Contents

1. BASIC

1.1. `pbds.h`

```cpp
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;
using namespace std;
using ord_set = tree<int, null_type, less<int>, rb_tree_tag,
tree_order_statistics_node_update>;
using ord_mset =  tree<int, null_type, less_equal<int>, rb_tree_tag,
tree_order_statistics_node_update>;
//find_by_order
//order_of_key
```

2. Ds

2.1. `dsu.h`

```cpp
class DSU {
    std::vector<int> f, siz;
public:
    DSU() {}
    DSU(int n) {
        init(n);
    }

    void init(int n) {
        f.resize(n);
        for(int i = 0;i < n;i ++) f[i] = i;
        siz.assign(n, 1);
    }

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

    bool same(int x, int y) {
        return find(x) == find(y);
    }

    bool merge(int x, int y) {
        x = find(x);
        y = find(y);
        if (x == y) {
            return false;
        }
        siz[x] += siz[y];
        f[y] = x;
        return true;
    }

    int size(int x) {
        return siz[find(x)];
    }
};
```

2.2. segTree_add.h

```cpp
// AC 带懒惰标记线段树
template <class TYPE_NAME>
class lazyTree
{
    /*
     *  TYPE_NAME 需要支持: + += != 和自定义的合并运算符
     *  实现了懒惰标记，仅支持区间批量增加
     *  区间批量减需要 TYPE_NAME 支持-，且有-a = 0 - a
     *  额外处理了一个单点修改为对应值的函数，非原生实现，其复杂度为 4logn
     *  不提供在线
     *  不提供持久化
     */
private:
    vector<TYPE_NAME> d;
    vector<TYPE_NAME> a;
    vector<TYPE_NAME> b;
    int n;
    const TYPE_NAME INI = 0; // 不会影响合并运算的初始值，如 max 取 INF，min 取 0，mti 取 1

    void subbuild(int s, int t, int p)
    {
        if (s == t)
        {
            d[p] = a[s];
            return;
        }
        int m = s + ((t - s) >> 1); //  (s+t)/2
        subbuild(s, m, p * 2);
        subbuild(m + 1, t, p * 2 + 1);
        d[p] = d[p * 2] + d[p * 2 + 1];
        //     合并运算符 ↑
    }

    TYPE_NAME subGetSum(int l, int r, int s, int t, int p)
    {
        if (l <= s && t <= r)
            return d[p];
        int m = s + ((t - s) >> 1);
        if (b[p] != 0)
        {
            d[p * 2] += b[p] * (m - s + 1); // 合并运算符的高阶运算 此处运算为应用懒惰标记
            d[p * 2 + 1] += b[p] * (t - m); // 合并运算符的高阶运算 此处运算为应用懒惰标记
            b[p * 2] += b[p];               // 下传标记，无需修改
            b[p * 2 + 1] += b[p];           // 下传标记，无需修改
            b[p] = 0;
        }
        TYPE_NAME ansl = INI;
        TYPE_NAME ansr = INI;
        if (l <= m)
            ansl = subGetSum(l, r, s, m, p * 2);
        if (r > m)
            ansr = subGetSum(l, r, m + 1, t, p * 2 + 1);
```

```
53          return ansl + ansr;
54          // 合并运算符↑
55      }
56
57      void subUpdate(int l, int r, TYPE_NAME c, int s, int t, int p)
58      {
59          if (l <= s && t <= r)
60          {
61              d[p] += (t - s + 1) * c; // 合并运算符的高阶运算 此处运算为修改整匹配区间值
62              b[p] += c;               // 记录懒惰标记，无需修改
63              return;
64          }
65          int m = s + ((t - s) >> 1);
66          if (b[p] != 0 && s != t)
67          {
68              d[p * 2] += b[p] * (m - s + 1); // 合并运算符的高阶运算 此处运算为应用懒惰标记
69              d[p * 2 + 1] += b[p] * (t - m); // 合并运算符的高阶运算 此处运算为应用懒惰标记
70              b[p * 2] += b[p];               // 下传标记，无需修改
71              b[p * 2 + 1] += b[p];           // 下传标记，无需修改
72              b[p] = 0;
73          }
74          if (l <= m)
75              subUpdate(l, r, c, s, m, p * 2);
76          if (r > m)
77              subUpdate(l, r, c, m + 1, t, p * 2 + 1);
78          d[p] = d[p * 2] + d[p * 2 + 1];
79          //    合并运算符 ↑
80      }
81
82  public:
83      lazyTree(TYPE_NAME _n)
84      {
85          n = _n;
86          d.resize(4 * n + 5);
87          a.resize(4 * n + 5);
88          b.resize(4 * n + 5);
89      }
90
91      void build(vector<TYPE_NAME> _a)
92      {
93          a = _a;
94          subbuild(1, n, 1);
95      }
96
97      TYPE_NAME getsum(int l, int r)
98      {
99          return subGetSum(l, r, 1, n, 1);
100     }
101
102     void update(int l, int r, TYPE_NAME c) // 修改区间
103     {
104         subUpdate(l, r, c, 1, n, 1);
105     }
106
```

```
107         void update(int idx, TYPE_NAME tar)
108         { // 修改单点，未测试
109             TYPE_NAME tmp = getsum(idx, idx);
110             tar -= tmp;
111             subUpdate(idx, idx, tar, 1, n, 1);
112         }
113     };
```

