ICPC-ACM Template

Ruoyisius

2023 年 10 月 9 日

目录

一、 图论

1.1 增广路

```cpp
struct augment_path {
    vector<vector<int> > g;
    vector<int> pa; // 匹配
    vector<int> pb;
    vector<int> vis; // 访问
    int n, m;       // 两个点集中的顶点数量
    int dfn;        // 时间戳记
    int res;        // 匹配数

    augment_path(int _n, int _m) : n(_n), m(_m) {
        assert(0 <= n && 0 <= m);
        pa = vector<int>(n, -1);
        pb = vector<int>(m, -1);
        vis = vector<int>(n);
        g.resize(n);
        res = 0;
        dfn = 0;
    }

    void add(int from, int to) {
        assert(0 <= from && from < n && 0 <= to && to < m);
        g[from].push_back(to);
    }

    bool dfs(int v) {
        vis[v] = dfn;
        for (int u : g[v]) {
            if (pb[u] == -1) {
                pb[u] = v;
                pa[v] = u;
                return true;
            }
        }
        for (int u : g[v]) {
            if (vis[pb[u]] != dfn && dfs(pb[u])) {
                pa[v] = u;
                pb[u] = v;
                return true;
            }
        }
        return false;
    }
    int solve() {
        while (true) {
            dfn++;
            int cnt = 0;
            for (int i = 0; i < n; i++) {
```

```
                if (pa[i] == -1 && dfs(i)) {
                cnt++;
                }
            }
            if (cnt == 0){
                break;
            }
            res += cnt;
        }
        return res;
    }
};
```

1.2 SPFA

```
/*
 * Args:
 *   g[]: graph, (u, v, w) = (u, g[u][i].first, g[u][i].second)
 *   st: source vertex
 * Return:
 *   dis[]: distance from source vertex to each other vertex
 */
vector<pair<int, int> > g[N];
int dis[N], vis[N];
void spfa(int st)
{
  memset(dis, -1, sizeof(dis));
  memset(vis, 0, sizeof(vis));
  queue<int> q;
  q.push(st);
  dis[st] = 0;
  vis[st] = true;
  while (!q.empty()) {
    int u = q.front();
    q.pop();
    vis[u] = false;
    for (auto x : g[u]) {
      int v = x.first, w = x.second;
      if (dis[v] == -1 || dis[u] + w < dis[v]) {
        dis[v] = dis[u] + w;
        if (!vis[v]) {
          vis[v] = true;
          q.push(v);
        }
      }
    }
  }
}
```

1.3 Tarjan

```
vector<int> dfn(n, -1), low(n, -1), be(n, -1);
int tot = 0, cnt = 0;
vector<int> st;
function<void(int)> tarjan = [&](int cur) {
    dfn[cur] = low[cur] = tot++;
    st.push_back(cur);
    for (auto &nex : g[cur]) {
        if (dfn[nex] == -1) {
            tarjan(nex);
            low[cur] = min(low[cur], low[nex]);
        } else if (be[nex] == -1) {
            low[cur] = min(low[cur], dfn[nex]);
        }
    }
    if (dfn[cur] == low[cur]) {
        int v;
        do {
            v = st.back();
            st.pop_back();
            be[v] = cnt;
        } while (v != cur);
        cnt++;
    }
};

for (int i = 0; i < n; i++) {
    if (dfn[i] == -1) {
        tarjan(i);
    }
}
```

1.4 基环树

```
vector<vector<int>> E(n);
vector<int> deg(n);
for (int j = 0; j < n; j++) {
    deg[j] = E[j].size();
}
queue<int> Q;
for (int j = 0; j < n; j++) {
    if (deg[j] == 1) {
        Q.push(j);
    }
}
vector<bool> used(n, false);
while (!Q.empty()) {
```

```
    int v = Q.front();
    Q.pop();
    used[v] = true;
    for (int w : E[v]) {
        deg[w]--;
        if (deg[w] == 1) {
            Q.push(w);
        }
    }
}
vector<int> p(n, -1);
vector<int> d(n, -1);
for (int j = 0; j < n; j++) {
    if (!used[j]) {
        d[j] = 0;
        Q.push(j);
    }
}
while (!Q.empty()) {
    int v = Q.front();
    Q.pop();
    for (int w : E[v]) {
        if (d[w] == -1) {
            d[w] = d[v] + 1;
            p[w] = v;
            Q.push(w);
        }
    }
}
```

二、 网络流

2.1 Dinic

```cpp
struct Edge {
    int from, to, cap, flow;
};
struct Dinic {
    int s, t;
    bool vis[N];
    int d[N];
    int cur[N];

    vector<Edge> edges;
    vector<int> G[N];
    void AddEdge(int from, int to, int cap) {
        edges.push_back({from, to, cap, 0});
        edges.push_back({to, from, 0, 0});
        //edges.push_back({to, from, cap, 0}); 如果是无向图.
        G[from].push_back(edges.size() - 2);
        G[to].push_back(edges.size() - 1);
    }

    bool BFS() {
        memset(vis, 0, sizeof(vis));
        queue<int> Q;
        Q.push(s);
        d[s] = 0;
        vis[s] = 1;
        while (!Q.empty()) {
            int u = Q.front(); Q.pop();
            for (int i = 0; i < G[u].size(); i++) {
                Edge& e = edges[G[u][i]];
                int v = e.to;
                if (!vis[v] && e.cap > e.flow) {
                    vis[v] = 1;
                    d[v] = d[u] + 1;
                    Q.push(v);
                }
            }
        }
        return vis[t];
    }

    int DFS(int u, int a) {
        if (u == t || a == 0) return a;
        int flow = 0, f;
        for (int& i = cur[u]; i < G[u].size(); i++) { //这里取引用，使得u的当前弧被i改变，再次访问到u时，将跳过u已经访问过的支路
            Edge& e = edges[G[u][i]], ee = edges[G[u][i] ^ 1];
            int v = e.to;
```

```
47              if (d[v] == d[u] + 1 && (f = DFS(v, min(a, e.cap - e.flow))) > 0) {
48                  e.flow += f;
49                  ee.flow -= f;
50                  flow += f;
51                  a -= f;
52                  if (a == 0) break;
53              }
54          }
55          return flow;
56      }
57
58      int Maxflow(int s, int t) {
59          this->s = s; this->t = t;
60          int flow = 0;
61          while (BFS()) {
62              memset(cur, 0, sizeof(cur));
63              flow += DFS(s, INF);
64          }
65          return flow;
66      }
67  };
```

