0; j < (int)v.size(); j++)
            tmp = (tmp + (long long)a[i - j - 1] *
              ↪ v[j]) % p;

        if (a[i] == tmp)
            continue;

        if (k < 0) {
            k = i;
            delta = (a[i] - tmp + p) % p;
            v = vector<int>(i + 1);

            continue;
        }

        vector<int> u = v;
        int val = (long long)(a[i] - tmp + p) *
          ↪ qpow(delta, p - 2) % p;

        if (v.size() < last.size() + i - k)
            v.resize(last.size() + i - k);

        (v[i - k - 1] += val) %= p;

        for (int j = 0; j < (int)last.size(); j++) {
            v[i - k + j] = (v[i - k + j] - (long
              ↪ long)val * last[j]) % p;
            if (v[i - k + j] < 0)
                v[i - k + j] += p;
        }

        if ((int)u.size() - i < (int)last.size() - k)
          ↪ {
            last = u;
            k = i;
            delta = a[i] - tmp;
            if (delta < 0)
                delta += p;
        }
    }

    for (auto &x : v) // 一般是需要最小递推式的, 所以处
      ↪ 理一下
        x = (p - x) % p;
    v.insert(v.begin(), 1);

    return v; // ∀i, sum_{j=0}^{m} a_{i-j}v_j = 0
}
```

如果要求向量序列的递推式, 就把每位乘一个随机权值(或者说是乘一个随机行向量v^T)变成求数列递推式即可.
如果是矩阵序列的话就随机一个行向量u^T和列向量v, 然后把矩阵变成$u^T Av$的数列就行了.

1.1.1 优化矩阵快速幂DP

如果f_i是一个向量, 并且转移是一个矩阵, 那显然$\{f_i\}$是一个线性递推序列.
假设f_i有n维, 先暴力求出$f_{0\sim 2n-1}$, 然后跑Berlekamp-Massey, 最后调用前面的快速齐次线性递推(**??**页)即可. (快速齐次线性递推的结果是一个序列, 某个给定初值的结果就是点乘, 所以只需要跑一次.)
如果要求f_m, 并且矩阵有k个非零项的话, 复杂度就是$O(nk + n \log m \log n)$. (因为暴力求前$2n-1$个f_i需要$O(nk)$时间.)

1.1.2 求矩阵最小多项式

矩阵A的最小多项式是次数最小的并且$f(A) = 0$的多项式f.
实际上最小多项式就是$\{A^i\}$的最小递推式, 所以直接调用Berlekamp-Massey就好了, 并且显然它的次数不超过n.
瓶颈在于求出A^i, 实际上我们只要处理$A^i v$就行了, 每次对向量做递推.
假设A有k个非零项, 则复杂度为$O(kn + n^2)$.

1.1.3 求稀疏矩阵的行列式

如果能求出特征多项式, 则常数项乘上$(-1)^n$就是行列式, 但是最小多项式不一定就是特征多项式.
把A乘上一个随机对角阵B(实际上就是每行分别乘一个随机数), 则AB的最小多项式有很大概率就是特征多项式, 最后再除掉$\det B$就行了.
设A有k个非零项, 则复杂度为$O(kn + n^2)$.

1.1.4 求稀疏矩阵的秩

设A是一个$n \times m$的矩阵, 首先随机一个$n \times n$的对角阵P和一个$m \times m$的对角阵Q, 然后计算$QAPA^TQ$ 的最小多项式即可.
实际上不用计算这个矩阵, 因为求最小多项式时要用它乘一个向量, 我们依次把这几个矩阵乘到向量里就行了. 答案就是最小多项式除掉所有x因子后剩下的次数.
设A有k个非零项, 复杂度为$O(kn + n^2)$.

1.1.5 解稀疏方程组

问题 $Ax = b$, 其中A是一个$n \times n$的**满秩**稀疏矩阵, b和x是$1 \times n$的**列**向量, A, b已知, 需要解出x.

做法 显然$x = A^{-1}b$. 如果我们能求出$\{A^i b\}(i \geq 0)$的最小递推式$\{r_{0\sim m-1}\}(m \leq n)$, 那么就有结论

$$A^{-1}b = -\frac{1}{r_{m-1}} \sum_{i=0}^{m-2} A^i b r_{m-2-i}$$

因为A是稀疏矩阵, 直接按定义递推出$b \sim A^{2n-1}b$即可. 设A中有k个非零项, 则复杂度为$O(kn + n^2)$.

```cpp
vector<int> solve_sparse_equations(const
  ↪ vector<tuple<int, int, int> > &A, const
  ↪ vector<int> &b) {
    int n = (int)b.size(); // 0-based

    vector<vector<int> > f({b});

    for (int i = 1; i < 2 * n; i++) {
        vector<int> v(n);
        auto &u = f.back();
```

```
        for (auto [x, y, z] : A) // [x, y, value]
            v[x] = (v[x] + (long long)u[y] * z) % p;

        f.push_back(v);
    }

    vector<int> w(n);
    mt19937 gen;
    for (auto &x : w)
        x = uniform_int_distribution<int>(1, p - 1)
          ↪ (gen);

    vector<int> a(2 * n);
    for (int i = 0; i < 2 * n; i++)
        for (int j = 0; j < n; j++)
            a[i] = (a[i] + (long long)f[i][j] * w[j])
              ↪ % p;

    auto c = berlekamp_massey(a);
    int m = (int)c.size();

    vector<int> ans(n);

    for (int i = 0; i < m - 1; i++)
        for (int j = 0; j < n; j++)
            ans[j] = (ans[j] + (long long)c[m - 2 - i]
              ↪ * f[i][j]) % p;

    int inv = qpow(p - c[m - 1], p - 2);

    for (int i = 0; i < n; i++)
        ans[i] = (long long)ans[i] * inv % p;

    return ans;
}
```

2 数论

3 图论

3.1 最小直径生成树

首先要找到图的绝对中心(可能在点上, 也可能在某条边上), 然后以绝对中心为起点建最短路树就是最小直径生成树.

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

using namespace std;

constexpr int maxn = 505;
constexpr long long inf = 0x3f3f3f3f3f3f3f3fll;

int g[maxn][maxn], id[maxn][maxn], pr[maxn]; // g是邻接
  ↪ 矩阵
long long f[maxn][maxn], d[maxn];
bool vis[maxn];

vector<pair<int, int>>
  ↪ minimum_diameter_spanning_tree(int n) { // 1-based
    for (int i = 1; i ≤ n; i++)
        for (int j = 1; j ≤ n; j++)
            g[i][j] *= 2; // 输入的边权都要乘2

    memset(f, 63, sizeof(f));

    for (int i = 1; i ≤ n; i++)
        f[i][i] = 0;

    for (int i = 1; i ≤ n; i++)
        for (int j = 1; j ≤ n; j++)
            if (g[i][j])
                f[i][j] = g[i][j];

    for (int k = 1; k ≤ n; k++)
        for (int i = 1; i ≤ n; i++)
            for (int j = 1; j ≤ n; j++)
                f[i][j] = min(f[i][j], f[i][k] + f[k]
                  ↪ [j]);

    for (int i = 1; i ≤ n; i++) {
        for (int j = 1; j ≤ n; j++)
            id[i][j] = j; // 距离i第j近的点

        sort(id[i] + 1, id[i] + n + 1, [&i] (int x,
          ↪ int y) {
            return f[i][x] < f[i][y];
        });
    }

    int o = 0;
    long long ansv = inf; // vertex

    for (int i = 1; i ≤ n; i++)
        if (f[i][id[i][n]] * 2 < ansv) {
            ansv = f[i][id[i][n]] * 2;
            o = i;
        }

    int u = 0, v = 0;
    long long disu = -inf, disv = -inf, anse = inf;

    for (int x = 1; x ≤ n; x++)
        for (int y = 1; y ≤ n; y++)
            if (g[x][y]) { // 如果g[x][y] = 0说明没有边
                int w = g[x][y];

                for (int i = n - 1, j = n; i; i--)
                    if (f[y][id[x][i]] > f[y][id[x]
                      ↪ [j]]) {
                        long long tmp = f[x][id[x][i]]
                          ↪ + f[y][id[x][j]] + w;
                        if (tmp < anse) {
                            anse = tmp;
                            u = x;
                            v = y;

                            disu = tmp / 2 - f[x]
                              ↪ [id[x][i]];
                            disv = w - disu;
                        }
                        j = i;
                    }
            }

    printf("%lld\n", min(ansv, anse) / 2); // 直径

    memset(d, 63, sizeof(d));

    if (ansv ≤ anse)
        d[o] = 0;
    else {
        d[u] = disu;
        d[v] = disv;
    }
```

```
    for (int k = 1; k ≤ n; k++) { // Dijkstra
        int x = 0;
        for (int i = 1; i ≤ n; i++)
            if (!vis[i] && d[i] < d[x])
                x = i;

        vis[x] = true;
        for (int y = 1; y ≤ n; y++)
            if (g[x][y] && !vis[y]) {
                if (d[y] > d[x] + g[x][y]) {
                    d[y] = d[x] + g[x][y];
                    pr[y] = x;
                }
                else if (d[y] == d[x] + g[x][y] &&
                  ↪ d[pr[y]] < d[x])
                    pr[y] = x;
            }
    }

    vector<pair<int, int>> vec;
    for (int i = 1; i ≤ n; i++)
        if (pr[i])
            vec.emplace_back(i, pr[i]);

    if (ansv > anse)
        vec.emplace_back(u, v);

    return vec;
}

int main() {

    int n, m;
    scanf("%d%d", &n, &m);

    while (m--) {
        int x, y, z;
        scanf("%d%d%d", &x, &y, &z);

        g[x][y] = g[y][x] = z; // 无向图
    }

    auto vec = minimum_diameter_spanning_tree(n);
    for (auto [x, y] : vec)
        printf("%d %d\n", x, y);

    return 0;
}
```