2.3. segTree_MX_MI.h

```cpp
//AC MJ 的 MIN/MAX 树
template<class Info>
struct SegmentTree {
    int n;
    std::vector<Info> info;
    SegmentTree() : n(0) {}
    SegmentTree(int n_, Info v_ = Info()) {
        init(n_, v_);
    }
    template<class T>
    SegmentTree(std::vector<T> init_) {
        init(init_);
    }
    void init(int n_, Info v_ = Info()) {
        init(std::vector(n_, v_));
    }
    template<class T>
    void init(std::vector<T> init_) {
        n = init_.size();
        info.assign(4 << std::__lg(n), Info());
        std::function<void(int, int, int)> build = [&](int p, int l, int r) {
            if (r - l == 1) {
                info[p] = init_[l];
                return;
            }
            int m = (l + r) / 2;
            build(2 * p, l, m);
            build(2 * p + 1, m, r);
            pull(p);
        };
        build(1, 0, n);
    }
    void pull(int p) {
        info[p] = info[2 * p] + info[2 * p + 1];
    }
    void modify(int p, int l, int r, int x, const Info &v) {
        if (r - l == 1) {
            info[p] = v;
            return;
        }
        int m = (l + r) / 2;
        if (x < m) {
            modify(2 * p, l, m, x, v);
        } else {
            modify(2 * p + 1, m, r, x, v);
        }
        pull(p);
    }
    void modify(int p, const Info &v) {
        modify(1, 0, n, p, v);
    }
    Info rangeQuery(int p, int l, int r, int x, int y) {
        if (l >= y || r <= x) {
            return Info();
        }
        if (l >= x && r <= y) {
```

```
            return info[p];
        }
        int m = (l + r) / 2;
        return rangeQuery(2 * p, l, m, x, y) + rangeQuery(2 * p + 1, m, r, x,
y);
    }
    Info rangeQuery(int l, int r) {
        return rangeQuery(1, 0, n, l, r);
    }
    template<class F>
    int findFirst(int p, int l, int r, int x, int y, F &&pred) {
        if (l >= y || r <= x) {
            return -1;
        }
        if (l >= x && r <= y && !pred(info[p])) {
            return -1;
        }
        if (r - l == 1) {
            return l;
        }
        int m = (l + r) / 2;
        int res = findFirst(2 * p, l, m, x, y, pred);
        if (res == -1) {
            res = findFirst(2 * p + 1, m, r, x, y, pred);
        }
        return res;
    }
    template<class F>
    int findFirst(int l, int r, F &&pred) {
        return findFirst(1, 0, n, l, r, pred);
    }
    template<class F>
    int findLast(int p, int l, int r, int x, int y, F &&pred) {
        if (l >= y || r <= x) {
            return -1;
        }
        if (l >= x && r <= y && !pred(info[p])) {
            return -1;
        }
        if (r - l == 1) {
            return l;
        }
        int m = (l + r) / 2;
        int res = findLast(2 * p + 1, m, r, x, y, pred);
        if (res == -1) {
            res = findLast(2 * p, l, m, x, y, pred);
        }
        return res;
    }
    template<class F>
    int findLast(int l, int r, F &&pred) {
        return findLast(1, 0, n, l, r, pred);
    }
};
const int inf = 1E9;
struct Info
{
```

```
113     int mn {inf}, mnId, mx {-inf}, mxId;
114 } ;
115 Info operator+(Info a, Info b)
116 {
117     if (a.mn > b.mn)
118         a.mn = b.mn, a.mnId = b.mnId;
119     if (a.mx < b.mx)
120         a.mx = b.mx, a.mxId = b.mxId;
121     return a;
122 }
```