三、 数据结构

3.1 并查集

```cpp
struct DSU {
    std::vector<int> f, siz;

    DSU() {}
    DSU(int n) {
        init(n);
    }

    void init(int n) {
        f.resize(n);
        std::iota(f.begin(), f.end(), 0);
        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)];
    }
};
```

3.2 树状数组

```cpp
template <typename T>
```

```
struct Fenwick {
    int n;
    std::vector<T> a;

    Fenwick(int n = 0) {
        init(n);
    }

    void init(int n) {
        this->n = n;
        a.assign(n, T());
    }

    void add(int x, T v) {
        for (int i = x + 1; i <= n; i += i & -i) {
            a[i - 1] += v;
        }
    }

    T sum(int x) {
        auto ans = T();
        for (int i = x; i > 0; i -= i & -i) {
            ans += a[i - 1];
        }
        return ans;
    }

    T rangeSum(int l, int r) {
        return sum(r) - sum(l);
    }

    int kth(T k) {
        int x = 0;
        for (int i = 1 << std::__lg(n); i; i /= 2) {
            if (x + i <= n && k >= a[x + i - 1]) {
                x += i;
                k -= a[x - 1];
            }
        }
        return x;
    }
};
```

3.3 线段树

```
#include<bits/stdc++.h>
using i64=long long;
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();
        }
```

```
57          if (l >= x && r <= y) {
58              return info[p];
59          }
60          int m = (l + r) / 2;
61          return rangeQuery(2 * p, l, m, x, y) + rangeQuery(2 * p + 1, m, r, x, y
                );
62      }
63      Info rangeQuery(int l, int r) {
64          return rangeQuery(1, 0, n, l, r);
65      }
66      template<class F>
67      int findFirst(int p, int l, int r, int x, int y, F pred) {
68          if (l >= y || r <= x || !pred(info[p])) {
69              return -1;
70          }
71          if (r - l == 1) {
72              return l;
73          }
74          int m = (l + r) / 2;
75          int res = findFirst(2 * p, l, m, x, y, pred);
76          if (res == -1) {
77              res = findFirst(2 * p + 1, m, r, x, y, pred);
78          }
79          return res;
80      }
81      template<class F>
82      int findFirst(int l, int r, F pred) {
83          return findFirst(1, 0, n, l, r, pred);
84      }
85      template<class F>
86      int findLast(int p, int l, int r, int x, int y, F pred) {
87          if (l >= y || r <= x || !pred(info[p])) {
88              return -1;
89          }
90          if (r - l == 1) {
91              return l;
92          }
93          int m = (l + r) / 2;
94          int res = findLast(2 * p + 1, m, r, x, y, pred);
95          if (res == -1) {
96              res = findLast(2 * p, l, m, x, y, pred);
97          }
98          return res;
99      }
100     template<class F>
101     int findLast(int l, int r, F pred) {
102         return findLast(1, 0, n, l, r, pred);
103     }
104 };
105
106 constexpr i64 inf = 1E18;
107
```

```
struct Info {
    i64 cnt = 0;
    i64 sum = 0;
    i64 min = inf;
};

Info operator+(Info a, Info b) {
    Info c;
    c.cnt = a.cnt + b.cnt;
    c.sum = a.sum + b.sum;
    c.min = std::min(a.min, b.min);
    return c;
}
```

3.4 懒标记线段树

```
const int N = 1e5 + 5;
int n, m, a[N], t[N<<2], x, y, w, lazy[N<<2];
void build(int rt, int l, int r) {
    if (l == r) { t[rt] = a[l]; return; }
    int mid = (l + r) >> 1;
    build(rt << 1, l, mid);
    build(rt << 1 | 1, mid + 1, r);
    t[rt] = t[rt<<1] + t[rt<<1|1];
}
void updata(int rt, int l, int r, int w) {
    t[rt] += (r - l + 1) * w;
    lazy[rt] += w;
}
void pushdown(int rt, int l, int r) {
    int mid = (l + r) >> 1;
    updata(rt << 1, l, mid, lazy[rt]);
    updata(rt << 1 | 1, mid + 1, r, lazy[rt]);
    lazy[rt] = 0;
}
void add(int rt, int l, int r) {
    if (x <= l && r <= y) {
        updata(rt, l, r, w);
        return;
    }
    pushdown(rt, l, r);
    int mid = (l + r) >> 1;
    if (x <= mid) add(rt << 1, l, mid);
    if (y > mid) add(rt << 1 | 1, mid + 1, r);
    t[rt] = t[rt<<1] + t[rt<<1|1];
}
int sum(int rt, int l, int r) {
    if (x <= l && r <= y) return t[rt];
    int mid = (l + r) >> 1, ans = 0;
```

```
34      pushdown(rt, l, r);
35      if (x <= mid) ans += sum(rt << 1, l, mid);
36      if (y > mid) ans += sum(rt << 1 | 1, mid + 1, r);
37      return ans;
38  }
```

3.5 倍增 LCA-最近公共祖先

```
1   void bfs()
2   {
3       dep[1]=1;
4       que[tail++]=1;
5       while(head<tail)
6       {
7           int p=que[head++];
8           for(int x=last[p];x!=0;x=pre[x])
9           {
10              if(f[p][0]!=son[x])
11              {
12                  dep[son[x]]=dep[p]+1;
13                  f[son[x]][0]=p;
14                  que[tail++]=son[x];
15                  for(int j=1;j<=20;j++)
16                      f[son[x]][j]=f[f[son[x]][j-1]][j-1];
17              }
18          }
19      }
20  }
21  int lca(int a,int b)
22  {
23      if(dep[a]>dep[b]) swap(a,b);
24      for(int i=20;i>=0;i--)
25      {
26          if(dep[f[b][i]]>=dep[a]) b=f[b][i];
27          if(a==b) return a;
28      }
29      for(int i=20;i>=0;i--)
30          if(f[a][i]!=f[b][i]) a=f[a][i],b=f[b][i];
31      return f[a][0];
32  }
```

