ICPC-ACM Template

Ruoyisius

2023 年 9 月 22 日

目录

一、 图论

1.1 增广路

```
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

```cpp
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

```cpp
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

```
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;
            if (d[v] == d[u] + 1 && (f = DFS(v, min(a, e.cap - e.flow))) > 0) {
                e.flow += f;
                ee.flow -= f;
                flow += f;
                a -= f;
                if (a == 0) break;
            }
        }
        return flow;
    }

    int Maxflow(int s, int t) {
        this->s = s; this->t = t;
        int flow = 0;
        while (BFS()) {
            memset(cur, 0, sizeof(cur));
            flow += DFS(s, INF);
        }
        return flow;
    }
};
```

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>
```

```cpp
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

```cpp
#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();
        }
```

```cpp
        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 || !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 || !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);
    }
};

constexpr i64 inf = 1E18;

```

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

3.5 倍增 LCA-最近公共祖先

```
void bfs()
{
    dep[1]=1;
    que[tail++]=1;
    while(head<tail)
    {
        int p=que[head++];
        for(int x=last[p];x!=0;x=pre[x])
        {
            if(f[p][0]!=son[x])
            {
                dep[son[x]]=dep[p]+1;
                f[son[x]][0]=p;
                que[tail++]=son[x];
                for(int j=1;j<=20;j++)
                    f[son[x]][j]=f[f[son[x]][j-1]][j-1];
            }
        }
    }
}
int lca(int a,int b)
{
    if(dep[a]>dep[b]) swap(a,b);
    for(int i=20;i>=0;i--)
    {
        if(dep[f[b][i]]>=dep[a]) b=f[b][i];
        if(a==b) return a;
    }
    for(int i=20;i>=0;i--)
        if(f[a][i]!=f[b][i]) a=f[a][i],b=f[b][i];
    return f[a][0];
}
```

3.6 珂朵莉树

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

```cpp
    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

```cpp
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);
    }
```

```cpp

    void Union(int x, int y) {
        par[x] = y;
    }

    int Find(int x, int c = 0) {
        if (par[x] == x)
            return c ? -1 : x;
        int p = Find(par[x], 1);
        if (p == -1)
            return c ? par[x] : val[x];
        if (sdom[val[x]] > sdom[val[par[x]]])
            val[x] = val[par[x]];
        par[x] = p;
        return c ? p : val[x];
    }

    void dfs(int x) {
        RS[S[x] = ++cs] = x;
        par[cs] = sdom[cs] = val[cs] = cs;
        for (int e : E[x]) {
            if (S[e] == 0)
                dfs(e), rp[S[e]] = S[x];
            RE[S[e]].push_back(S[x]);
        }
    }

    int solve(int s, std::vector<int>& up) {
        dfs(s);
        for (int i = cs; i; i--) {
            for (int e : RE[i])
                sdom[i] = std::min(sdom[i], sdom[Find(e)]);
            if (i > 1)
                rdom[sdom[i]].push_back(i);
            for (int e : rdom[i]) {
                int p = Find(e);
                if (sdom[p] == i)
                    dom[e] = i;
                else
                    dom[e] = p;
            }
            if (i > 1)
                Union(i, rp[i]);
        }
        for (int i = 2; i <= cs; i++)
            if (sdom[i] != dom[i])
                dom[i] = dom[dom[i]];
        for (int i = 2; i <= cs; i++)
            up[RS[i]] = RS[dom[i]];
        return cs;
    }
};
```

四、 计算几何

五、 字符串

5.1 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];
      }
    }
  }
};
```

5.2 KMP& EXKMP