3.2 欧拉回路

C[x]是记录每条边对应的编号的.
另外为了保证复杂度需要加当前弧优化.

```
vector<int> G[maxn], C[maxn], v[maxn];
int cur[maxn];
bool vis[maxn * 2];

vector<pair<int, int> > vec;

int d[maxn];

void dfs(int x) {
    bool bad = false;

    while (!bad) {
        bad = true;

        for (int &i = cur[x]; i < (int)G[x].size(); i+
          ↪ +)
            if (!vis[C[x][i]]) {
                vis[C[x][i]] = true;
                vec.emplace_back(x, i);
                x = G[x][i];
                bad = false;

                break;
            }
    }
}
```

3.3 预流推进费用流(可处理负环) $O(nm \log C)$

不是很懂什么原理, 待研究.

```
// Push-Relabel implementation of the cost-scaling
  ↪ algorithm
// Runs in O( <max_flow> * log(V * max_edge_cost)) =
  ↪ O( V^3 * log(V * C))
// Really fast in practice, 3e4 edges are fine.
// Operates on integers, costs are multiplied by N!!

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

// source: unknown
template<typename flow_t = int, typename cost_t = int>
struct mcSFlow {
    struct Edge {
        cost_t c;
        flow_t f;
        int to, rev;
        Edge(int _to, cost_t _c, flow_t _f, int _rev):
          ↪ c(_c), f(_f), to(_to), rev(_rev) {}
    };

    static constexpr cost_t INFCOST =
      ↪ numeric_limits<cost_t>::max() / 2;

    cost_t eps;
    int N, S, T;
    vector<vector<Edge>> G;
    vector<unsigned int> isq, cur;
    vector<flow_t> ex;
    vector<cost_t> h;

    mcSFlow(int _N, int _S, int _T): eps(0), N(_N),
      ↪ S(_S), T(_T), G(_N) {}

    void add_edge(int a, int b, flow_t cap, cost_t
      ↪ cost) {
        assert(cap ≥ 0);
        assert(a ≥ 0 && a < N && b ≥ 0 && b < N);

        if (a == b) {
            assert(cost ≥ 0);
            return;
        }

        cost *= N;
        eps = max(eps, abs(cost));
        G[a].emplace_back(b, cost, cap, G[b].size());
        G[b].emplace_back(a, -cost, 0, G[a].size() -
          ↪ 1);
    }

    void add_flow(Edge &e, flow_t f) {
        Edge &back = G[e.to][e.rev];
```

```
        if (!ex[e.to] && f)
            hs[h[e.to]].push_back(e.to);

        e.f -= f;
        ex[e.to] += f;
        back.f += f;
        ex[back.to] -= f;
    }

    vector<vector<int>> hs;
    vector<int> co;

    flow_t max_flow() {
        ex.assign(N, 0);
        h.assign(N, 0);
        hs.resize(2 * N);
        co.assign(2 * N, 0);
        cur.assign(N, 0);
        h[S] = N;
        ex[T] = 1;
        co[0] = N - 1;

        for (auto &e : G[S])
            add_flow(e, e.f);

        if (hs[0].size())
            for (int hi = 0; hi >= 0;) {
                int u = hs[hi].back();
                hs[hi].pop_back();

                while (ex[u] > 0) { // discharge u
                    if (cur[u] == G[u].size()) {
                        h[u] = 1e9;

                        for (unsigned int i = 0; i < G[u].size(); ++i) {
                            auto &e = G[u][i];

                            if (e.f && h[u] > h[e.to] + 1) {
                                h[u] = h[e.to] + 1, cur[u] = i;
                            }
                        }

                        if (++co[h[u]], !--co[hi] && hi < N)
                            for (int i = 0; i < N; ++i)
                                if (hi < h[i] && h[i] < N) {
                                    --co[h[i]];
                                    h[i] = N + 1;
                                }

                        hi = h[u];
                    }
                    else if (G[u][cur[u]].f && h[u] == h[G[u][cur[u]].to] + 1)
                        add_flow(G[u][cur[u]], min(ex[u], G[u][cur[u]].f));
                    else
                        ++cur[u];
                }

                while (hi >= 0 && hs[hi].empty())
                    --hi;
            }

        return -ex[S];
    }

    void push(Edge &e, flow_t amt) {
        if (e.f < amt)
            amt = e.f;

        e.f -= amt;
        ex[e.to] += amt;
        G[e.to][e.rev].f += amt;
        ex[G[e.to][e.rev].to] -= amt;
    }

    void relabel(int vertex) {
        cost_t newHeight = -INFCOST;

        for (unsigned int i = 0; i < G[vertex].size(); ++i) {
            Edge const &e = G[vertex][i];

            if (e.f && newHeight < h[e.to] - e.c) {
                newHeight = h[e.to] - e.c;
                cur[vertex] = i;
            }
        }

        h[vertex] = newHeight - eps;
    }

    static constexpr int scale = 2;

    pair<flow_t, cost_t> minCostMaxFlow() {
        cost_t retCost = 0;

        for (int i = 0; i < N; ++i)
            for (Edge &e : G[i])
                retCost += e.c * (e.f);

        //find max-flow
        flow_t retFlow = max_flow();
        h.assign(N, 0);
        ex.assign(N, 0);
        isq.assign(N, 0);
        cur.assign(N, 0);
        queue<int> q;

        for (; eps; eps >>= scale) {
            //refine
            fill(cur.begin(), cur.end(), 0);

            for (int i = 0; i < N; ++i)
                for (auto &e : G[i])
                    if (h[i] + e.c - h[e.to] < 0 && e.f)
                        push(e, e.f);

            for (int i = 0; i < N; ++i) {
                if (ex[i] > 0) {
                    q.push(i);
                    isq[i] = 1;
                }
            }

            // make flow feasible
            while (!q.empty()) {
                int u = q.front();
                q.pop();
```

```
                isq[u] = 0;

                while (ex[u] > 0) {
                    if (cur[u] == G[u].size())
                        relabel(u);

                    for (unsigned int &i = cur[u],
                      ↪ max_i = G[u].size(); i <
                      ↪ max_i; ++i) {
                        Edge &e = G[u][i];

                        if (h[u] + e.c - h[e.to] < 0)
                          ↪ {
                            push(e, ex[u]);

                            if (ex[e.to] > 0 &&
                              ↪ isq[e.to] == 0) {
                                q.push(e.to);
                                isq[e.to] = 1;
                            }

                            if (ex[u] == 0)
                                break;
                        }
                    }
                }

            if (eps > 1 && eps >> scale == 0)
                eps = 1 << scale;
        }

        for (int i = 0; i < N; ++i)
            for (Edge &e : G[i])
                retCost -= e.c * (e.f);

        return make_pair(retFlow, retCost / 2 / N);
    }

    flow_t getFlow(Edge const &e) {
        return G[e.to][e.rev].f;
    }
};

int main() {

    int n, m;
    scanf("%d%d", &n, &m);

    mcSFlow<long long, long long> mcmf(n, 0, n - 1);

    while (m--) {
        int x, y, z, w;
        scanf("%d%d%d%d", &x, &y, &z, &w);

        mcmf.add_edge(x - 1, y - 1, z, w);
    }

    auto [flow, cost] = mcmf.minCostMaxFlow();

    printf("%lld %lld\n", flow, cost);

    return 0;
}
```