3.6 珂朵莉树

```
1   struct node {
2       int l, r;
3       mutable int v;
4       node(int a = 0, int b = 0, int c = 0) :l(a), r(b), v(c) { };
```

```
    friend bool operator<(const node& a, const node& b) {
        return a.l < b.l;
    }
};

class ODT :public set<node> {
public:
    ODT() :st(*this) { };
    ODT(int l, int r, int v) :set<node>({ node(l,r,v) }), st(*this) { };
    set<node>& st;

    set<node>::iterator split(int pos) {
        auto it = --st.upper_bound(node{ pos,0,0 });
        if (it->l == pos) return it;
        int l = it->l, r = it->r, v = it->v;
        st.erase(it);
        st.insert(node(l, pos - 1, v));
        return st.insert(node(pos, r, v)).first;
    }

    void assign(int l, int r, int v) {
        auto itr = split(r + 1), itl = split(l);
        st.erase(itl, itr);
        st.insert(node(l, r, v));
    }
};
```

3.7 ST

```
template <typename T>
class SparseTable {
  using VT = vector<T>;
  using VVT = vector<VT>;
  using func_type = function<T(const T &, const T &)>;

  VVT ST;
  VT A;

  static T default_func(const T &t1, const T &t2) { return max(t1, t2); }

  func_type op;

 public:
  SparseTable(const vector<T> &v, func_type _func = default_func) {
    op = _func;
    A = v;
    int len = v.size(), l1 = ceil(log2(len)) + 1;
    ST.assign(len, VT(l1, 0));
    for (int i = 0; i < len; ++i) {
      ST[i][0] = v[i];
```

```
    }
    for (int j = 1; j < l1; ++j) {
      int pj = (1 << (j - 1));
      for (int i = 0; i + pj < len; ++i) {
        ST[i][j] = op(ST[i][j - 1], ST[i + (1 << (j - 1))][j - 1]);
      }
    }
  }

  T query(int l, int r) {
    if(l == r) return A[l];
    int lt = r - l + 1;
    int q = ceil(log2(lt)) - 1;
    return op(ST[l][q], ST[r - (1 << q) + 1][q]);
  }
};
```

3.8 支配树

```
struct DominatorTree {
    int n, cs;
    std::vector<std::vector<int>> E, RE, rdom;
    std::vector<int> S, RS, par, val, sdom, rp, dom;

    DominatorTree(int n) {
        this->cs = 0;
        this->n = n;
        E.resize(n + 1);
        RE.resize(n + 1);
        rdom.resize(n + 1);
        S.resize(n + 1);
        RS.resize(n + 1);
        par.resize(n + 1);
        val.resize(n + 1);
        sdom.resize(n + 1);
        rp.resize(n + 1);
        dom.resize(n + 1);
        for (int i = 0; i <= n; i++)
        {
            par[i] = val[i] = sdom[i] = rp[i] = dom[i] = S[i] = RS[i] = 0;
            E[i].clear();
            RE[i].clear();
            rdom[i].clear();
        }
    }

    void add_edge(int x, int y) {
        E[x].push_back(y);
    }
```

```
31
32      void Union(int x, int y) {
33          par[x] = y;
34      }
35
36      int Find(int x, int c = 0) {
37          if (par[x] == x)
38              return c ? -1 : x;
39          int p = Find(par[x], 1);
40          if (p == -1)
41              return c ? par[x] : val[x];
42          if (sdom[val[x]] > sdom[val[par[x]]])
43              val[x] = val[par[x]];
44          par[x] = p;
45          return c ? p : val[x];
46      }
47
48      void dfs(int x) {
49          RS[S[x] = ++cs] = x;
50          par[cs] = sdom[cs] = val[cs] = cs;
51          for (int e : E[x]) {
52              if (S[e] == 0)
53                  dfs(e), rp[S[e]] = S[x];
54              RE[S[e]].push_back(S[x]);
55          }
56      }
57
58      int solve(int s, std::vector<int>& up) {
59          dfs(s);
60          for (int i = cs; i; i--) {
61              for (int e : RE[i])
62                  sdom[i] = std::min(sdom[i], sdom[Find(e)]);
63              if (i > 1)
64                  rdom[sdom[i]].push_back(i);
65              for (int e : rdom[i]) {
66                  int p = Find(e);
67                  if (sdom[p] == i)
68                      dom[e] = i;
69                  else
70                      dom[e] = p;
71              }
72              if (i > 1)
73                  Union(i, rp[i]);
74          }
75          for (int i = 2; i <= cs; i++)
76              if (sdom[i] != dom[i])
77                  dom[i] = dom[dom[i]];
78          for (int i = 2; i <= cs; i++)
79              up[RS[i]] = RS[dom[i]];
80          return cs;
81      }
82  };
```

四、 计算几何

4.1 点

```
using real = double;
using segment_type = tuple<real, real, real, real>;
using point_type = tuple <real, real>;
const auto p2p_distance{[&](const point_type& a, const point_type& b) {
    const auto & [x, y] = a;
    const auto & [u, v] = b;
    return hypot(x - u, y - v);
}};
const auto p2s_distance{[&](const point_type& p, const segment_type& s) {
    const auto [x, y]{p};
    const auto [a, b, c, d]{s};
    if ((x - a) * (c - a) + (y - b) * (d - b) < 0)
        return p2p_distance({a, b}, {x, y});
    if ((x - c) * (a - c) + (y - d) * (b - d) < 0)
        return p2p_distance({c, d}, {x, y});
    real cross = (a - c) * (b - y) - (a - x) * (b - d);
    return abs(cross) / p2p_distance({a, b}, {c, d});
}};
```