2.4. twoDimPrfxSum.h

```cpp
class prfx_2{
public:
    vector<vector<int>> mtx;
    int n,m;
    public:
    prfx_2(vector<vector<int>> _mtx){init(_mtx);};
    prfx_2() { };
    void init(vector<vector<int>> _mtx)
    {
        n = _mtx.size();
        m = _mtx[0].size();
        mtx.resize(n+1);
        for(auto &x:mtx) x.resize(m+1);
        lop(i,1,n+1)
            lop(j,1,m+1)
                mtx[i][j] = mtx[i-1][j] + mtx[i][j-1] - mtx[i-1][j-1] + _mtx[i-1][j-1];
    }

    int getsum(int x1,int y1,int x2,int y2)
    {
        x1 ++,x2 ++,y1 ++,y2 ++;
        return mtx[x2][y2] - mtx[x1-1][y2] - mtx[x2][y1-1] + mtx[x1-1][y1-1];
    }

    int getsum(pii d1,pii d2)
    {
        auto [x1,y1] = d1;
        auto [x2,y2] = d2;
        x1 ++,x2 ++,y1 ++,y2 ++;
        return mtx[x2][y2] - mtx[x1-1][y2] - mtx[x2][y1-1] + mtx[x1-1][y1-1];
    }

    vector<int> & operator[](std::size_t i)
    {
        return mtx[i];
    }

};

class conj_diff_2{
    vector<vector<int>> mtx;
    vector<vector<int>> prf;
    int n,m;
    public:

    conj_diff_2(int _n,int _m)
    {
        n = _n+1,m = _m+1;
        vector<vector<int>> initmp(n,vector<int> (m,0));
        init(initmp);
    }

    conj_diff_2(vector<vector<int>> _mtx)
    {
        this->init(_mtx);
    }
```

```
57
58      conj_diff_2(){ }
59
60      void init(vector<vector<int>> _mtx)
61      {
62          n = _mtx.size();
63          m = _mtx[0].size();
64          mtx.resize(n+2);
65          for(auto &x:mtx) x.resize(m+2);
66          prf.resize(n+2);
67          for(auto &x:prf) x.resize(m+2);
68          lop(i,1,n+1)
69              lop(j,1,m+1)
70                  prf[i][j] = _mtx[i-1][j-1];
71      }
72
73      void modify(int x1,int y1,int x2,int y2,int k)
74      {
75          x1 ++,x2 ++,y1 ++,y2 ++;
76          mtx[x1][y1] += k;
77          mtx[x2+1][y1] -= k,mtx[x1][y2+1] -= k;
78          mtx[x2+1][y2+1] += k;
79      }
80
81      void modify(pii d1,pii d2,int k)
82      {
83          this->modify(d1.fi,d1.se,d2.fi,d2.se,k);
84      }
85
86      vector<vector<int>> cacu()
87      {
88          auto res = prfx_2(mtx);
89          vector<vector<int>> rst(n,vector<int>(m));
90          lop(i,1,n+1)
91              lop(j,1,m+1)
92                  rst[i-1][j-1] = prf[i][j] +  res[i+1][j+1];
93          return rst;
94      }
95
96      vector<int> & operator[](std::size_t i)
97      {
98          return mtx[i];
99      }
100 };
```

3. GEO

3.1. Rotating_Calipers.h

```cpp
//Rotating_Calipers
template<typename VALUE_TYPE>
class Rotating_Calipers
{
public:
    using pv = pair<VALUE_TYPE, VALUE_TYPE>;
    using vec_pv = vector<pair<VALUE_TYPE, VALUE_TYPE>>;
    vec_pv p;

    static VALUE_TYPE cross(pv p1, pv p2, pv p0)
    {
        pv t1 = {p1.fi - p0.fi, p1.se - p0.se},
           t2 = {p2.fi - p0.fi, p2.se - p0.se};
        return t1.fi * t2.se - t1.se * t2.fi;
    }

    static VALUE_TYPE dis(const pv &p1,const pv &p2){
        return (p1.fi - p2.fi) * (p1.fi - p2.fi) + (p1.se - p2.se) * (p1.se - p2.se);
    };

public:

    Rotating_Calipers() {}

    Rotating_Calipers(vec_pv _A) {
        build(_A);
    }

    void build(const vec_pv & _A) {
        p = ConvexHull(_A);
    }

    static vec_pv ConvexHull(vec_pv A, VALUE_TYPE flag = 1)
    {
        int n = A.size();
        if (n <= 2) return A;
        vec_pv ans(n * 2);
        sort(A.begin(), A.end(),
        [](pv a,pv b) -> bool {
            if(fabs(a.fi - b.fi) < 1e-10)
                return a.se < b.se;
            else return a.fi < b.fi;}    );
        int now = -1;
        for (int i = 0; i < n; i++)
        { // 维护下凸包
            while (now > 0 && cross(A[i], ans[now], ans[now - 1]) < flag) now--;
            ans[++now] = A[i];
        }
        int pre = now;
        for (int i = n - 2; i >= 0; i--)
        { // 维护上凸包
```

```
            while (now > pre && cross(A[i], ans[now], ans[now - 1]) < flag)
now--;
            ans[++now] = A[i];
        }
        ans.resize(now);
        return ans;
    }

    VALUE_TYPE getDiameter() {
        int j = 1;
        VALUE_TYPE ans = 0;
        int m = p.size();
        p.push_back(p[0]);
        for(int i = 0;i < m;i ++)
        {
            while( cross(p[i+1],p[j],p[i]) > cross(p[i+1],p[j+1],p[i]) ) j =
(j+1)%m;
            ans = max(ans, max( dis(p[i],p[j]) , dis(p[i+1],p[j]) ) );
        }
        p.pop_back();
        return ans;
    }

    VALUE_TYPE getPerimeter() {
        VALUE_TYPE sum = 0;
        p.pb(p[0]);
        for(int i = 0;i < (int)p.size() - 1;i ++)
        {
            sum += sqrtl(dis(p[i],p[i+1]));
        }
        p.pop_back();
        return sum;
    }

};
```