3.4 网络流原理

3.4.1 最大流

- **判断一条边是否必定满流**

在残量网络中跑一遍Tarjan，如果某条满流边的两端处于同一SCC中则说明它不一定满流．（因为可以找出包含反向边的环，增广之后就不满流了．）

3.4.2 最小割

首先牢记最小割的定义：选权值和尽量小的一些边，使得删除这些边之后s无法到达t．

- **最小割输出一种方案**

在残量网络上从S开始floodfill，源点可达的记为S集，不可达的记为T，如果一条边的起点在S集而终点在T集，就将其加入最小割中．

- **最小割的可行边与必须边**
  - 可行边：满流，且残量网络上不存在u到v的路径，也就是u和v不在同一SCC中．（实际上也就是最大流必定满流的边．）
  - 必须边：满流，且残量网络上S可达u，v可达T．

- **字典序最小的最小割**

直接按字典序从小到大的顺序依次判断每条边能否在最小割中即可．

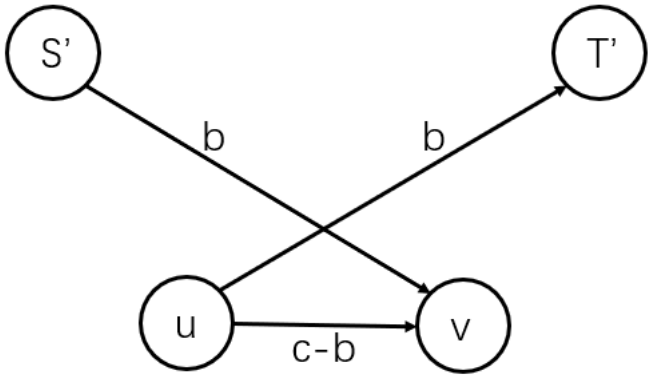
如果一条边是可行边，我们就需要把它删掉，同时进行退流，$u \to s$和$t \to v$都退掉等同于这条边容量的流量．

退流用Dinic实现即可．

3.4.3 上下界网络流

无源汇上下界可行流

新建源汇S', T'，然后如图所示转化每一条边．



在新图跑一遍最大流之后检查一遍辅助边，如果有辅助边没满流则无解，否则把每条边的流量加上b就是一组可行方案．

有源汇上下界最大流

如果不需要判断是否有解的话可以直接按照和上面一样的方法转化．因为附加边实际上算了两次流量，所以最终答案应该减掉所有下界之和．

（另外这里如果要压缩附加边的话，不能像无源汇的情况一样对每个点只开一个变量统计溢出的流量，正确的做法是进出流量各统计一下，每个点连两条附加边．）

如果需要判有解的话会出一点问题．这时候就需要转化成无源汇的情况，验证有解之后撤掉T到S的那条附加边再从S到T跑一遍最大流．

```
int ex[maxn], id[maxn];

int main() {

    memset(last, -1, sizeof(last));

    int n, m, src, sink;
```

```
    scanf("%d%d%d%d", &n, &m, &src, &sink);
    s = n + 1;
    t = n + 2;

    while (m--) {
        int x, y, b, c;
        scanf("%d%d%d%d", &x, &y, &b, &c);

        addedge(x, y, c - b);

        ex[y] += b;
        ex[x] -= b;
    }

    for (int i = 1; i ≤ n; i++) {
        id[i] = cnte;

        if (ex[i] ≥ 0)
            addedge(s, i, ex[i]);
        else
            addedge(i, t, -ex[i]);
    }

    addedge(sink, src, (~0u) >> 1);

    Dinic();

    if (any_of(id + 1, id + n + 1, [] (int i) {return
      ↪ (bool)e[i].cap;}))
        printf("please go home to sleep\n");
    else {
        int flow = e[cnte - 1].cap;
        e[cnte - 1].cap = e[cnte - 2].cap = 0;
        s = src;
        t = sink;

        printf("%d\n", flow + Dinic());
    }

    return 0;
}
```

有源汇上下界最小流

按照上面的方法转换后先跑一遍最大流，然后撤掉超级源汇和附加边，反过来跑一次最大流退流，最大流减去退掉的流量就是最小流.

3.4.4 常见建图方法

- **最大流/费用流**

流量不是很多的时候可以理解成很多条路径，并且每条边可以经过的次数有限.

- **最小割**

常用的模型是**最大权闭合子图**. 当然它并不是万能的，因为限制条件可以带权值.

1. 如果某些点全部在S集或者T集则获得一个正的收益

把这个条件建成一个点，向要求的点连∞边，然后s向它连∞边. (如果是T集就都反过来)

那么如果它在S集就一定满足它要求的点都在S集，反之如果是T集亦然.

2. 如果两个点不在同一集合中则需要付出代价

建双向边，那显然如果它们不在同一集合中就需要割掉中间的边，付出对应的代价.

3. 二分图，如果相邻的两个点在同一集合则需要付出代价

染色后给一半的点反转源汇，就转换成上面的问题了.

3.4.5 例题

- 费用流

1. 序列上选和尽量大的数，但连续k个数中最多选p个.

费用流建图，先建一条$n+1$个点的无限容量的链表示不选，然后每个点往后面k个位置连边，答案是流量为p的最大费用流. 因为条件等价于选p次并且每次选的所有数间隔都至少是k.

2. 还要求连续k个数中最少选q个.

任选一个位置把图前后切开就会发现通过截面的流量总和恰为p. 注意到如果走了最开始的链就代表不选，因此要限制至少有q 的流量不走链，那么只需要把链的容量改成$p-q$就行了.

3.5 Stoer-Wagner全局最小割

```
const int N = 601;
int fa[N], siz[N], edge[N][N];

int find(int x) {
    return fa[x] == x ? x : fa[x] = find(fa[x]);
}

int dist[N], vis[N], bin[N];
int n, m;

int contract(int& s, int& t) { // Find s,t
    memset(dist, 0, sizeof(dist));
    memset(vis, false, sizeof(vis));

    int i, j, k, mincut, maxc;

    for (i = 1; i ≤ n; i++) {
        k = -1;
        maxc = -1;

        for (j = 1; j ≤ n; j++)
            if (!bin[j] && !vis[j] && dist[j] > maxc)
              ↪ {
                k = j;
                maxc = dist[j];
            }

        if (k == -1)
            return mincut;

        s = t;
        t = k;
        mincut = maxc;
        vis[k] = true;

        for (j = 1; j ≤ n; j++)
            if (!bin[j] && !vis[j]) dist[j] += edge[k]
              ↪ [j];
    }

    return mincut;
}

const int inf = 0x3f3f3f3f;

int Stoer_Wagner() {
    int mincut, i, j, s, t, ans;
    for (mincut = inf, i = 1; i < n; i++) {
        ans = contract(s, t);
        bin[t] = true;

        if (mincut > ans)
            mincut = ans;
        if (mincut == 0)
```

```
            return 0;

        for (j = 1; j ≤ n; j++)
            if (!bin[j])
                edge[s][j] = (edge[j][s] += edge[j]
                  ↪ [t]);
    }

    return mincut;
}

int main() {
    cin >> n >> m;

    if (m < n - 1) {
        cout << 0;
        return 0;
    }

    for (int i = 1; i ≤ n; ++i)
        fa[i] = i, siz[i] = 1;

    for (int i = 1, u, v, w; i ≤ m; ++i) {
        cin >> u >> v >> w;

        int fu = find(u), fv = find(v);
        if (fu != fv) {
            if (siz[fu] > siz[fv]) swap(fu, fv);
            fa[fu] = fv, siz[fv] += siz[fu];
        }

        edge[u][v] += w, edge[v][u] += w;
    }

    int fr = find(1);

    if (siz[fr] != n) {
        cout << 0;
        return 0;
    }

    cout << Stoer_Wagner();

    return 0;
}
```

4 数据结构

4.1 历史和

EC-Final2020 G, 原题是询问某个区间有多少子区间, 满足子区间中数的种类数为奇数.
离线之后转化成枚举右端点并用线段树维护左端点, 然后就是一个支持区间反转(0/1互换)和询问历史和的线段树.
“既然标记会复合 就说明在两个标记中间没有经过任何 pushup 操作
也就是说一个这两个标记对应着 相同的 0 的数量 以及 相同的 1 的数量
那么标记对于答案的影响只能是 a * 0 + b * 1
我们维护 a b 即可”