五、 字符串

5.1 Hash

```cpp
struct Shash{
    const ll base[2]={29,31};
    const ll hashmod[2]={(ll)1e9,998244353};

    array<vector<ll>,2>hsh,pwMod;
    void init(string &s){
        int n=s.size();s='␣'+s;
        hsh[0].resize(n+1),hsh[1].resize(n+1);
        pwMod[0].resize(n+1),pwMod[1].resize(n+1);
        for(int i=0;i<2;i++){
            pwMod[i][0]=1;
            for(int j=1;j<=n;j++){
                pwMod[i][j]=pwMod[i][j-1]*base[i]%hashmod[i];
                hsh[i][j]=(hsh[i][j-1]*base[i]+s[j])%hashmod[i];
            }
        }
    }
    pair<ll,ll>get(int l,int r){
        pair<ll,ll>ans;
        ans.fi=(hsh[0][r]-hsh[0][l-1]*pwMod[0][r-l+1])%hashmod[0];
        ans.se=(hsh[1][r]-hsh[1][l-1]*pwMod[1][r-l+1])%hashmod[1];
        ans.fi=(ans.fi+hashmod[0])%hashmod[0];
        ans.se=(ans.se+hashmod[1])%hashmod[1];
        return ans;
    }
    bool same(int la,int ra,int lb,int rb){
        return get(la,ra)==get(lb,rb);
    }
};
```

5.2 AC 自动机

```cpp
struct AC {
  int n, tot, alp;
  std::vector<int> fail;
  std::vector<std::vector<int>> tr;
  AC() {}
  AC(int n, int m = 26) {
    alp = m;
    fail.resize(n);
    tr.resize(n);
    fail.assign(n, 0);
    for (int i = 0; i < n; i++) {
      tr[i].assign(m, 0);
```

```
    }
    init();
  }
  void init() {
     tot = -1, new_node();
  }
  int new_node() { return ++tot, fail[tot] = 0, tr[tot].assign(tr[tot].size(),
      0), tot; }
  void insert(const std::string& s) {
    for (int i = 0, u = 0; i < s.size(); i++) {
      int c = s[i];
      if(!tr[u][c]) tr[u][c] = new_node();
      u = tr[u][c];
    }
  }
  void build() {
    std::queue<int> q;
    int ql = 1, qr = 0;
    for (int i = 0; i < alp; i++) {
      if (tr[0][i]) {
        q.push(tr[0][i]);
      }
    }
    while(!q.empty()) {
      int u = q.front();
      q.pop();
      for (int c = 0; c < alp; c++) {
        if (tr[u][c]) fail[tr[u][c]] = tr[fail[u]][c], q.push(tr[u][c]);
        else tr[u][c] = tr[fail[u]][c];
      }
    }
  }
};
```

5.3 KMP& EXKMP

```
struct KMP {
    std::vector<int> fail;
    std::string pattern;
    KMP() {}

    KMP(const std::string& p) {
        init(p);
    }

    void init(const std::string& p) {
        pattern = p;
        fail.resize(pattern.size() + 1);
        fail[0] = -1; // 失配数组第一个元素为-1
        int j = -1; // j表示失配数组的值
```

```
15          for (int i = 0; i < pattern.size(); i++) { // 遍历模式串
16              while (j >= 0 && pattern[i] != pattern[j]) { // 如果失配，则回溯
17                  j = fail[j]; // 回溯到失配位置的失配数组值
18              }
19              j++; // 失配数组值加1
20              fail[i + 1] = j; // 更新失配数组
21          }
22      }
23      // 匹配函数，返回所有匹配位置
24      std::vector<int> match(const std::string& s) {
25          std::vector<int> res;
26          int j = 0;
27          for (int i = 0; i < s.size(); i++) { // 遍历文本串
28              while (j >= 0 && s[i] != pattern[j]) { // 如果失配，则回溯
29                  j = fail[j]; // 回溯到失配位置的失配数组值
30              }
31              // dbg(i, j, s[i], pattern[j], pattern.size());
32              j++; // 失配数组值加1
33              if (j == pattern.size()) { // 如果匹配成功
34
35                  res.push_back(i - j + 1); // 存储匹配位置
36                  j = fail[j]; // 回溯到失配位置的失配数组值
37              }
38          }
39          return res; // 返回所有匹配位置
40      }
41  };
42  struct EXKMP {
43      string pattern;
44      vector<int> z;
45      EXKMP() {}
46
47      EXKMP(const std::string& p) {
48          init(p);
49      }
50
51      void init(const std::string& p) {
52          pattern = p;
53          int n = p.size();
54          z.resize(n);
55          z.assign(n, 0);
56          z[0] = p.size();
57          for (int i = 1, l, r = -1; i < n; i++) {
58              if (i <= r) z[i] = min(z[i - l], r - i + 1);
59              while(i + z[i] < n && p[z[i]] == p[i + z[i]]) z[i]++;
60              if (r < i + z[i] - 1) l = i, r = i + z[i] - 1;
61          }
62      }
63      // 匹配函数，返回所有位置的最长前缀
64      std::vector<int> match(const std::string& s) {
65          vector<int> pre(s.size(), 0);
66          int m = s.size(), n = pattern.size();
```

```
        for (int i = 0, l, r = -1; i < m; i++) {
            if (i <= r) pre[i] = min(z[i - l], r - i + 1);
            while(pre[i] < n && i + pre[i] < m && pattern[pre[i]] == s[i + pre[i
                ]]) pre[i]++;
            if (i + pre[i] - 1 > r) l = i, r = i + pre[i] - 1;
        }
        return pre;
    }
};
```

5.4 Manacher

```
struct Manacher{
    string s, t; // s 为原串 t 为补充后的串
    int n;      // t 的长度
    vector<int> d; // 回文半径
    ma(string s) : s(s), n(s.size() * 2 + 3) {
        init(t);
        build(d);
    }
    void init(string &t) {
        t = "$#";
        for (int i = 0; i < s.size(); i++) {
            t += s[i];
            t += "#";
        }
        t += '@';
    }
    void build(vector<int> & d) {
        auto equ = [&](char l, char r) {
            return l == r;
        };
        d = vector<int>(n, 0);
        d[1] = 1;
        for (int i = 2, l = 1, r = 1; i < n; i++) {
            if (i <= r) d[i] = min(d[r + l - i], r - i + 1);
            while (equ(t[i - d[i]], t[i + d[i]])) {
                d[i]++;
            }
            if (i + d[i] - 1 > r) r = i + d[i] - 1, l = i - d[i] + 1;
        }
        return;
    }
};
```