4. GRAPH

4.1. FLOW

4.1.1. max_Flow.h

```
class maxFlow//AC
{
private:
    class edge
    {
    public:
        ll int nxt,                    // 出度
              cap,                    // 容量
              flow;                   // 流量
        pair<int, int> revNodeIdx; // 反向边
    public:
        edge(int _nxt, int _cap)
        {
            nxt = _nxt;
            cap = _cap;
            flow = 0;
        }
        void setRevIdx(int _i, int _j)
        {
            revNodeIdx.first = _i;
            revNodeIdx.second = _j;
        }
    };
    vector<vector<edge>> edgeList; // 节点列表
    vector<int> dep;               // 深度
    vector<int> fir;               // 节点最近合法边
    ll int maxFlowAns;

    int T, S;

public:
    maxFlow(int _n)
    {
        maxFlowAns = 0;
        S = 1;
        T = _n;
        edgeList.resize(_n + 1);
        dep.resize(_n + 1);
        fir.resize(_n+1);
    }

    void resetTS(int _T, int _S)
    {
        T = _T;
        S = _S;
    }

    void addedge(int _u, int _v, int _w)
    {
        edgeList[_u].push_back(edge(_v, _w));
        edgeList[_v].push_back(edge(_u, 0)); // 反向建边
```

```
52          edgeList[_u][edgeList[_u].size() - 1].setRevIdx(_v,
    edgeList[_v].size() - 1);
53          edgeList[_v][edgeList[_v].size() - 1].setRevIdx(_u,
    edgeList[_u].size() - 1);
54      }
55
56      bool bfs() // 统计深度
57      {
58          queue<int> que;
59          for (auto x = dep.begin(); x != dep.end(); x++)
60              (*x) = 0;
61
62          dep[S] = 1;
63          que.push(S);
64          while (que.size())
65          {
66              ll int at = que.front();
67              que.pop();
68              for (int i = 0; i < edgeList[at].size(); i++)
69              {
70                  auto tar = edgeList[at][i];
71                  if ((!dep[tar.nxt]) && (tar.flow < tar.cap))
72                  {
73                      dep[tar.nxt] = dep[at] + 1;
74                      que.push(tar.nxt);
75                  }
76              }
77          }
78          return dep[T];
79      }
80
81      ll int dfs(int at, ll int flow)
82      {
83          if ((at == T) || (!flow))
84              return flow; // 到了或者没了
85          ll int ret = 0;  // 本节点最大流
86          for (int &i = fir[at]; i < edgeList[at].size(); i++)
87          {
88              auto tar = edgeList[at][i];       // 目前遍历的边
89              int tlFlow = 0;                   // 目前边的最大流
90              if (dep[at] == dep[tar.nxt] - 1) // 遍历到的边为合法边
91              {
92                  tlFlow = dfs(tar.nxt, min((ll)tar.cap - tar.flow, flow -
    ret));
93                  if (!tlFlow)
94  continue;                                                    // 若最大流
    为 0，什么都不做
95                  ret +=
    tlFlow;                                                  // 本节点最大流
    累加
96                  edgeList[at][i].flow +=
    tlFlow;                                          // 本节点实时流量
97                  edgeList[tar.revNodeIdx.first][tar.revNodeIdx.second].flow -=
    tlFlow; // 反向边流量
98                  if (ret == flow)
```

```
                    return ret; // 充满了就不继续扫了
                }
            }
            return ret;
        }

        ll int dinic()
        {
            if (maxFlowAns)
                return maxFlowAns;
            while (bfs())
            {
                for(auto x = fir.begin();x != fir.end();x ++) (*x) = 0;
                maxFlowAns += dfs(S, INT_MAX);
            }
            return maxFlowAns;
        }
};
```

4.1.2. min_Cost.h

```
const int INF = 0x3f3f3f3f

class PD//AC
{
public:
    class edge
    {
    public:
        int v, f, c, next;
        edge(int _v,int _f,int _c,int _next)
        {
            v = _v;
            f = _f;
            c = _c;
            next = _next;
        }
        edge() { }
    } ;

    void vecset(int value,vector<int> &arr)
    {
        for(int i = 0;i < arr.size();i ++) arr[i] = value;
        return;
    }

    class node
    {
    public:
        int v, e;
    } ;

    class mypair
    {
    public:
        int dis, id;

        bool operator<(const mypair &a) const { return dis > a.dis; }

        mypair(int d, int x)
        {
            dis = d;
            id = x;
        }
    };

    vector<int> head;
    vector<int> dis;
    vector<int> vis;
    vector<int> h;
    vector<edge> e;
    vector<node> p;
    int n, m, s, t, cnt = 1, maxf, minc;

    PD(int _n,int _m,int _s,int _t)
    {
        n = _n;
        m = _m;
        s = _s;
```

```
        t = _t;
        maxf = 0;
        minc = 0;
        head.resize(n+2);
        dis.resize(n+2);
        vis.resize(n+2);
        e.resize(2);
        h.resize(n+2);
        p.resize(m+2);
    }

    void addedge(int u, int v, int f, int c)
    {
        e.push_back(edge(v,f,c,head[u]));
        head[u] = e.size()-1;
        e.push_back(edge(u,0,-c,head[v]));
        head[v] = e.size()-1;
    }

    bool dijkstra()
    {
        priority_queue<mypair> q;
        vecset(INF,dis);
        vecset(0,vis);
        dis[s] = 0;
        q.push(mypair(0, s));
        while (!q.empty())
        {
            int u = q.top().id;
            q.pop();
            if (vis[u])
                continue;
            vis[u] = 1;
            for (int i = head[u]; i; i = e[i].next)
            {
                int v = e[i].v, nc = e[i].c + h[u] - h[v];
                if (e[i].f && dis[v] > dis[u] + nc)
                {
                    dis[v] = dis[u] + nc;
                    p[v].v = u;
                    p[v].e = i;
                    if (!vis[v])
                        q.push(mypair(dis[v], v));
                }
            }
        }
        return dis[t] != INF;
    }

    void spfa()
    {
        queue<int> q;
        vecset(63,h);
        h[s] = 0, vis[s] = 1;
        q.push(s);
        while (!q.empty())
        {
            int u = q.front();
```

```
            q.pop();
            vis[u] = 0;
            for (int i = head[u]; i; i = e[i].next)
            {
                int v = e[i].v;
                if (e[i].f && h[v] > h[u] + e[i].c)
                {
                    h[v] = h[u] + e[i].c;
                    if (!vis[v])
                    {
                        vis[v] = 1;
                        q.push(v);
                    }
                }
            }
        }
    }

    int pd()
    {
        spfa();
        while (dijkstra())
        {
            int minf = INF;
            for (int i = 1; i <= n; i++)
                h[i] += dis[i];
            for (int i = t; i != s; i = p[i].v)
                minf = min(minf, e[p[i].e].f);
            for (int i = t; i != s; i = p[i].v)
            {
                e[p[i].e].f -= minf;
                e[p[i].e ^ 1].f += minf;
            }
            maxf += minf;
            minc += minf * h[t];
        }
        return 0;
    }

    void printAns()
    {
        cout << maxf << " " << minc << "\n";
    }
};
```