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

using namespace std;

constexpr int maxn = (1 << 20) + 5;

int cnt[maxn][2], mul[maxn][2];
bool rev[maxn];
long long sum[maxn];

int now;

void build(int l, int r, int o) {
    cnt[o][0] = r - l + 1;

    if (l == r)
        return;

    int mid = (l + r) / 2;

    build(l, mid, o * 2);
    build(mid + 1, r, o * 2 + 1);
}

void apply(int o, bool flip, long long w0, long long
  ↪ w1) {
    sum[o] += w0 * cnt[o][0] + w1 * cnt[o][1];

    if (flip)
        swap(cnt[o][0], cnt[o][1]);

    if (rev[o])
        swap(w0, w1);

    mul[o][0] += w0;
    mul[o][1] += w1;
    rev[o] ^= flip;
}

void pushdown(int o) {
    if (!mul[o][0] && !mul[o][1] && !rev[o])
        return;

    apply(o * 2, rev[o], mul[o][0], mul[o][1]);
    apply(o * 2 + 1, rev[o], mul[o][0], mul[o][1]);

    mul[o][0] = mul[o][1] = 0;
    rev[o] = false;
}

void update(int o) {
    cnt[o][0] = cnt[o * 2][0] + cnt[o * 2 + 1][0];
    cnt[o][1] = cnt[o * 2][1] + cnt[o * 2 + 1][1];

    sum[o] = sum[o * 2] + sum[o * 2 + 1];
}

int s, t;

void modify(int l, int r, int o) {
    if (s ≤ l && t ≥ r) {
        apply(o, true, 0, 0);
        return;
    }

    int mid = (l + r) / 2;
    pushdown(o);

    if (s ≤ mid)
        modify(l, mid, o * 2);
    if (t > mid)
        modify(mid + 1, r, o * 2 + 1);

    update(o);
}
```

```cpp
long long query(int l, int r, int o) {
    if (s ≤ l && t ≥ r)
        return sum[o];

    int mid = (l + r) / 2;
    pushdown(o);

    long long ans = 0;
    if (s ≤ mid)
        ans += query(l, mid, o * 2);
    if (t > mid)
        ans += query(mid + 1, r, o * 2 + 1);

    return ans;
}

vector<pair<int, int> > vec[maxn]; // pos, id

long long ans[maxn];
int a[maxn], last[maxn];

int main() {

    int n;
    scanf("%d", &n);

    build(1, n, 1);

    for (int i = 1; i ≤ n; i++)
        scanf("%d", &a[i]);

    int m;
    scanf("%d", &m);

    for (int i = 1; i ≤ m; i++) {
        int l, r;
        scanf("%d%d", &l, &r);

        vec[r].emplace_back(l, i);
    }

    for (int i = 1; i ≤ n; i++) {
        s = last[a[i]] + 1;
        t = now = i;

        modify(1, n, 1);
        apply(1, false, 0, 1);

        for (auto [l, k] : vec[i]) {
            s = l;
            ans[k] = query(1, n, 1);
        }

        last[a[i]] = i;
    }

    for (int i = 1; i ≤ m; i++)
        printf("%lld\n", ans[i]);

    return 0;
}
```

4.2 二叉堆

```cpp
struct my_binary_heap {
    static constexpr int maxn = 100005;

    int a[maxn], size;

    my_binary_heap() : size(0) {}

    void push(int val) {
        a[++size] = val;

        for (int x = size; x > 1; x /= 2) {
            if (a[x] < a[x / 2])
                swap(a[x], a[x / 2]);
            else
                break;
        }
    }

    int &top() {
        return a[1];
    }

    int pop() {
        int res = a[1];
        a[1] = a[size--];

        for (int x = 1, son; ; x = son) {
            if (x * 2 == size)
                son = x * 2;
            else if (x * 2 > size)
                break;
            else if (a[x * 2] < a[x * 2 + 1])
                son = x * 2;
            else
                son = x * 2 + 1;

            if (a[son] < a[x])
                swap(a[x], a[son]);
            else
                break;
        }

        return res;
    }
};
```

5 字符串

5.1 区间本质不同子串计数(后缀自动机+LCT+线段树)

问题: 给定一个字符串s, 多次询问$[l,r]$区间的本质不同的子串个数, 可能强制在线.

做法: 考虑建出后缀自动机, 然后枚举右端点, 用线段树维护每个左端点的答案.

显然只有right集合在$[l,r]$中的串才有可能有贡献, 所以我们可以只考虑每个串最大的right.

每次右端点+1时找到它对应的结点u, 则u到根节点路径上的每个点, 它的right集合都会被r更新.

对于某个特定的左端点l, 我们需要保证本质不同的子串左端点不能越过它; 因此对于一个结点p, 我们知道它对应的子串长度$(val_{par_p}, val_p]$之后, 在p的right集合最大值减去对应长度, 这样对应的l内全部+1即可; 这样询问时就只需要查询r对应的线段树中$[l,r]$的区间和. (当然旧的right对应的区间也要减掉)

实际上可以发现更新时都是把路径分成若干个整段更新right集合, 因此可以用LCT维护这个过程.

时间复杂度$O(n\log^2 n)$, 空间$O(n)$, 当然如果强制在线的话, 就把线段树改成主席树, 空间复杂度就和时间复杂度同阶了.

```cpp
int tim; // tim实际上就是当前的右端点

```

```
node *access(node *x) {
    node *y = null;

    while (x != null) {
        splay(x);

        x -> ch[1] = null;
        x -> refresh();

        if (x -> val) // val记录的是上次访问时间，也就是right集合最大值
            update(x -> val - val[x -> r] + 1, x -> val - val[par[x -> l]], -1);

        x -> val = tim;
        x -> lazy = true;

        update(x -> val - val[x -> r] + 1, x -> val - val[par[x -> l]], 1);

        x -> ch[1] = y;

        (y = x) -> refresh();

        x = x -> p;
    }

    return y;
}

// 以下是main函数中的用法
for (int i = 1; i <= n; i++) {
    tim++;
    access(null + id[i]);