5.5 后缀自动机

```
class SAM {
public:
    class state {
    public:
        state() = default;
        state(int len, int link) :len(len), link(link) { };
        state(int len, int link, map<char, int>& next) :len(len), link(link),
            next(next) { };
        int len, link;
        std::map<char, int> next;
    };
    vector<state> st;
    SAM() {
        st.push_back(state(0, -1));
    }
    SAM(const string& s) :SAM() {
        for (auto ch : s) {
            sam_extend(ch);
        }
    }
    void sam_extend(char ch) {
        int p = st.size() - 1, cur = st.size();
        st.push_back(state(st.back().len + 1, -1));
        while (p != -1 && st[p].next.count(ch) == 0) {
            st[p].next[ch] = cur;
            p = st[p].link;
        }
        if (p == -1) {
            st[cur].link = 0;
        } else {
            int q = st[p].next[ch];
            if (st[q].len == st[p].len + 1) {
                st[cur].link = q;
            } else {
                st.push_back(state(st[p].len + 1, st[q].link, st[q].next));
                int clone = st.size() - 1;
                while (p != -1 && st[p].next[ch] == q) {
                    st[p].next[ch] = clone;
                    p = st[p].link;
                }
                st[q].link = st[cur].link = clone;
            }
        }
    }
};
```

```
struct SAM {
    int vcnt, last;
    std::vector<int> len, link;
    std::vector<std::vector<int>> tr;
```

```
5
6       void init(int size) {
7           vcnt = last = 0;
8           len.resize(size);
9           link.resize(size);
10          tr.resize(size, std::vector<int>(26, 0));
11          link[0] = -1;
12      }
13
14      void clear() {
15          vcnt = last = 0;
16          len.clear();
17          link.clear();
18          tr.clear();
19      }
20
21      void add(int c) {
22          int cur = ++vcnt;
23          len[cur] = len[last] + 1;
24          int p = last;
25          while (p != -1 && !tr[p][c])
26              tr[p][c] = cur, p = link[p];
27          if (p == -1)
28              link[cur] = 0;
29          else {
30              int q = tr[p][c];
31              if (len[q] == len[p] + 1)
32                  link[cur] = q;
33              else {
34                  int clone = ++vcnt;
35                  len[clone] = len[p] + 1;
36                  link[clone] = link[q];
37                  tr[clone] = tr[q];
38                  while (p != -1 && tr[p][c] == q)
39                      tr[p][c] = clone, p = link[p];
40                  link[q] = clone;
41                  link[cur] = clone;
42              }
43          }
44          last = cur;
45      }
46  };
47
48  int main() {
49      SAM S;
50      int size = 2 * 100; // 根据需要的大小设置
51      S.init(size);
52
53      // 使用 SAM 对字符串进行处理
54      std::string input = "abcabca";
55      for (char c : input) {
56          int index = c - 'a';
```

```
57          S.add(index);
58      }
59
60      // 输出 SAM 中的一些信息
61      std::cout << "Number of states: " << S.vcnt << std::endl;
62      std::cout << "Last state: " << S.last << std::endl;
63
64      return 0;
65  }
```

5.6 序列自动机

```
1   class SequenceAM :public vector<vector<int>>{
2   public:
3       SequenceAM() = default;
4       SequenceAM(const string& s, int sigma = 26)
5        :vector<vector<int>>(s.size() + 1, vector<int>(sigma, 0)) {
6           auto &nxt=*this;
7           for(int i=s.size();i>=1;i--) {
8               nxt[i][s[i-1]-'a']=i;
9               nxt[i-1]=nxt[i];
10          }
11      }
12  };
```

六、 数学

6.1 欧拉函数

单独求欧拉函数 phi(x)

```
int phi(int n) {
        int m = 1;
        for (int i = 2; n > 1; ++i) {
                if (n % i == 0) {
                        m *= i - 1;
                        n /= i;
                        while (n % i == 0) {
                                m *= i;
                                n /= i;
                        }
                }
        }
        return m;
}
```

预处理 phi(x)

```
for (i = 1; i <= maxn; i++) phi[i] = i;
for (i = 2; i <= maxn; i += 2) phi[i] /= 2;
for (i = 3; i <= maxn; i += 2) if(phi[i] == i) {
    for (j = i; j <= maxn; j += i)
        phi[j] = phi[j] / i * (i - 1);
}
```

6.2 扩展 GCD

求 x, y 满足 gcd(a, b) = a * x + b * y

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

6.3 各种筛

```
线性筛素数
保证每个数只会被它的最小质因子给筛掉（不同于埃氏筛中每个数会被它所有质因子筛一遍从而使复杂度过高）
int pri[N],tot,zhi[N];//zhi[i]为1的表示不是质数
void sieve()
{
    zhi[1]=1;
    for (int i=2;i<=n;i++)
    {
        if (!zhi[i]) pri[++tot]=i;
        for (int j=1;j<=tot&&i*pri[j]<=n;j++)
        {
            zhi[i*pri[j]]=1;
            if (i%pri[j]==0) break;
        }
    }
}
所有线性筛积性函数都必须基于线性筛素数。
线性筛莫比乌斯函数
int mu[N],pri[N],tot,zhi[N];
void sieve()
{
    zhi[1]=mu[1]=1;
    for (int i=2;i<=n;i++)
    {
        if (!zhi[i]) pri[++tot]=i,mu[i]=-1;
        for (int j=1;j<=tot&&i*pri[j]<=n;j++)
        {
            zhi[i*pri[j]]=1;
            if (i%pri[j]) mu[i*pri[j]]=-mu[i];
            else {mu[i*pri[j]]=0;break;}
        }
    }
}
线性筛欧拉函数
int phi[N],pri[N],tot,zhi[N];
void sieve()
{
    zhi[1]=phi[1]=1;
    for (int i=2;i<=n;i++)
    {
        if (!zhi[i]) pri[++tot]=i,phi[i]=i-1;
        for (int j=1;j<=tot&&i*pri[j]<=n;j++)
        {
            zhi[i*pri[j]]=1;
            if (i%pri[j]) phi[i*pri[j]]=phi[i]*phi[pri[j]];
            else {phi[i*pri[j]]=phi[i]*pri[j];break;}
        }
    }
}
线性筛约数个数
```

```
记d(i)
d(i)表示i的约数个数,d(i)=k (i=1)(ai+1) d(i)=i=1k(ai+1)
维护每一个数的最小值因子出现的次数 (即a1) 即可
int d[N],a[N],pri[N],tot,zhi[N];
void sieve()
{
    zhi[1]=d[1]=1;
    for (int i=2;i<=n;i++)
    {
        if (!zhi[i]) pri[++tot]=i,d[i]=2,a[i]=1;
        for (int j=1;j<=tot&&i*pri[j]<=n;j++)
        {
            zhi[i*pri[j]]=1;
            if (i%pri[j]) d[i*pri[j]]=d[i]*d[pri[j]],a[i*pri[j]]=1;
            else {d[i*pri[j]]=d[i]/(a[i]+1)*(a[i]+2);a[i*pri[j]]=a[i]+1;break;}
        }
    }
}
```