```cpp
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
        for (int i = 0; i < pattern.size(); i++) { // 遍历模式串
            while (j >= 0 && pattern[i] != pattern[j]) { // 如果失配，则回溯
                j = fail[j]; // 回溯到失配位置的失配数组值
            }
            j++; // 失配数组值加1
            fail[i + 1] = j; // 更新失配数组
        }
    }
    // 匹配函数，返回所有匹配位置
    std::vector<int> match(const std::string& s) {
        std::vector<int> res;
        int j = 0;
        for (int i = 0; i < s.size(); i++) { // 遍历文本串
            while (j >= 0 && s[i] != pattern[j]) { // 如果失配，则回溯
                j = fail[j]; // 回溯到失配位置的失配数组值
            }
            // dbg(i, j, s[i], pattern[j], pattern.size());
            j++; // 失配数组值加1
            if (j == pattern.size()) { // 如果匹配成功

                res.push_back(i - j + 1); // 存储匹配位置
                j = fail[j]; // 回溯到失配位置的失配数组值
            }
        }
        return res; // 返回所有匹配位置
    }
};
struct EXKMP {
    string pattern;
    vector<int> z;
    EXKMP() {}

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

    void init(const std::string& p) {
        pattern = p;
        int n = p.size();
        z.resize(n);
        z.assign(n, 0);
        z[0] = p.size();
        for (int i = 1, l, r = -1; i < n; i++) {
            if (i <= r) z[i] = min(z[i - l], r - i + 1);
            while(i + z[i] < n && p[z[i]] == p[i + z[i]]) z[i]++;
            if (r < i + z[i] - 1) l = i, r = i + z[i] - 1;
        }
    }
    // 匹配函数，返回所有位置的最长前缀
    std::vector<int> match(const std::string& s) {
        vector<int> pre(s.size(), 0);
        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;
    }
};
```

5.3 Manacher

```cpp
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;
    }
};
```

5.4 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));
```

```cpp
                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;
            }
        }
    }
};
```

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

    void init(int size) {
        vcnt = last = 0;
        len.resize(size);
        link.resize(size);
        tr.resize(size, std::vector<int>(26, 0));
        link[0] = -1;
    }

    void clear() {
        vcnt = last = 0;
        len.clear();
        link.clear();
        tr.clear();
    }

    void add(int c) {
        int cur = ++vcnt;
        len[cur] = len[last] + 1;
        int p = last;
        while (p != -1 && !tr[p][c])
            tr[p][c] = cur, p = link[p];
        if (p == -1)
            link[cur] = 0;
        else {
            int q = tr[p][c];
            if (len[q] == len[p] + 1)
                link[cur] = q;
            else {
                int clone = ++vcnt;
                len[clone] = len[p] + 1;
                link[clone] = link[q];
                tr[clone] = tr[q];
                while (p != -1 && tr[p][c] == q)
                    tr[p][c] = clone, p = link[p];
                link[q] = clone;
                link[cur] = clone;
            }
        }
        last = cur;
    }
};

int main() {
    SAM S;
    int size = 2 * 100; // 根据需要的大小设置
    S.init(size);

    // 使用 SAM 对字符串进行处理
    std::string input = "abcabca";
    for (char c : input) {
        int index = c - 'a';
        S.add(index);
    }

    // 输出 SAM 中的一些信息
    std::cout << "Number␣of␣states:␣" << S.vcnt << std::endl;
    std::cout << "Last␣state:␣" << S.last << std::endl;