    if (i >= m) // 例题询问长度是固定的，如果不固定的话就按照右端点离线即可
        ans[i - m + 1] = query(i - m + 1, i);
}
```

还有一份完整的代码，因为写起来确实细节挺多的．这份代码支持在尾部加一个字符或者询问区间有多少子串至少出现了两次，并且强制在线．

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

using namespace std;

constexpr int maxn = 200005, maxm = maxn * 17 * 15;

int mx[maxm][2], lc[maxm], rc[maxm], seg_cnt;
int root[maxn];

int s, t, d;

void modify_seg(int l, int r, int &o) {
    int u = o;
    o = ++seg_cnt;

    mx[o][0] = max(mx[u][0], t);
    mx[o][1] = max(mx[u][1], d);

    if (l == r)
        return;

    lc[o] = lc[u];
    rc[o] = rc[u];

    int mid = (l + r) / 2;
    if (s <= mid)
        modify_seg(l, mid, lc[o]);
    else
        modify_seg(mid + 1, r, rc[o]);
}

int query_seg(int l, int r, int o, int k) {
    if (s <= l && t >= r)
        return mx[o][k];

    int mid = (l + r) / 2, ans = 0;

    if (s <= mid)
        ans = max(ans, query_seg(l, mid, lc[o], k));
    if (t > mid)
        ans = max(ans, query_seg(mid + 1, r, rc[o], k));

    return ans;
}

int N;

void modify(int pos, int u, int v, int &rt) {
    s = pos;
    t = u;
    d = v;

    modify_seg(1, N, rt);
}

int query(int l, int r, int rt) {
    s = l;
    t = r;
    int ans = query_seg(1, N, rt, 0);

    s = 1;
    t = l;
    return max(ans, query_seg(1, N, rt, 1) - l);
}

struct node {
    int size, l, r, id, tim;
    node *ch[2], *p;
    bool tag;

    node() = default;

    void apply(int v) {
        tim = v;
        tag = true;
    }

    void pushdown() {
        if (tag) {
            ch[0] -> tim = ch[1] -> tim = tim;
            ch[0] -> tag = ch[1] -> tag = true;

            tag = false;
        }
    }

    void update() {
        size = ch[0] -> size + ch[1] -> size + 1;
        l = (ch[0] -> l ? ch[0] -> l : id);
        r = (ch[1] -> r ? ch[1] -> r : id);
    }
} null[maxn];

inline bool isroot(node *x) {
```

```
    return x != x -> p -> ch[0] && x != x -> p ->
      ↪ ch[1];
}

inline bool dir(node *x) {
    return x == x -> p -> ch[1];
}

void init(node *x, int i) {
    *x = node();
    x -> ch[0] = x -> ch[1] = x -> p = null;
    x -> size = 1;
    x -> id = x -> l = x -> r = i;
}

void rot(node *x, int d) {
    node *y = x -> ch[d ^ 1];

    y -> p = x -> p;
    if (!isroot(x))
        x -> p -> ch[dir(x)] = y;

    if ((x -> ch[d ^ 1] = y -> ch[d]) != null)
        y -> ch[d] -> p = x;
    (y -> ch[d] = x) -> p = y;

    x -> update();
    y -> update();
}

void splay(node *x) {
    x -> pushdown();

    while (!isroot(x)) {
        if (!isroot(x -> p))
            x -> p -> p -> pushdown();
        x -> p -> pushdown();
        x -> pushdown();

        if (isroot(x -> p)) {
            rot(x -> p, dir(x) ^ 1);
            break;
        }

        if (dir(x) == dir(x -> p))
            rot(x -> p -> p, dir(x -> p) ^ 1);
        else
            rot(x -> p, dir(x) ^ 1);

        rot(x -> p, dir(x) ^ 1);
    }
}

void splay(node *x, node *rt) {
    x -> pushdown();

    while (x -> p != rt) {
        if (x -> p -> p != rt)
            x -> p -> p -> pushdown();
        x -> p -> pushdown();
        x -> pushdown();

        if (x -> p -> p == rt) {
            rot(x -> p, dir(x) ^ 1);
            break;
        }

        if (dir(x) == dir(x -> p))
            rot(x -> p -> p, dir(x -> p) ^ 1);
        else
            rot(x -> p, dir(x) ^ 1);

        rot(x -> p, dir(x) ^ 1);
    }
}

int val[maxn], par[maxn], go[maxn][26], sam_cnt,
  ↪ sam_last;

node *access(node *x, int r) {
    root[r] = root[r - 1];

    node *y = null;

    while (x != null) {
        splay(x);

        x -> pushdown();

        x -> ch[1] = null;
        x -> update();

        if (x -> tim && val[x -> r]) { // last time
          ↪ visited
            int right = x -> tim, left = right - val[x
              ↪ -> r] + 1;
            modify(left, val[x -> r], right + 1,
              ↪ root[r]);
        }

        x -> apply(r);
        x -> pushdown();

        x -> ch[1] = y;
        (y = x) -> update();

        x = x -> p;
    }

    return y;
}

void new_leaf(node *x, node *par) {
    x -> p = par;
}

void new_node(node *x, node *y, node *par) {
    splay(y);

    if (isroot(y) && y -> p == par) {
        assert(y -> ch[0] == null);

        y -> ch[0] = x;
        x -> p = y;
        y -> update();
    }
    else {
        splay(par, y);

        assert(y -> ch[0] == par);
        assert(par -> ch[1] == null);
        par -> ch[1] = x;
        x -> p = par;

        par -> update();
        y -> update();
    }

    x -> tim = y -> tim;
```

```
}

void extend(int c) {
    int p = sam_last, np = ++sam_cnt;
    val[np] = val[p] + 1;

    init(null + np, np);

    while (p && !go[p][c]) {
        go[p][c] = np;
        p = par[p];
    }

    if (!p) {
        par[np] = 1;
        new_leaf(null + np, null + par[np]);
    }
    else {
        int q = go[p][c];

        if (val[q] == val[p] + 1) {
            par[np] = q;
            new_leaf(null + np, null + par[np]);
        }
        else {
            int nq = ++sam_cnt;
            val[nq] = val[p] + 1;
            memcpy(go[nq], go[q], sizeof(go[q]));

            init(null + nq, nq);

            new_node(null + nq, null + q, null +
              ↪ par[q]);
            new_leaf(null + np, null + nq);

            par[nq] = par[q];
            par[np] = par[q] = nq;

            while (p && go[p][c] == q) {
                go[p][c] = nq;
                p = par[p];
            }
        }
    }

    sam_last = np;
}

char str[maxn];

int main() {

    init(null, 0);

    sam_last = sam_cnt = 1;
    init(null + 1, 1);

    int n, m;
    scanf("%s%d", str + 1, &m);
    n = strlen(str + 1);
    N = n + m;

    for (int i = 1; i ≤ n; i++) {
        extend(str[i] - 'a');
        access(null + sam_last, i);
    }

    int tmp = 0;

    while (m--) {
        int op;
        scanf("%d", &op);

        if (op == 1) {
            scanf(" %c", &str[++n]);

            str[n] = (str[n] - 'a' + tmp) % 26 + 'a';

            extend(str[n] - 'a');
            access(null + sam_last, n);
        }
        else {
            int l, r;
            scanf("%d%d", &l, &r);

            l = (l - 1 + tmp) % n + 1;
            r = (r - 1 + tmp) % n + 1;

            printf("%d\n", tmp = query(l, r,
              ↪ root[r]));
        }
    }

    return 0;
}
```

6 动态规划

6.1 例题

6.1.1 103388A Assigning Prizes 容斥

题意 给定一个长为 n 的序列 a_i, 要求构造非严格递减序列 b_i, 满足 $a_i \le b_i \le R$, 求方案数. $n \le 5 \times 10^3, R, a_i \le 10^9$.

做法 a_i 的范围太大了, 不能简单地记录上一位的值.
考虑使用容斥. 方便起见把 a_i 直接变成 $R - a_i + 1$, 条件就变成了 $b_i \le a_i$ 且 $b_i \ge b_{i-1}$.
这里有两个限制条件, 可以固定 $b_i \le a_i$ 是必须满足的条件, 只对 $b_i \ge b_{i-1}$ 使用容斥, 枚举哪些位置是比上一位小的(违反限制), 其他位置随意.
枚举后的形态一定是有若干个区间是严格递减的, 其他位置随意. 考虑如果一个区间 $[l, r]$ 是严格递减的, 显然所有的数都 $< a_l$, 所以这段区间的方案数就是 $\binom{a_l}{r-l+1}$. 另外实际上 b_l 是没有违反限制的, 所以这里对系数的贡献是 $(-1)^{r-l}$.
考虑令 dp_i 表示只考虑前 i 个位置的答案, 转移时自然就是枚举一个 j, 然后计算 dp_{j-1} 乘上区间 $[j, i]$ 严格递减的方案数. 另外还有一种情况是 b_i 没有违反限制, 这时显然直接在 dp_{i-1} 的基础上乘上一个 a_i 就好了. (转移时还要注意, 由于枚举的是严格递减区间, 自然就不能枚举只有一个数的区间.)

```
constexpr int maxn = 5005, p = (int)1e9 + 7;

int inv[maxn];
int a[maxn], f[maxn][maxn], dp[maxn];

int main() {

    int n, m;
    scanf("%d%d", &n, &m);

    inv[1] = 1;
    for (int i = 2; i ≤ n; i++)
        inv[i] = (long long)(p - p / i) * inv[p % i] %
          ↪ p;
```

```
14
15      for (int i = 1; i ≤ n; i++) {
16          scanf("%d", &a[i]);
17          a[i] = m - a[i] + 1;
18      }
19
20      if (any_of(a + 1, a + n + 1, [] (int x) {return x
          ↪ ≤ 0;})) {
21          printf("0\n");
22          return 0;
23      }
24
25      for (int i = n - 1; i; i--)
26          a[i] = min(a[i], a[i + 1]);
27
28      // b_i ≥ b_{i - 1} && b_i ≤ a_i
29      // 我们可以假设 b_i ≤ a_i 是必定被满足的□然后对 bi
          ↪ 非严格递增的条件进行容斥□枚举某一段是严格递减的
30      // 如果 [j, i] 严格递减□显然它们都 ≤ a_j□所以这个区
          ↪ 间的方案数是 {a_j \choose i - j + 1}
31      // 如果 i 是合法的□直接一个个转移即可□因为这一部分的
          ↪ 转移和区间长度没有关系
32
33      for (int i = 1; i ≤ n; i++) {
34          f[i][0] = 1;
35
36          for (int j = 1; j ≤ n - i + 1 && j ≤ a[i];
              ↪ j++)
37              f[i][j] = (long long)f[i][j - 1] * (a[i] -
                  ↪ j + 1) % p * inv[j] % p;
38      }
39
40      dp[0] = 1;
41
42      for (int i = 1; i ≤ n; i++) {
43          dp[i] = (long long)dp[i - 1] * a[i] % p;
44
45          for (int j = 1; j < i; j++) {
46              int tmp = (long long)dp[j - 1] * f[j][i -
                  ↪ j + 1] % p;
47
48              if ((i - j) % 2)
49                  tmp = p - tmp;
50
51              dp[i] = (dp[i] + tmp) % p;
52          }
53      }
54
55      printf("%d\n", dp[n]);
56
57      return 0;
58  }
```

7 计算几何

7.1 最近点对

首先分治的做法是众所周知的.