6.4 多项式

```
using i64 = long long;
template<class T>
#define constexpr
constexpr T power(T a, i64 b) {
    T res = 1;
    for (; b; b /= 2, a *= a) {
        if (b % 2) {
            res *= a;
        }
    }
    return res;
}

template<int P>
struct MInt {
    int x;
    constexpr MInt() : x{} {}
    constexpr MInt(i64 x) : x{norm(x % getMod())} {}

    static int Mod;
    constexpr static int getMod() {
        if (P > 0) {
            return P;
        } else {
            return Mod;
        }
    }
    constexpr static void setMod(int Mod_) {
```

```
29          Mod = Mod_;
30      }
31      constexpr int norm(int x) const {
32          if (x < 0) {
33              x += getMod();
34          }
35          if (x >= getMod()) {
36              x -= getMod();
37          }
38          return x;
39      }
40      constexpr int val() const {
41          return x;
42      }
43      explicit constexpr operator int() const {
44          return x;
45      }
46      constexpr MInt operator-() const {
47          MInt res;
48          res.x = norm(getMod() - x);
49          return res;
50      }
51      constexpr MInt inv() const {
52          assert(x != 0);
53          return power(*this, getMod() - 2);
54      }
55      constexpr MInt &operator*=(MInt rhs) & {
56          x = 1LL * x * rhs.x % getMod();
57          return *this;
58      }
59      constexpr MInt &operator+=(MInt rhs) & {
60          x = norm(x + rhs.x);
61          return *this;
62      }
63      constexpr MInt &operator-=(MInt rhs) & {
64          x = norm(x - rhs.x);
65          return *this;
66      }
67      constexpr MInt &operator/=(MInt rhs) & {
68          return *this *= rhs.inv();
69      }
70      friend constexpr MInt operator*(MInt lhs, MInt rhs) {
71          MInt res = lhs;
72          res *= rhs;
73          return res;
74      }
75      friend constexpr MInt operator+(MInt lhs, MInt rhs) {
76          MInt res = lhs;
77          res += rhs;
78          return res;
79      }
80      friend constexpr MInt operator-(MInt lhs, MInt rhs) {
```

```
81          MInt res = lhs;
82          res -= rhs;
83          return res;
84      }
85      friend constexpr MInt operator/(MInt lhs, MInt rhs) {
86          MInt res = lhs;
87          res /= rhs;
88          return res;
89      }
90      friend constexpr std::istream &operator>>(std::istream &is, MInt &a) {
91          i64 v;
92          is >> v;
93          a = MInt(v);
94          return is;
95      }
96      friend constexpr std::ostream &operator<<(std::ostream &os, const MInt &a)
            {
97          return os << a.val();
98      }
99      friend constexpr bool operator==(MInt lhs, MInt rhs) {
100         return lhs.val() == rhs.val();
101     }
102     friend constexpr bool operator!=(MInt lhs, MInt rhs) {
103         return lhs.val() != rhs.val();
104     }
105 };
106
107 template<>
108 int MInt<0>::Mod = 1;
109
110 template<int V, int P>
111 constexpr MInt<P> CInv = MInt<P>(V).inv();
112
113 const int P = 998244353;
114 using Z = MInt<P>;
115
116 std::vector<int> rev;
117 template<int P>
118 std::vector<MInt<P>> roots{0, 1};
119
120 template<int P>
121 constexpr MInt<P> findPrimitiveRoot() {
122     MInt<P> i = 2;
123     int k = __builtin_ctz(P - 1);
124     while (true) {
125         if (power(i, (P - 1) / 2) != 1) {
126             break;
127         }
128         i += 1;
129     }
130     return power(i, (P - 1) >> k);
131 }
```

```
132
133 template<int P>
134 constexpr MInt<P> primitiveRoot = findPrimitiveRoot<P>();
135
136 template<>
137 constexpr MInt<998244353> primitiveRoot<998244353> {31};
138
139 template<int P>
140 constexpr void dft(std::vector<MInt<P>> &a) {
141     int n = a.size();
142
143     if (int(rev.size()) != n) {
144         int k = __builtin_ctz(n) - 1;
145         rev.resize(n);
146         for (int i = 0; i < n; i++) {
147             rev[i] = rev[i >> 1] >> 1 | (i & 1) << k;
148         }
149     }
150
151     for (int i = 0; i < n; i++) {
152         if (rev[i] < i) {
153             std::swap(a[i], a[rev[i]]);
154         }
155     }
156     if (roots<P>.size() < n) {
157         int k = __builtin_ctz(roots<P>.size());
158         roots<P>.resize(n);
159         while ((1 << k) < n) {
160             auto e = power(primitiveRoot<P>, 1 << (__builtin_ctz(P - 1) - k - 1)
                );
161             for (int i = 1 << (k - 1); i < (1 << k); i++) {
162                 roots<P>[2 * i] = roots<P>[i];
163                 roots<P>[2 * i + 1] = roots<P>[i] * e;
164             }
165             k++;
166         }
167     }
168     for (int k = 1; k < n; k *= 2) {
169         for (int i = 0; i < n; i += 2 * k) {
170             for (int j = 0; j < k; j++) {
171                 MInt<P> u = a[i + j];
172                 MInt<P> v = a[i + j + k] * roots<P>[k + j];
173                 a[i + j] = u + v;
174                 a[i + j + k] = u - v;
175             }
176         }
177     }
178 }
179
180 template<int P>
181 constexpr void idft(std::vector<MInt<P>> &a) {
182     int n = a.size();
```

```
183     std::reverse(a.begin() + 1, a.end());
184     dft(a);
185     MInt<P> inv = (1 - P) / n;
186     for (int i = 0; i < n; i++) {
187         a[i] *= inv;
188     }
189 }
190 
191 template<int P = 998244353>
192 struct Poly : public std::vector<MInt<P>> {
193     using Value = MInt<P>;
194 
195     Poly() : std::vector<Value>() {}
196     explicit constexpr Poly(int n) : std::vector<Value>(n) {}
197 
198     explicit constexpr Poly(const std::vector<Value> &a) : std::vector<Value>(a
        ) {}
199     constexpr Poly(const std::initializer_list<Value> &a) : std::vector<Value>(
        a) {}
200 
201     template<class InputIt, class = std::_RequireInputIter<InputIt>>
202     explicit constexpr Poly(InputIt first, InputIt last) : std::vector<Value>(
        first, last) {}
203 
204     template<class F>
205     explicit constexpr Poly(int n, F f) : std::vector<Value>(n) {
206         for (int i = 0; i < n; i++) {
207             (*this)[i] = f(i);
208         }
209     }
210 
211     constexpr Poly shift(int k) const {
212         if (k >= 0) {
213             auto b = *this;
214             b.insert(b.begin(), k, 0);
215             return b;
216         } else if (this->size() <= -k) {
217             return Poly();
218         } else {
219             return Poly(this->begin() + (-k), this->end());
220         }
221     }
222     constexpr Poly trunc(int k) const {
223         Poly f = *this;
224         f.resize(k);
225         return f;
226     }
227     constexpr friend Poly operator+(const Poly &a, const Poly &b) {
228         Poly res(std::max(a.size(), b.size()));
229         for (int i = 0; i < a.size(); i++) {
230             res[i] += a[i];
231         }
```

```
232         for (int i = 0; i < b.size(); i++) {
233             res[i] += b[i];
234         }
235         return res;
236     }
237     constexpr friend Poly operator-(const Poly &a, const Poly &b) {
238         Poly res(std::max(a.size(), b.size()));
239         for (int i = 0; i < a.size(); i++) {
240             res[i] += a[i];
241         }
242         for (int i = 0; i < b.size(); i++) {
243             res[i] -= b[i];
244         }
245         return res;
246     }
247     constexpr friend Poly operator-(const Poly &a) {
248         std::vector<Value> res(a.size());
249         for (int i = 0; i < int(res.size()); i++) {