4.2. TREE

4.2.1. `lca.h`

```cpp
class LCA{
public:
    vector<vector<int>> cnj;
    vector<int> lg,dep;
    vector<array<int,32>> fa;
    int n;

    LCA(int _n) {
        n = _n;
        cnj.resize(n+1);
        lg.resize(n+1),fa.resize(n+1),dep.resize(n+1);
        for(int i = 1; i <= n; i ++)
            lg[i] = lg[i-1] + (1 << lg[i-1] == i);
    }

    void addEdge(int u,int v) {
        cnj[u].push_back(v);
        cnj[v].push_back(u);
    }

    void build(int rt = 1) {
        using itf = function<void(int,int)>;
        itf dfs = [&](int p,int f) -> void {
            fa[p][0] = f,dep[p] = dep[f] + 1;
            for(int i = 1;i <= lg[dep[p]];i ++) fa[p][i] = fa[fa[p][i-1]][i-1];
            for(auto x:cnj[p]) if(x == f) continue;
            else dfs(x,p);
        };
        dfs(rt,0);
    }

    int get(int x,int y) {
        if(dep[x] < dep[y]) swap(x,y);
        while(dep[x] > dep[y]) x = fa[x][lg[dep[x] - dep[y]] - 1];
        if(x == y) return x;
        for(int k = lg[dep[x]]-1;k >= 0;k --) if(fa[x][k] != fa[y][k]) x = fa[x][k],y = fa[y][k];
        return fa[x][0];
    }
};
```

5. MATH

5.1. NUMBER_THEORY

5.1.1. basic.h

```
__builtin_ffsll(x)
// 返回 x 的二进制末尾最后一个 1 的位置

__builtin_clzll(x)
// 返回 x 的二进制的前导 0 的个数。

__builtin_ctzll(x)
// 返回 x 的二进制末尾连续 0 的个数。

__builtin_clrsbll(x)
// 当 x 的符号位为 0 时返回 x 的二进制的前导 0 的个数减一，否则返回 x 的二进制的前导 1 的个数减一。

__builtin_popcountll(x)
// 返回 x 的二进制中 1 的个数。

__builtin_parity(x)
// 判断 x 的二进制中 1 的个数的奇偶性。

int binpow(int x, int y)
{
    int res = 1;
    while (y > 0)
    {
        if (y & 1)
            res = res * x % mod;
        x = x * x % mod;
        y >>= 1;
    }
    return res;
}

void exgcd(int a, int b, int& x, int& y) {
  if (b == 0) {
    x = 1, y = 0;
    return;
  }
  exgcd(b, a % b, y, x);
  y -= a / b * x;
}

binpow(x,mod-2)
```

5.1.2. Comb.h

```cpp
const int N = 1e6;
const int mod = 1e9+7;

int binpow(int x, int y)
{
    int ans = 1;
    while (y)
    {
        if (y & 1) ans = ans * x % mod;
        x = x * x % mod;
        y >>= 1;
    }
    return ans;
}

vector<int> fac(N), inv(N);

void init()
{
    fac[0] = inv[0] = 1;
    for (int i = 1; i < N; i++) fac[i] = fac[i - 1] * i % mod;
    inv[N - 1] = binpow(fac[N - 1], mod - 2);
    for (int i = N - 2; i >= 1; i--)
    {
        inv[i] = inv[i + 1] * (i + 1) % mod;
    }
}

auto C = [&](int x, int y) -> int
{
    return (fac[x] * inv[y] % mod) * inv[x - y] % mod;
};
```