有期望$O(n)$的随机增量法：首先将所有点随机打乱，然后每次增加一个点，更新答案.

假设当前最近点对距离为s，则把平面划分成$s \times s$的方格，用哈希表存储每个方格有哪些点.

加入一个新点时只需要枚举自身和周围共计9个方格中的点，显然枚举到的点最多16个. 如果加入之后答案变小了就$O(n)$暴力重构.

前i个点中i是最近点对中的点的概率至多为$\frac{2}{i}$，所以每个点的期望贡献都是$O(1)$，总的复杂度就是期望$O(n)$.

如果对每个点都要求出距离最近的点的话，也有随机化的$O(n)$做法:

一个真的随机算法:

A simple randomized sieve algorithm for the closest-pair problem (https://w
/grant/cp.pdf)

1. 循环直到删完所有点:
   - 随机选一个点，计算它到所有点的最短距离 d。
   - 将所有点划分到 $l = d/3$ 的网格里，比如 $(\lfloor \frac{x}{l} \rfloor, \lfloor \frac{y}{l} \rfloor)$。
   - 将九宫格内孤立的点删除，这意味着这些点的最近点对距离不小于 $\frac{2\sqrt{2}}{3}d$,
2. 取最后一个 d，将所有点划分到 $(\lfloor \frac{x}{d} \rfloor, \lfloor \frac{y}{d} \rfloor)$ 的网格里，暴力计算九宫格内的

第一部分每次期望会删掉至少一半的点，因为有 $\geq 1/2$ 概率随到一个最近点距离
此第一部分的复杂度是 $O(n)$ 的。

第二部分分析类似分治做法，周围只有常数个点。

所以总复杂度是 $O(n)$ 的。

42 6 条评论

8 杂项

8.1 STL

8.1.1 vector

- `vector(int nSize)`: 创建一个vector, 元素个数为nSize
- `vector(int nSize, const T &value)`: 创建一个vector, 元素个数为nSize, 且值均为value
- `vector(begin, end)`: 复制[begin, end)区间内另一个数组的元素到vector中
- `void assign(int n, const T &x)`: 设置向量中前n个元素的值为x
- `void assign(const_iterator first, const_iterator last)`: 向量中[first, last)中元素设置成当前向量元素
- `void emplace_back(Args&& ... args)`: 自动构造并push_back一个元素, 例如对一个存储pair的vector可以`v.emplace_back(x, y)`

8.1.2 list

- `assign()` 给list赋值
- `back()` 返回最后一个元素
- `begin()` 返回指向第一个元素的迭代器
- `clear()` 删除所有元素
- `empty()` 如果list是空的则返回true
- `end()` 返回末尾的迭代器
- `erase()` 删除一个元素
- `front()` 返回第一个元素
- `insert()` 插入一个元素到list中
- `max_size()` 返回list能容纳的最大元素数量
- `merge()` 合并两个list
- `pop_back()` 删除最后一个元素
- `pop_front()` 删除第一个元素
- `push_back()` 在list的末尾添加一个元素
- `push_front()` 在list的头部添加一个元素
- `rbegin()` 返回指向第一个元素的逆向迭代器
- `remove()` 从list删除元素
- `remove_if()` 按指定条件删除元素
- `rend()` 指向list末尾的逆向迭代器
- `resize()` 改变list的大小
- `reverse()` 把list的元素倒转
- `size()` 返回list中的元素个数

- `sort()` 给list排序
- `splice()` 合并两个list
- `swap()` 交换两个list
- `unique()` 删除list中重复的元

8.1.3 unordered_set/map

- `unordered_map<int, int, hash>`: 自定义哈希函数，其中hash是一个带重载括号的类.

8.2 德州扑克

一般来说德扑里Ace都是最大的，所以把Ace的点数规定为14会好写许多.
附一个高低奥马哈的参考代码，除了有四张底牌和需要比低之外和德扑区别不大.

```cpp
struct Card {
    int suit, value; // Ace is treated as 14

    Card(string s) {
        char a = s[0];

        if (isdigit(a))
            value = a - '0';
        else if (a == 'T')
            value = 10;
        else if (a == 'A')
            value = 14;
        else if (a == 'J')
            value = 11;
        else if (a == 'Q')
            value = 12;
        else if (a == 'K')
            value = 13;
        else
            value = -1; // error

        char b = s[1];
        suit = b; // Club, Diamond, Heart, Spade
    }

    friend bool operator < (const Card &a, const Card &b) {
        return a.value < b.value;
    }

    friend bool operator == (const Card &a, const Card &b) {
        return a.value == b.value;
    }
};

constexpr int Highcard = 1, Pair = 2, TwoPairs = 3, ThreeofaKind = 4, Straight = 5,
Flush = 6, FullHouse = 7, FourofaKind = 8, StraightFlush = 9;

struct Hand {
    vector<Card> v;
    int type;

    Hand() : type(0) {}

    Hand(const Hand &o) : v(o.v), type(o.type) {}

    Hand(const vector<Card> &v) : v(v), type(0) {}

    void init_high() {
        sort(v.begin(), v.end()); // 升序排序

        bool straight = false;
        if (v.back().value == 14) {
            if (v[0].value == 2 && v[1].value == 3 && v[2].value == 4 && v[3].value == 5) {
                straight = true;
                rotate(v.begin(), v.begin() + 1, v.end());
            }
        }

        if (!straight) {
            bool ok = true;
            for (int i = 1; i < 5; i++)
                ok &= (v[i].value == v[i - 1].value + 1);

            if (ok)
                straight = true;
        }

        bool flush = all_of(v.begin(), v.end(), [&](const Card &a) {return a.suit == v.front().suit;});

        if (flush && straight) { // 同花顺
            type = StraightFlush;
            reverse(v.begin(), v.end());
            return;
        }

        vector<int> c;
        c.assign(15, 0);

        for (auto &o : v)
            c[o.value]++;

        vector<int> kind[5];

        for (int i = 2; i <= 14; i++)
            if (c[i] > 1)
                kind[c[i]].push_back(i);

        if (!kind[4].empty()) { // 四条
            type = FourofaKind;

            for (int i = 0; i < 4; i++)
                if (v[i].value != kind[4].front()) {
                    swap(v[i], v.back());
                    break;
                }

            return;
        }

        if (!kind[3].empty() && !kind[2].empty()) {
            type = FullHouse;

            sort(v.begin(), v.end(), [&](const Card &a, const Card &b) {
                bool ta = (a.value == kind[3].front()), tb = (b.value == kind[3].front());

                return ta > tb;
            });

            return;
        }
```

```
        if (flush) {
            type = Flush;
            sort(v.begin(), v.end());
            reverse(v.begin(), v.end());

            return;
        }

        if (straight) {
            type = Straight;
            reverse(v.begin(), v.end());
            return;
        }

        if (!kind[3].empty()) {
            type = ThreeofaKind;

            sort(v.begin(), v.end(), [&] (const Card &a, const Card &b) {
                bool ta = (a.value == kind[3].front()), tb = (b.value == kind[3].front());

                return ta > tb;
            });

            if (v[3] < v[4])
                swap(v[3], v[4]);

            return;
        }

        if ((int)kind[2].size() == 2) {
            type = TwoPairs;

            sort(v.begin(), v.end(), [&] (const Card &a, const Card &b) {
                bool ta = (c[a.value] == 2), tb = (c[b.value] == 2);

                if (ta != tb)
                    return ta > tb;

                return a.value > b.value;
            });

            return;
        }

        if ((int)kind[2].size() == 1) {
            type = Pair;

            sort(v.begin(), v.end(), [&] (const Card &a, const Card &b) {
                bool ta = (c[a.value] == 2), tb = (c[b.value] == 2);

                if (ta != tb)
                    return ta > tb;

                return a.value > b.value;
            });

            return;
        }

        type = Highcard;

        sort(v.begin(), v.end());
        reverse(v.begin(), v.end());
    }

    void init_low() {
        for (auto &o : v)
            if (o.value == 14)
                o.value = 1;

        sort(v.begin(), v.end());
        reverse(v.begin(), v.end());
    }

    friend int cmp_high(const Hand &a, const Hand &b) {
        if (a.type != b.type)
            return a.type < b.type ? -1 : 1;

        if (a.v != b.v)
            return a.v < b.v ? -1 : 1;

        return 0;
    }

    friend bool small_high(const Hand &a, const Hand &b) {
        return cmp_high(a, b) < 0;
    }

    friend int cmp_low(const Hand &a, const Hand &b) {
        for (int i = 0; i < 5; i++)
            if (a.v[i].value != b.v[i].value)
                return a.v[i] < b.v[i] ? 1 : -1;

        return 0;
    }

    friend bool small_low(const Hand &a, const Hand &b) {
        return cmp_low(a, b) < 0;
    }

    bool operator ! () const {
        return v.empty();
    }

    string str() const {
        stringstream ss;

        for (auto &o : v)
            ss << o.value << ' ';

        return ss.str();
    }
};

Hand get_max_high(vector<Card> u, vector<Card> v) { // private, public
    Hand ans;

    for (int i = 0; i < 4; i++)
        for (int j = i + 1; j < 4; j++)
            for (int k = 0; k < 5; k++)
                for (int p = k + 1; p < 5; p++)
                    for (int q = p + 1; q < 5; q++) {
                        Hand tmp({u[i], u[j], v[k], v[p], v[q]});

                        tmp.init_high();