250             res[i] = -a[i];
251         }
252         return Poly(res);
253     }
254     constexpr friend Poly operator*(Poly a, Poly b) {
255         if (a.size() == 0 || b.size() == 0) {
256             return Poly();
257         }
258         if (a.size() < b.size()) {
259             std::swap(a, b);
260         }
261         int n = 1, tot = a.size() + b.size() - 1;
262         while (n < tot) {
263             n *= 2;
264         }
265         if (((P - 1) & (n - 1)) != 0 || b.size() < 128) {
266             Poly c(a.size() + b.size() - 1);
267             for (int i = 0; i < a.size(); i++) {
268                 for (int j = 0; j < b.size(); j++) {
269                     c[i + j] += a[i] * b[j];
270                 }
271             }
272             return c;
273         }
274         a.resize(n);
275         b.resize(n);
276         dft(a);
277         dft(b);
278         for (int i = 0; i < n; ++i) {
279             a[i] *= b[i];
280         }
281         idft(a);
282         a.resize(tot);
283         return a;
```

```
284     }
285     constexpr friend Poly operator*(Value a, Poly b) {
286         for (int i = 0; i < int(b.size()); i++) {
287             b[i] *= a;
288         }
289         return b;
290     }
291     constexpr friend Poly operator*(Poly a, Value b) {
292         for (int i = 0; i < int(a.size()); i++) {
293             a[i] *= b;
294         }
295         return a;
296     }
297     constexpr friend Poly operator/(Poly a, Value b) {
298         for (int i = 0; i < int(a.size()); i++) {
299             a[i] /= b;
300         }
301         return a;
302     }
303     constexpr Poly &operator+=(Poly b) {
304         return (*this) = (*this) + b;
305     }
306     constexpr Poly &operator-=(Poly b) {
307         return (*this) = (*this) - b;
308     }
309     constexpr Poly &operator*=(Poly b) {
310         return (*this) = (*this) * b;
311     }
312     constexpr Poly &operator*=(Value b) {
313         return (*this) = (*this) * b;
314     }
315     constexpr Poly &operator/=(Value b) {
316         return (*this) = (*this) / b;
317     }
318     constexpr Poly deriv() const {
319         if (this->empty()) {
320             return Poly();
321         }
322         Poly res(this->size() - 1);
323         for (int i = 0; i < this->size() - 1; ++i) {
324             res[i] = (i + 1) * (*this)[i + 1];
325         }
326         return res;
327     }
328     constexpr Poly integr() const {
329         Poly res(this->size() + 1);
330         for (int i = 0; i < this->size(); ++i) {
331             res[i + 1] = (*this)[i] / (i + 1);
332         }
333         return res;
334     }
335     constexpr Poly inv(int m) const {
```

```
336         Poly x{(*this)[0].inv()};
337         int k = 1;
338         while (k < m) {
339             k *= 2;
340             x = (x * (Poly{2} - trunc(k) * x)).trunc(k);
341         }
342         return x.trunc(m);
343     }
344     constexpr Poly log(int m) const {
345         return (deriv() * inv(m)).integr().trunc(m);
346     }
347     constexpr Poly exp(int m) const {
348         Poly x{1};
349         int k = 1;
350         while (k < m) {
351             k *= 2;
352             x = (x * (Poly{1} - x.log(k) + trunc(k))).trunc(k);
353         }
354         return x.trunc(m);
355     }
356     constexpr Poly pow(int k, int m) const {
357         int i = 0;
358         while (i < this->size() && (*this)[i] == 0) {
359             i++;
360         }
361         if (i == this->size() || 1LL * i * k >= m) {
362             return Poly(m);
363         }
364         Value v = (*this)[i];
365         auto f = shift(-i) * v.inv();
366         return (f.log(m - i * k) * k).exp(m - i * k).shift(i * k) * power(v, k)
                ;
367     }
368     constexpr Poly sqrt(int m) const {
369         Poly x{1};
370         int k = 1;
371         while (k < m) {
372             k *= 2;
373             x = (x + (trunc(k) * x.inv(k)).trunc(k)) * CInv<2, P>;
374         }
375         return x.trunc(m);
376     }
377     constexpr Poly mulT(Poly b) const {
378         if (b.size() == 0) {
379             return Poly();
380         }
381         int n = b.size();
382         std::reverse(b.begin(), b.end());
383         return ((*this) * b).shift(-(n - 1));
384     }
385     constexpr std::vector<Value> eval(std::vector<Value> x) const {
386         if (this->size() == 0) {
```

```
387             return std::vector<Value>(x.size(), 0);
388         }
389         const int n = std::max(x.size(), this->size());
390         std::vector<Poly> q(4 * n);
391         std::vector<Value> ans(x.size());
392         x.resize(n);
393         std::function<void(int, int, int)> build = [&](int p, int l, int r) {
394             if (r - l == 1) {
395                 q[p] = Poly{1, -x[l]};
396             } else {
397                 int m = (l + r) / 2;
398                 build(2 * p, l, m);
399                 build(2 * p + 1, m, r);
400                 q[p] = q[2 * p] * q[2 * p + 1];
401             }
402         };
403         build(1, 0, n);
404         std::function<void(int, int, int, const Poly &)> work = [&](int p, int
                l, int r, const Poly &num) {
405             if (r - l == 1) {
406                 if (l < int(ans.size())) {
407                     ans[l] = num[0];
408                 }
409             } else {
410                 int m = (l + r) / 2;
411                 work(2 * p, l, m, num.mulT(q[2 * p + 1]).resize(m - l));
412                 work(2 * p + 1, m, r, num.mulT(q[2 * p]).resize(r - m));
413             }
414         };
415         work(1, 0, n, mulT(q[1].inv(n)));
416         return ans;
417     }
418 };
419
420 template<int P = 998244353>
421 Poly<P> berlekampMassey(const Poly<P> &s) {
422     Poly<P> c;
423     Poly<P> oldC;
424     int f = -1;
425     for (int i = 0; i < s.size(); i++) {
426         auto delta = s[i];
427         for (int j = 1; j <= c.size(); j++) {
428             delta -= c[j - 1] * s[i - j];
429         }
430         if (delta == 0) {
431             continue;
432         }
433         if (f == -1) {
434             c.resize(i + 1);
435             f = i;
436         } else {
437             auto d = oldC;
```

```
            d *= -1;
            d.insert(d.begin(), 1);
            MInt<P> df1 = 0;
            for (int j = 1; j <= d.size(); j++) {
                df1 += d[j - 1] * s[f + 1 - j];
            }
            assert(df1 != 0);
            auto coef = delta / df1;
            d *= coef;
            Poly<P> zeros(i - f - 1);
            zeros.insert(zeros.end(), d.begin(), d.end());
            d = zeros;
            auto temp = c;
            c += d;
            if (i - temp.size() > f - oldC.size()) {
                oldC = temp;
                f = i;
            }
        }
    }
    c *= -1;
    c.insert(c.begin(), 1);
    return c;
}


template<int P = 998244353>
MInt<P> linearRecurrence(Poly<P> p, Poly<P> q, i64 n) {
    int m = q.size() - 1;
    while (n > 0) {
        auto newq = q;
        for (int i = 1; i <= m; i += 2) {
            newq[i] *= -1;
        }
        auto newp = p * newq;
        newq = q * newq;
        for (int i = 0; i < m; i++) {
            p[i] = newp[i * 2 + n % 2];
        }
        for (int i = 0; i <= m; i++) {
            q[i] = newq[i * 2];
        }
        n /= 2;
    }
    return p[0] / q[0];
}
```