    return 0;
}
```

5.5 序列自动机

```
class SequenceAM :public vector<vector<int>>{
public:
    SequenceAM() = default;
    SequenceAM(const string& s, int sigma = 26)
     :vector<vector<int>>(s.size() + 1, vector<int>(sigma, 0)) {
        auto &nxt=*this;
        for(int i=s.size();i>=1;i--) {
            nxt[i][s[i-1]-'a']=i;
            nxt[i-1]=nxt[i];
        }
    }
};
```

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_) {
        Mod = Mod_;
    }
    constexpr int norm(int x) const {
        if (x < 0) {
            x += getMod();
        }
        if (x >= getMod()) {
            x -= getMod();
        }
        return x;
    }
    constexpr int val() const {
        return x;
    }
    explicit constexpr operator int() const {
        return x;
    }
    constexpr MInt operator-() const {
        MInt res;
        res.x = norm(getMod() - x);
        return res;
    }
    constexpr MInt inv() const {
        assert(x != 0);
        return power(*this, getMod() - 2);
    }
    constexpr MInt &operator*=(MInt rhs) & {
        x = 1LL * x * rhs.x % getMod();
        return *this;
    }
    constexpr MInt &operator+=(MInt rhs) & {
        x = norm(x + rhs.x);
        return *this;
    }
    constexpr MInt &operator-=(MInt rhs) & {
        x = norm(x - rhs.x);
        return *this;
    }
    constexpr MInt &operator/=(MInt rhs) & {
        return *this *= rhs.inv();
    }
    friend constexpr MInt operator*(MInt lhs, MInt rhs) {
        MInt res = lhs;
        res *= rhs;
        return res;
    }
    friend constexpr MInt operator+(MInt lhs, MInt rhs) {
        MInt res = lhs;
        res += rhs;
        return res;
    }
    friend constexpr MInt operator-(MInt lhs, MInt rhs) {
        MInt res = lhs;
        res -= rhs;
        return res;
    }
    friend constexpr MInt operator/(MInt lhs, MInt rhs) {
        MInt res = lhs;
        res /= rhs;
        return res;
    }
    friend constexpr std::istream &operator>>(std::istream &is, MInt &a) {
        i64 v;
        is >> v;
        a = MInt(v);
        return is;
    }
    friend constexpr std::ostream &operator<<(std::ostream &os, const MInt &a)
        {
        return os << a.val();
    }
    friend constexpr bool operator==(MInt lhs, MInt rhs) {
        return lhs.val() == rhs.val();
    }
    friend constexpr bool operator!=(MInt lhs, MInt rhs) {
        return lhs.val() != rhs.val();
    }
};

template<>
int MInt<0>::Mod = 1;

template<int V, int P>
constexpr MInt<P> CInv = MInt<P>(V).inv();

const int P = 998244353;
using Z = MInt<P>;

std::vector<int> rev;
template<int P>
std::vector<MInt<P>> roots{0, 1};

template<int P>
constexpr MInt<P> findPrimitiveRoot() {
    MInt<P> i = 2;
    int k = __builtin_ctz(P - 1);
    while (true) {
        if (power(i, (P - 1) / 2) != 1) {
            break;
        }
        i += 1;
    }
    return power(i, (P - 1) >> k);
}
```

```cpp

template<int P>
constexpr MInt<P> primitiveRoot = findPrimitiveRoot<P>();

template<>
constexpr MInt<998244353> primitiveRoot<998244353> {31};

template<int P>
constexpr void dft(std::vector<MInt<P>> &a) {
    int n = a.size();

    if (int(rev.size()) != n) {
        int k = __builtin_ctz(n) - 1;
        rev.resize(n);
        for (int i = 0; i < n; i++) {
            rev[i] = rev[i >> 1] >> 1 | (i & 1) << k;
        }
    }

    for (int i = 0; i < n; i++) {
        if (rev[i] < i) {
            std::swap(a[i], a[rev[i]]);
        }
    }
    if (roots<P>.size() < n) {
        int k = __builtin_ctz(roots<P>.size());
        roots<P>.resize(n);
        while ((1 << k) < n) {
            auto e = power(primitiveRoot<P>, 1 << (__builtin_ctz(P - 1) - k - 1)
                );
            for (int i = 1 << (k - 1); i < (1 << k); i++) {
                roots<P>[2 * i] = roots<P>[i];
                roots<P>[2 * i + 1] = roots<P>[i] * e;
            }
            k++;
        }
    }
    for (int k = 1; k < n; k *= 2) {
        for (int i = 0; i < n; i += 2 * k) {