```

```
                    if (!ans || cmp_high(tmp, ans) > 0)
                        ans = tmp;
                }

    return ans;
}

Hand get_max_low(vector<Card> tu, vector<Card> tv) {
    vector<Card> u, v;

    for (auto o : tu)
        if (o.value == 14 || o.value <= 8)
            u.push_back(o);

    for (auto o : tv)
        if (o.value == 14 || o.value <= 8)
            v.push_back(o);

    Hand ans;

    for (int i = 0; i < (int)u.size(); i++)
        for (int j = i + 1; j < (int)u.size(); j++)
            for (int k = 0; k < (int)v.size(); k++)
                for (int p = k + 1; p < (int)v.size(); p++)
                    for (int q = p + 1; q < (int)v.size(); q++) {
                        vector<Card> vec = {u[i], u[j], v[k], v[p], v[q]};

                        bool bad = false;

                        for (int a = 0; a < 5; a++)
                            for (int b = a + 1; b < 5; b++)
                                if (vec[a].value == vec[b].value)
                                    bad = true;

                        if (bad)
                            continue;

                        Hand tmp(vec);

                        tmp.init_low();

                        if (!ans || cmp_low(tmp, ans) > 0)
                            ans = tmp;
                    }

    return ans;
}

int main() {

    ios::sync_with_stdio(false);

    int T;
    cin >> T;

    while (T--) {
        int p;
        cin >> p;

        vector<Card> alice, bob, pub;

        for (int i = 0; i < 4; i++) {
            string s;
            cin >> s;
            alice.push_back(Card(s));
        }

        for (int i = 0; i < 4; i++) {
            string s;
            cin >> s;
            bob.push_back(Card(s));
        }

        for (int i = 0; i < 5; i++) {
            string s;
            cin >> s;
            pub.push_back(Card(s));
        }

        Hand alice_high = get_max_high(alice, pub), bob_high = get_max_high(bob, pub);
        Hand alice_low = get_max_low(alice, pub), bob_low = get_max_low(bob, pub);

        int dh = cmp_high(alice_high, bob_high);
        int ans[2] = {0};

        if (!alice_low && !bob_low) {
            if (!dh) {
                ans[0] = p - p / 2;
                ans[1] = p / 2;
            }
            else
                ans[dh == -1] = p;
        }
        else if (!alice_low || !bob_low) {
            ans[!alice_low] += p / 2;

            if (!dh) {
                ans[0] += p - p / 2 - (p - p / 2) / 2;
                ans[1] += (p - p / 2) / 2;
            }
            else
                ans[dh == -1] += p - p / 2;
        }
        else {
            int dl = cmp_low(alice_low, bob_low);

            if (!dl) {
                ans[0] += p / 2 - p / 2 / 2;
                ans[1] += p / 2 / 2;
            }
            else
                ans[dl == -1] += p / 2;

            if (!dh) {
                ans[0] += p - p / 2 - (p - p / 2) / 2;
                ans[1] += (p - p / 2) / 2;
            }
            else
                ans[dh == -1] += p - p / 2;
        }

        cout << ans[0] << ' ' << ans[1] << '\n';
    }

    return 0;
}
```

Visual Studio Code

Keyboard shortcuts for Windows

General

Ctrl+Shift+P, F1	Show Command Palette
Ctrl+P	Quick Open, Go to File...
Ctrl+Shift+N	New window/instance
Ctrl+Shift+W	Close window/instance
Ctrl+,	User Settings
Ctrl+K Ctrl+S	Keyboard Shortcuts

Basic editing

Ctrl+X	Cut line (empty selection)
Ctrl+C	Copy line (empty selection)
Alt+ ↑ / ↓	Move line up/down
Shift+Alt + ↓ / ↑	Copy line up/down
Ctrl+Shift+K	Delete line
Ctrl+Enter	Insert line below
Ctrl+Shift+Enter	Insert line above
Ctrl+Shift+\	Jump to matching bracket
Ctrl+] / [	Indent/outdent line
Home / End	Go to beginning/end of line
Ctrl+Home	Go to beginning of file
Ctrl+End	Go to end of file
Ctrl+↑ / ↓	Scroll line up/down
Alt+PgUp / PgDn	Scroll page up/down
Ctrl+Shift+[	Fold (collapse) region
Ctrl+Shift+]	Unfold (uncollapse) region
Ctrl+K Ctrl+[	Fold (collapse) all subregions
Ctrl+K Ctrl+]	Unfold (uncollapse) all subregions
Ctrl+K Ctrl+0	Fold (collapse) all regions
Ctrl+K Ctrl+J	Unfold (uncollapse) all regions
Ctrl+K Ctrl+C	Add line comment
Ctrl+K Ctrl+U	Remove line comment
Ctrl+/	Toggle line comment
Shift+Alt+A	Toggle block comment
Alt+Z	Toggle word wrap

Navigation

Ctrl+T	Show all Symbols
Ctrl+G	Go to Line...
Ctrl+P	Go to File...
Ctrl+Shift+O	Go to Symbol...
Ctrl+Shift+M	Show Problems panel
F8	Go to next error or warning
Shift+F8	Go to previous error or warning
Ctrl+Shift+Tab	Navigate editor group history
Alt+ ← / →	Go back / forward
Ctrl+M	Toggle Tab moves focus

Search and replace

Ctrl+F	Find
Ctrl+H	Replace
F3 / Shift+F3	Find next/previous
Alt+Enter	Select all occurences of Find match
Ctrl+D	Add selection to next Find match
Ctrl+K Ctrl+D	Move last selection to next Find match
Alt+C / R / W	Toggle case-sensitive / regex / whole word

Multi-cursor and selection

Alt+Click	Insert cursor
Ctrl+Alt+ ↑ / ↓	Insert cursor above / below
Ctrl+U	Undo last cursor operation
Shift+Alt+I	Insert cursor at end of each line selected
Ctrl+L	Select current line
Ctrl+Shift+L	Select all occurrences of current selection
Ctrl+F2	Select all occurrences of current word
Shift+Alt+→	Expand selection
Shift+Alt+←	Shrink selection
Shift+Alt + (drag mouse)	Column (box) selection
Ctrl+Shift+Alt + (arrow key)	Column (box) selection
Ctrl+Shift+Alt +PgUp/PgDn	Column (box) selection page up/down

Rich languages editing

Ctrl+Space, Ctrl+I	Trigger suggestion
Ctrl+Shift+Space	Trigger parameter hints
Shift+Alt+F	Format document
Ctrl+K Ctrl+F	Format selection
F12	Go to Definition
Alt+F12	Peek Definition
Ctrl+K F12	Open Definition to the side
Ctrl+.	Quick Fix
Shift+F12	Show References
F2	Rename Symbol
Ctrl+K Ctrl+X	Trim trailing whitespace
Ctrl+K M	Change file language