5.1.3. CRT.h

```cpp
int CRT(vector<int> &r, vector<int> &a)
{ // % r === a
    int n = a.size();
    __int128 k = 1, ans = 0;
    for (int i = 0; i < n; i++) k *= r[i];
    for (int i = 0; i < n; i++)
    {
        __int128 m = k / r[i];
        int b, y;
        exgcd(m, r[i], b, y); // b * m mod r[i] = 1
        ans = (ans + a[i] * m * b % k) % k;
    }
    return (ans % k + k) % k;
}



int mul(int a, int b, int m) {
    return (__int128)a * b % m;
}

int exgcd (int a,int b,int &x,int &y) {
    if (b == 0) { x = 1, y = 0; return a; }
    int g = exgcd(b, a % b, x, y), tp = x;
    x = y, y = tp - a / b * y;
    return g;
};

int EXCRT(vector<int> &a,vector<int> &r) { // % r == a
    int x, y, k;
    int n = r.size();
    int M = a[0], ans = r[0];
    for (int i = 1; i < n; ++ i) {
        int ca = M, cb = a[i], cc = (r[i] - ans % cb + cb) % cb;
        int gcd = exgcd(ca, cb, x, y), bg = cb / gcd;
        if (cc % gcd != 0) return -1;
        x = mul(x, cc / gcd, bg);
        ans += x * M;
        M *= bg;
        ans = (ans % M + M) % M;
    }
    return (ans % M + M) % M;
}
```

5.1.4. `Eular_phi.h`

```cpp
int euler_phi(int n) {
  int ans = n;
  for (int i = 2; i * i <= n; i++)
    if (n % i == 0) {
      ans = ans / i * (i - 1);
      while (n % i == 0) n /= i;
    }
  if (n > 1) ans = ans / n * (n - 1);
  return ans;
}
```

5.1.5. `Eular_sieve.h`

```cpp
vector<int> init(int n)
{
    vector<int> pri;
    vector<bool> vis(n, 0);
    for (int i = 2; i <= n; i++)
    {
        if (!vis[i])
            pri.push_back(i);
        for (int j = 0; j < pri.size(); j++)
        {
            if (i * pri[j] > n)
                break;
            vis[pri[j] * i] = 1;
            if (i % pri[j] == 0)
                break;
        }
    }
    return pri;
}
```

5.1.6. factor_pr.h

```cpp
#define int long long
#define pii pair<int, int>
const int INF = 1145141919810LL;
using namespace std;

class Pollard_Rho
{
private:

    vector<int> B;

    int mul(int a, int b, int m)
    {
        int r = a * b - m * (int)(1.L / m * a * b);
        return r - m * (r >= m) + m * (r < 0);
    }

    int mypow(int a, int b, int m)
    {
        int res = 1 % m;
        for (; b; b >>= 1, a = mul(a, a, m))
        {
            if (b & 1)
            {
                res = mul(res, a, m);
            }
        }
        return res;
    }

    bool MR(int n)
    {
        if (n <= 1)
            return 0;
        for (int p : B)
        {
            if (n == p)
                return 1;
            if (n % p == 0)
                return 0;
        }
        int m = (n - 1) >> __builtin_ctz(n - 1);
        for (int p : B)
        {
            int t = m, a = mypow(p, m, n);
            while (t != n - 1 && a != 1 && a != n - 1)
            {
                a = mul(a, a, n);
                t *= 2;
            }
            if (a != n - 1 && t % 2 == 0)
                return 0;
        }
        return 1;
    }

    inline const int getfecsum(int _n)
```

```
    {
        int sum = 0;
        while (_n)
        {
            sum += _n % 10;
            _n /= 10;
        }
        return sum;
    };

    int PR(int n)
    {
        for (int p : B)
        {
            if (n % p == 0)
                return p;
        }
        auto f = [&](int x) -> int
        {
            x = mul(x, x, n) + 1;
            return x >= n ? x - n : x;
        };
        int x = 0, y = 0, tot = 0, p = 1, q, g;
        for (int i = 0; (i & 255) || (g = gcd(p, n)) == 1; i++, x = f(x), y = f(f(y)))
        {
            if (x == y)
            {
                x = tot++;
                y = f(x);
            }
            q = mul(p, abs(x - y), n);
            if (q)
                p = q;
        }
        return g;
    }

    vector<int> fac(int n)
    {
        // if(n == 0)
        // #define pb emplace_back
        if (n <= 1)
            return {};
        if (MR(n))
            return {n};
        int d = PR(n);
        auto v1 = fac(d), v2 = fac(n / d);
        auto i1 = v1.begin(), i2 = v2.begin();
        vector<int> ans;
        while (i1 != v1.end() || i2 != v2.end())
        {
            if (i1 == v1.end())
            {
                ans.pb(*i2++);
            }
            else if (i2 == v2.end())
            {
```

```
                ans.pb(*i1++);
            }
            else
            {
                if (*i1 < *i2)
                {
                    ans.pb(*i1++);
                }
                else
                {
                    ans.pb(*i2++);
                }
            }
        }
        return ans;
    }

public:

    Pollard_Rho(){
        B = {2, 3, 5, 7, 11, 13, 17, 19, 23};
    }

    vector<pii> fac_Comp(int n)
    {
        auto srt = fac(n);
        map<int, int> cnt;
        for (auto x : srt)
            cnt[x]++;
        vector<pii> rt;
        for (auto x : cnt)
            rt.push_back(x);
        return rt;
    }

    vector<int> fac_pri(int n)
    {
        return fac(n);
    }

    vector<int> fac_all(int n)
    {
        vector<pii> rt = fac_Comp(n);
        vector<int> v;
        function<void(int, int)> dfs = [&](int id, int x)
        {
            if (id == rt.size())
            {
                v.push_back(x);
                return;
            }
            for(int i = 0;i <= rt[id].se;i ++)
            {
                dfs(id + 1, x * (mypow(rt[id].fi, i, INF)));
            }
        };
        dfs(0, 1);
        return v;
```

```
    }
};
```