6.5 斯特林数

```
std::vector S(n + 2, std::vector<Z>(n + 2));
for (int i = 0; i <= n + 1; i++) {
    S[i][0] = !i;
    for (int j = 1; j <= i; j++) {
        S[i][j] = S[i - 1][j - 1] + S[i - 1][j] * j;
    }
}
```

6.6 高斯消元

```
void gauss(int n, double g[maxn][maxn]) { // input: N * (N + 1) Matrix
        for (int i = 1; i <= n; ++i) {
                double temp = 0;
                int pos = -1;
                for (int j = i; j <= n; ++j) {
                        if (fabs(g[j][i]) > temp) temp = fabs(g[j][i]), pos = j;
                }
                if (pos == -1) continue;
                for (int k = 1; k <= n + 1; ++k) swap(g[pos][k], g[i][k]);
                temp = g[i][i];
                for (int k = 1; k <= n + 1; ++k) g[i][k] /= temp;
                for (int j = i + 1; j <= n; ++j) {
                        temp = g[j][i];
                        for (int k = 1; k <= n + 1; ++k) g[j][k] -= temp * g[i][k
                            ];
                }
        }
        for (int i = n; i >= 1; --i) {
                for (int j = 1; j < i; ++j) {
                        g[j][n + 1] -= g[i][n + 1] * g[j][i];
                        g[j][i] = 0;
                }
        }
}
```

七、 其他

7.1 快读快写

```cpp
template <typename T> inline void read(T& t) {
    int f = 0, c = getchar(); t = 0;
    while (!isdigit(c)) f |= c == '-', c = getchar();
    while (isdigit(c)) t = t * 10 + c - 48, c = getchar();
    if (f) t = -t;
}

template <typename T> void print(T x) {
    if (x < 0) x = -x, putchar('-');
    if (x > 9) print(x / 10);
    putchar(x % 10 + 48);
}
```