Editor management

Ctrl+F4, Ctrl+W	Close editor
Ctrl+K F	Close folder
Ctrl+\	Split editor
Ctrl+ 1 / 2 / 3	Focus into 1st, 2nd or 3rd editor group
Ctrl+K Ctrl+ ←/→	Focus into previous/next editor group
Ctrl+Shift+PgUp / PgDn	Move editor left/right
Ctrl+K ← / →	Move active editor group

File management

Ctrl+N	New File
Ctrl+O	Open File...
Ctrl+S	Save
Ctrl+Shift+S	Save As...
Ctrl+K S	Save All
Ctrl+F4	Close
Ctrl+K Ctrl+W	Close All
Ctrl+Shift+T	Reopen closed editor
Ctrl+K Enter	Keep preview mode editor open
Ctrl+Tab	Open next
Ctrl+Shift+Tab	Open previous
Ctrl+K P	Copy path of active file
Ctrl+K R	Reveal active file in Explorer
Ctrl+K O	Show active file in new window/instance

Display

F11	Toggle full screen
Shift+Alt+0	Toggle editor layout (horizontal/vertical)
Ctrl+ = / -	Zoom in/out
Ctrl+B	Toggle Sidebar visibility
Ctrl+Shift+E	Show Explorer / Toggle focus
Ctrl+Shift+F	Show Search
Ctrl+Shift+G	Show Source Control
Ctrl+Shift+D	Show Debug
Ctrl+Shift+X	Show Extensions
Ctrl+Shift+H	Replace in files
Ctrl+Shift+J	Toggle Search details
Ctrl+Shift+U	Show Output panel
Ctrl+Shift+V	Open Markdown preview
Ctrl+K V	Open Markdown preview to the side
Ctrl+K Z	Zen Mode (Esc Esc to exit)

Debug

F9	Toggle breakpoint
F5	Start/Continue
Shift+F5	Stop
F11 / Shift+F11	Step into/out
F10	Step over
Ctrl+K Ctrl+I	Show hover

Integrated terminal

Ctrl+`	Show integrated terminal
Ctrl+Shift+`	Create new terminal
Ctrl+C	Copy selection
Ctrl+V	Paste into active terminal
Ctrl+↑ / ↓	Scroll up/down
Shift+PgUp / PgDn	Scroll page up/down
Ctrl+Home / End	Scroll to top/bottom

Other operating systems' keyboard shortcuts and additional unassigned shortcuts available at aka.ms/vscodekeybindings

Visual Studio Code

Keyboard shortcuts for Linux

General

Ctrl+Shift+P, F1	Show Command Palette
Ctrl+P	Quick Open, Go to File...
Ctrl+Shift+N	New window/instance
Ctrl+W	Close window/instance
Ctrl+,	User Settings
Ctrl+K Ctrl+S	Keyboard Shortcuts

Basic editing

Ctrl+X	Cut line (empty selection)
Ctrl+C	Copy line (empty selection)
Alt+ ↓ / ↑	Move line down/up
Ctrl+Shift+K	Delete line
Ctrl+Enter / Ctrl+Shift+Enter	Insert line below/ above
Ctrl+Shift+\	Jump to matching bracket
Ctrl+] / Ctrl+[	Indent/Outdent line
Home / End	Go to beginning/end of line
Ctrl+ Home / End	Go to beginning/end of file
Ctrl+ ↑ / ↓	Scroll line up/down
Alt+ PgUp / PgDn	Scroll page up/down
Ctrl+Shift+ [/]	Fold/unfold region
Ctrl+K Ctrl+ [/]	Fold/unfold all subregions
Ctrl+K Ctrl+0 / Ctrl+K Ctrl+J	Fold/Unfold all regions
Ctrl+K Ctrl+C	Add line comment
Ctrl+K Ctrl+U	Remove line comment
Ctrl+/	Toggle line comment
Ctrl+Shift+A	Toggle block comment
Alt+Z	Toggle word wrap

Rich languages editing

Ctrl+Space, Ctrl+I	Trigger suggestion
Ctrl+Shift+Space	Trigger parameter hints
Ctrl+Shift+I	Format document
Ctrl+K Ctrl+F	Format selection
F12	Go to Definition
Ctrl+Shift+F10	Peek Definition
Ctrl+K F12	Open Definition to the side
Ctrl+.	Quick Fix
Shift+F12	Show References
F2	Rename Symbol
Ctrl+K Ctrl+X	Trim trailing whitespace
Ctrl+K M	Change file language

Multi-cursor and selection

Alt+Click	Insert cursor*
Shift+Alt+ ↑ / ↓	Insert cursor above/below
Ctrl+U	Undo last cursor operation
Shift+Alt+I	Insert cursor at end of each line selected
Ctrl+L	Select current line
Ctrl+Shift+L	Select all occurrences of current selection
Ctrl+F2	Select all occurrences of current word
Shift+Alt + →	Expand selection
Shift+Alt + ←	Shrink selection
Shift+Alt + drag mouse	Column (box) selection

Display

F11	Toggle full screen
Shift+Alt+0	Toggle editor layout (horizontal/vertical)
Ctrl+ = / -	Zoom in/out
Ctrl+B	Toggle Sidebar visibility
Ctrl+Shift+E	Show Explorer / Toggle focus
Ctrl+Shift+F	Show Search
Ctrl+Shift+G	Show Source Control
Ctrl+Shift+D	Show Debug
Ctrl+Shift+X	Show Extensions
Ctrl+Shift+H	Replace in files
Ctrl+Shift+J	Toggle Search details
Ctrl+Shift+C	Open new command prompt/terminal
Ctrl+K Ctrl+H	Show Output panel
Ctrl+Shift+V	Open Markdown preview
Ctrl+K V	Open Markdown preview to the side
Ctrl+K Z	Zen Mode (Esc Esc to exit)

Search and replace

Ctrl+F	Find
Ctrl+H	Replace
F3 / Shift+F3	Find next/previous
Alt+Enter	Select all occurrences of Find match
Ctrl+D	Add selection to next Find match
Ctrl+K Ctrl+D	Move last selection to next Find match

Navigation

Ctrl+T	Show all Symbols
Ctrl+G	Go to Line...
Ctrl+P	Go to File...
Ctrl+Shift+O	Go to Symbol...
Ctrl+Shift+M	Show Problems panel
F8	Go to next error or warning
Shift+F8	Go to previous error or warning
Ctrl+Shift+Tab	Navigate editor group history
Ctrl+Alt+-	Go back
Ctrl+Shift+-	Go forward
Ctrl+M	Toggle Tab moves focus

Editor management

Ctrl+W	Close editor
Ctrl+K F	Close folder
Ctrl+\	Split editor
Ctrl+ 1 / 2 / 3	Focus into 1st, 2nd, 3rd editor group
Ctrl+K Ctrl + ←	Focus into previous editor group
Ctrl+K Ctrl + →	Focus into next editor group
Ctrl+Shift+PgUp	Move editor left
Ctrl+Shift+PgDn	Move editor right
Ctrl+K ←	Move active editor group left/up
Ctrl+K →	Move active editor group right/down

File management

Ctrl+N	New File
Ctrl+O	Open File...
Ctrl+S	Save
Ctrl+Shift+S	Save As...
Ctrl+W	Close
Ctrl+K Ctrl+W	Close All
Ctrl+Shift+T	Reopen closed editor
Ctrl+K Enter	Keep preview mode editor open
Ctrl+Tab	Open next
Ctrl+Shift+Tab	Open previous
Ctrl+K P	Copy path of active file
Ctrl+K R	Reveal active file in Explorer
Ctrl+K O	Show active file in new window/instance

Debug

F9	Toggle breakpoint
F5	Start / Continue
F11 / Shift+F11	Step into/out
F10	Step over
Shift+F5	Stop
Ctrl+K Ctrl+I	Show hover

Integrated terminal

Ctrl+`	Show integrated terminal
Ctrl+Shift+`	Create new terminal
Ctrl+Shift+C	Copy selection
Ctrl+Shift+V	Paste into active terminal
Ctrl+Shift+ ↑ / ↓	Scroll up/down
Shift+ PgUp / PgDn	Scroll page up/down
Shift+ Home / End	Scroll to top/bottom

* The Alt+Click gesture may not work on some Linux distributions. You can change the modifier key for the Insert cursor command to Ctrl+Click with the “editor.multiCursorModifier” setting.