5.2. OTHER

5.2.1. `Frac.h`

```cpp
template<class T>
struct Frac {
    T num;
    T den;
    Frac(T num_, T den_) : num(num_), den(den_) {
        if (den < 0) {
            den = -den;
            num = -num;
        }
    }
    Frac() : Frac(0, 1) {}
    Frac(T num_) : Frac(num_, 1) {}
    explicit operator double() const {
        return 1. * num / den;
    }
    Frac &operator+=(const Frac &rhs) {
        num = num * rhs.den + rhs.num * den;
        den *= rhs.den;
        return *this;
    }
    Frac &operator-=(const Frac &rhs) {
        num = num * rhs.den - rhs.num * den;
        den *= rhs.den;
        return *this;
    }
    Frac &operator*=(const Frac &rhs) {
        num *= rhs.num;
        den *= rhs.den;
        return *this;
    }
    Frac &operator/=(const Frac &rhs) {
        num *= rhs.den;
        den *= rhs.num;
        if (den < 0) {
            num = -num;
            den = -den;
        }
        return *this;
    }
    friend Frac operator+(Frac lhs, const Frac &rhs) {
        return lhs += rhs;
    }
    friend Frac operator-(Frac lhs, const Frac &rhs) {
        return lhs -= rhs;
    }
    friend Frac operator*(Frac lhs, const Frac &rhs) {
        return lhs *= rhs;
    }
    friend Frac operator/(Frac lhs, const Frac &rhs) {
        return lhs /= rhs;
    }
    friend Frac operator-(const Frac &a) {
        return Frac(-a.num, a.den);
    }
```

```cpp
    friend bool operator==(const Frac &lhs, const Frac &rhs) {
        return lhs.num * rhs.den == rhs.num * lhs.den;
    }
    friend bool operator!=(const Frac &lhs, const Frac &rhs) {
        return lhs.num * rhs.den != rhs.num * lhs.den;
    }
    friend bool operator<(const Frac &lhs, const Frac &rhs) {
        return lhs.num * rhs.den < rhs.num * lhs.den;
    }
    friend bool operator>(const Frac &lhs, const Frac &rhs) {
        return lhs.num * rhs.den > rhs.num * lhs.den;
    }
    friend bool operator<=(const Frac &lhs, const Frac &rhs) {
        return lhs.num * rhs.den <= rhs.num * lhs.den;
    }
    friend bool operator>=(const Frac &lhs, const Frac &rhs) {
        return lhs.num * rhs.den >= rhs.num * lhs.den;
    }
    friend std::ostream &operator<<(std::ostream &os, Frac x) {
        T g = std::gcd(x.num, x.den);
        if (x.den == g) {
            return os << x.num / g;
        } else {
            return os << x.num / g << "/" << x.den / g;
        }
    }
};

using F = Frac<int>;
```

6. STRING

6.1. compress_print.h

```cpp
const int N = 1 << 21;
static const int mod1 = 1E9 + 7, base1 = 127;
static const int mod2 = 1E9 + 9, base2 = 131;
vector<int> val1;
vector<int> val2;
void init(int n = N)
{
    val1.resize(n + 1), val2.resize(n + 2);
    val1[0] = 1, val2[0] = 1;
    for (int i = 1; i <= n; i++)
    {
        val1[i] = val1[i - 1] * base1 % mod1;
        val2[i] = val2[i - 1] * base2 % mod2;
    }
}

string compress(vector<string> in)
{ // 前后缀压缩
    vector<int> h1{1};
    vector<int> h2{1};
    string ans = "#";
    for (auto s : in)
    {
        s = "#" + s;
        int st = 0;
        int chk1 = 0;
        int chk2 = 0;
        for (int j = 1; j < s.size() && j < ans.size(); j++)
        {
            chk1 = (chk1 * base1 % mod1 + s[j]) % mod1;
            chk2 = (chk2 * base2 % mod2 + s[j]) % mod2;
            if ((h1.back() == (h1[ans.size() - 1 - j] * val1[j] % mod1+ chk1) % mod1) &&
                (h2.back() == (h2[ans.size() - 1 - j] * val2[j] % mod2+ chk2) % mod2)    )
                st = j;
        }
        for (int j = st + 1; j < s.size(); j++)
        {
            ans += s[j];
            h1.push_back((h1.back() * base1 % mod1 + s[j]) % mod1);
            h2.push_back((h2.back() * base2 % mod2 + s[j]) % mod2);
        }
    }
    return ans.substr(1);
}
```