            for (int j = 0; j < k; j++) {
                MInt<P> u = a[i + j];
                MInt<P> v = a[i + j + k] * roots<P>[k + j];
                a[i + j] = u + v;
                a[i + j + k] = u - v;
            }
        }
    }
}

template<int P>
constexpr void idft(std::vector<MInt<P>> &a) {
    int n = a.size();
    std::reverse(a.begin() + 1, a.end());
    dft(a);
    MInt<P> inv = (1 - P) / n;
    for (int i = 0; i < n; i++) {
        a[i] *= inv;
    }
}

template<int P = 998244353>
struct Poly : public std::vector<MInt<P>> {
    using Value = MInt<P>;

    Poly() : std::vector<Value>() {}
    explicit constexpr Poly(int n) : std::vector<Value>(n) {}

    explicit constexpr Poly(const std::vector<Value> &a) : std::vector<Value>(a
        ) {}
    constexpr Poly(const std::initializer_list<Value> &a) : std::vector<Value>(
        a) {}

    template<class InputIt, class = std::_RequireInputIter<InputIt>>
    explicit constexpr Poly(InputIt first, InputIt last) : std::vector<Value>(
        first, last) {}

    template<class F>
    explicit constexpr Poly(int n, F f) : std::vector<Value>(n) {
        for (int i = 0; i < n; i++) {
            (*this)[i] = f(i);
        }
    }

    constexpr Poly shift(int k) const {
        if (k >= 0) {
            auto b = *this;
            b.insert(b.begin(), k, 0);
            return b;
        } else if (this->size() <= -k) {
            return Poly();
        } else {
            return Poly(this->begin() + (-k), this->end());
        }
    }
    constexpr Poly trunc(int k) const {
        Poly f = *this;
        f.resize(k);
        return f;
    }
    constexpr friend Poly operator+(const Poly &a, const Poly &b) {
        Poly res(std::max(a.size(), b.size()));
        for (int i = 0; i < a.size(); i++) {
            res[i] += a[i];
        }
        for (int i = 0; i < b.size(); i++) {
            res[i] += b[i];
        }
        return res;
    }
    constexpr friend Poly operator-(const Poly &a, const Poly &b) {
        Poly res(std::max(a.size(), b.size()));
        for (int i = 0; i < a.size(); i++) {
            res[i] += a[i];
        }
        for (int i = 0; i < b.size(); i++) {
            res[i] -= b[i];
        }
        return res;
    }
    constexpr friend Poly operator-(const Poly &a) {
        std::vector<Value> res(a.size());
        for (int i = 0; i < int(res.size()); i++) {
            res[i] = -a[i];
        }
        return Poly(res);
    }
    constexpr friend Poly operator*(Poly a, Poly b) {
        if (a.size() == 0 || b.size() == 0) {
            return Poly();
        }
        if (a.size() < b.size()) {
            std::swap(a, b);
        }
        int n = 1, tot = a.size() + b.size() - 1;
        while (n < tot) {
            n *= 2;
        }
        if (((P - 1) & (n - 1)) != 0 || b.size() < 128) {
            Poly c(a.size() + b.size() - 1);
            for (int i = 0; i < a.size(); i++) {
                for (int j = 0; j < b.size(); j++) {
                    c[i + j] += a[i] * b[j];
                }
            }
            return c;
        }
        a.resize(n);
        b.resize(n);
        dft(a);
        dft(b);
        for (int i = 0; i < n; ++i) {
            a[i] *= b[i];
        }
        idft(a);
        a.resize(tot);
        return a;
    }
    constexpr friend Poly operator*(Value a, Poly b) {
        for (int i = 0; i < int(b.size()); i++) {
            b[i] *= a;
        }
        return b;
    }
    constexpr friend Poly operator*(Poly a, Value b) {
        for (int i = 0; i < int(a.size()); i++) {
            a[i] *= b;
        }
        return a;
    }
    constexpr friend Poly operator/(Poly a, Value b) {
        for (int i = 0; i < int(a.size()); i++) {
            a[i] /= b;
        }
        return a;
    }
    constexpr Poly &operator+=(Poly b) {
        return (*this) = (*this) + b;
    }
    constexpr Poly &operator-=(Poly b) {
        return (*this) = (*this) - b;
    }
    constexpr Poly &operator*=(Poly b) {