6.2. get_occr.h

```cpp
#include <template_overAll.h>

/*
 * 找到某一堆短字符串在长字符串中的出现位置
 * dira=1 为最早出现的后端点下标 dira=0 为最晚出现的前端点下标
 * 源字符串 s 长度为|s|，查找字符串列表中所有字符串长度和为|_s|
 * 则时间复杂度为 O(max(|_s|log(|_s|),|s|))
 */
class get_occr
{
private:
    string s;
public:
    get_occr(string _s) { s = _s; }
    vector<int> locate(vector<string> _s,bool dira = 1)
    {
        int n = _s.size();
        vector<int> occr(n,-1);
        map<char,vector<pair<int,int>>> gncing;
        if(dira == 1)
        {
            for(int i = 0;i < n;i++)
                gncing[_s[i][0]].push_back({i,0});
            for(int i = 0;i < s.size();i ++)
            {
                vector<pair<int,int>> gnctmp = gncing[s[i]];
                gncing[s[i]].clear();
                for(int j = 0;j < gnctmp.size();j ++)
                {
                    if(gnctmp[j].se+1 < _s[gnctmp[j].fi].size())
                            gncing[_s[gnctmp[j].fi]
[gnctmp[j].se+1]].push_back({gnctmp[j].fi,gnctmp[j].se+1});
                    else occr[gnctmp[j].fi] = i;
                }
            }
        } else {
            for(int i = 0;i < n;i++) gncing[_s[i]
[_s[i].size()-1]].push_back({i,_s[i].size()-1});
            for(int i= s.size()-1;i >=0;i --)
            {
                vector<pair<int,int>> gnctmp = gncing[s[i]];
                gncing[s[i]].clear();
                for(int j = 0;j < gnctmp.size();j ++)
                {
                    if(gnctmp[j].se -1 >= 0)
                            gncing[_s[gnctmp[j].fi]
[gnctmp[j].se-1]].push_back({gnctmp[j].fi,gnctmp[j].se-1});
                    else occr[gnctmp[j].fi] = i;
                }
            }
        }
        return occr;
    }
};
```

6.3. hash_print.h

```cpp
#define int long long
const int N = 1 << 21;
static const int mod1 = 1E9 + 7, base1 = 127;
static const int mod2 = 1E9 + 9, base2 = 131;
vector<int> val1;
vector<int> val2;
using puv = pair<int,int>;
void init(int n = N)
{
    val1.resize(n + 1), val2.resize(n + 2);
    val1[0] = 1, val2[0] = 1;
    for (int i = 1; i <= n; i++)
    {
        val1[i] = val1[i - 1] * base1 % mod1;
        val2[i] = val2[i - 1] * base2 % mod2;
    }
}
class hstring
{
public:
    vector<int> h1;
    vector<int> h2;
    string s;

    hstring(string s_) : s(s_), h1{1}, h2{1}
    {
        build();
    }

    void build()
    {
        for (auto it : s)
        {
            h1.push_back((h1.back() * base1 % mod1 + it) % mod1);
            h2.push_back((h2.back() * base2 % mod2 + it) % mod2);
        }
    }

    puv get()
    { // 输出整串的哈希值
        return {h1.back(), h2.back()};
    }

    puv substring(int l, int r)
    { // 输出子串的哈希值
        if (l > r) swap(l, r);
        int ans1 = (mod + h1[r + 1] - h1[l] * val1[r - l + 1] % mod1) % mod1;
        int ans2 = (mod + h2[r + 1] - h2[l] * val2[r - l + 1] % mod2) % mod2;
        return {ans1, ans2};
    }

    puv modify(int idx, char x)
    { //修改 idx 位为 x
        int n = s.size() - 1;
        int ans1 = (h1.back() + val1[n - idx] * (x - s[idx]) % mod1) % mod1;
        int ans2 = (h2.back() + val2[n - idx] * (x - s[idx]) % mod2) % mod2;
        return {ans1, ans2};
```

```
58         }
59 };
```

6.4. KMP.h

```cpp
#include <template_overAll.h>

class KMP
{
private:
    string s;
    string inis;
public:
    vector<int> pi;
    KMP(string _s)
    {
        s = _s;
        inis = s;
    }
    void prefix_function()
    {
        pi.clear();
        int n = (int)s.length();
        pi.resize(n);
        for (int i = 1; i < n; i++)
        {
            int j = pi[i - 1];
            while (j > 0 && s[i] != s[j])
                j = pi[j - 1];
            if (s[i] == s[j])
                j++;
            pi[i] = j;
        }
        return;
    }
    vector<int> find_occr(string p)
    {
        s = inis;
        s = p + "#" + s;
        prefix_function();
        vector<int> v;
        for (int i = p.size() + 1; i < s.size(); i++)
            if (pi[i] == p.size())
                v.push_back(i - 2 * p.size());
        return v;
    }
};
```

6.5. trie_Tree.h

```cpp
#include <template_overAll.h>

class Trie//AC
{
public:
    vector<map<char, int>> t;
    int root = 0;
    Trie()
    {
        t.resize(1);
    }
    void addedge(string _s)
    {
        int pvidx = root;
        _s.push_back('-');
        for (int i = 0; i < _s.size(); i++)
        {
            if (t[pvidx].find(_s[i]) != t[pvidx].end())
            {
                pvidx = t[pvidx][_s[i]];
            }
            else
            {
                t[pvidx][_s[i]] = t.size();
                t.push_back(map<char, int>());
                pvidx = t[pvidx][_s[i]];
            }
        }
    }
    bool ifcmp(string &s)
    {
        int pvidx = root;
        for(int i = 0;i < s.size();i ++)
        {
            if(t[pvidx].find(s[i]) != t[pvidx].end()) pvidx = t[pvidx][s[i]];
            else return 0;
        }
        return t[pvidx].find('-') != t[pvidx].end();
    }
};
```