        return (*this) = (*this) * b;
    }
    constexpr Poly &operator*=(Value b) {
        return (*this) = (*this) * b;
    }
    constexpr Poly &operator/=(Value b) {
        return (*this) = (*this) / b;
    }
    constexpr Poly deriv() const {
        if (this->empty()) {
            return Poly();
        }
        Poly res(this->size() - 1);
        for (int i = 0; i < this->size() - 1; ++i) {
            res[i] = (i + 1) * (*this)[i + 1];
        }
        return res;
    }
    constexpr Poly integr() const {
        Poly res(this->size() + 1);
        for (int i = 0; i < this->size(); ++i) {
            res[i + 1] = (*this)[i] / (i + 1);
        }
        return res;
    }
    constexpr Poly inv(int m) const {
        Poly x{(*this)[0].inv()};
        int k = 1;
        while (k < m) {
            k *= 2;
            x = (x * (Poly{2} - trunc(k) * x)).trunc(k);
        }
        return x.trunc(m);
    }
    constexpr Poly log(int m) const {
        return (deriv() * inv(m)).integr().trunc(m);
    }
    constexpr Poly exp(int m) const {
        Poly x{1};
        int k = 1;
        while (k < m) {
            k *= 2;
            x = (x * (Poly{1} - x.log(k) + trunc(k))).trunc(k);
        }
        return x.trunc(m);
    }
    constexpr Poly pow(int k, int m) const {
        int i = 0;
        while (i < this->size() && (*this)[i] == 0) {
            i++;
        }
        if (i == this->size() || 1LL * i * k >= m) {
            return Poly(m);
        }
        Value v = (*this)[i];
        auto f = shift(-i) * v.inv();
        return (f.log(m - i * k) * k).exp(m - i * k).shift(i * k) * power(v, k)
            ;
    }
    constexpr Poly sqrt(int m) const {
        Poly x{1};
        int k = 1;
        while (k < m) {
            k *= 2;
            x = (x + (trunc(k) * x.inv(k)).trunc(k)) * CInv<2, P>;
        }
        return x.trunc(m);
    }
    constexpr Poly mulT(Poly b) const {
        if (b.size() == 0) {
            return Poly();
        }
        int n = b.size();
        std::reverse(b.begin(), b.end());
        return ((*this) * b).shift(-(n - 1));
    }
    constexpr std::vector<Value> eval(std::vector<Value> x) const {
        if (this->size() == 0) {
            return std::vector<Value>(x.size(), 0);
        }
        const int n = std::max(x.size(), this->size());
        std::vector<Poly> q(4 * n);
        std::vector<Value> ans(x.size());
        x.resize(n);
        std::function<void(int, int, int)> build = [&](int p, int l, int r) {
            if (r - l == 1) {
                q[p] = Poly{1, -x[l]};
            } else {
                int m = (l + r) / 2;
                build(2 * p, l, m);
                build(2 * p + 1, m, r);
                q[p] = q[2 * p] * q[2 * p + 1];
            }
        };
        build(1, 0, n);
        std::function<void(int, int, int, const Poly &)> work = [&](int p, int l, int r, const Poly &num) {
            if (r - l == 1) {
                if (l < int(ans.size())) {
                    ans[l] = num[0];
                }
            } else {
                int m = (l + r) / 2;
                work(2 * p, l, m, num.mulT(q[2 * p + 1]).resize(m - l));
                work(2 * p + 1, m, r, num.mulT(q[2 * p]).resize(r - m));
            }
        };
        work(1, 0, n, mulT(q[1].inv(n)));
        return ans;
    }
};

template<int P = 998244353>
Poly<P> berlekampMassey(const Poly<P> &s) {
    Poly<P> c;
    Poly<P> oldC;
    int f = -1;
    for (int i = 0; i < s.size(); i++) {
        auto delta = s[i];
        for (int j = 1; j <= c.size(); j++) {
            delta -= c[j - 1] * s[i - j];
        }
        if (delta == 0) {
            continue;
        }
        if (f == -1) {
            c.resize(i + 1);
            f = i;
        } else {
            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

```cpp
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

```cpp
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);
}
```