c++

```cpp
#include <bits/stdc++.h>
#define LL long long
#define P pair<int, int>
#define lowbit(x) (x & -x)
#define mem(a, b) memset(a, b, sizeof(a))
#define mid ((l + r) >> 1)
#define lc rt<<1
#define rc rt<<1|1
#define endl '\n'
const int maxn = 1e5 + 5;
const int inf = 0x3f3f3f3f;
const int mod = 1e9 + 7;
using namespace std;

int main () {
    ios::sync_with_stdio(0);
    cin.tie(0), cout.tie(0);

    return 0;
}
```

矩阵快速幂

```cpp
struct Matrix{
    int n;
    LL mat[151][151];
    Matrix(){}
    Matrix(int _n) {
        n = _n;
        for (int i = 0; i < n; ++i) {
            for (int j = 0; j < n; ++j) {
                mat[i][j] = 0;
            }
        }
    }
    Matrix operator *(const Matrix &b) const{
        Matrix ans = Matrix(n);
        for (int i = 0; i < n; ++i) {
            for (int j = 0; j < n; ++j) {
                for (int k = 0; k < n; ++k) {
                    ans.mat[i][j] += mat[i][k] * b.mat[k][j];
                    ans.mat[i][j] %= mod;
                    // ans.mat[i][j] = (ans.mat[i][j] + mat[i][k] * b.mat[k][j] % m
                }
            }
        }
        return ans;
    }
};
Matrix Pow(Matrix a, int n) {
    Matrix ans = Matrix(a.n);
    Matrix tmp = a;
    for (int i = 0; i < a.n; ++i) {
        ans.mat[i][i] = 1;
    }
    while (n) {
        if (n & 1) ans = ans * tmp;
        tmp = tmp * tmp;
        n >>= 1;
    }
    return ans;
}
/*
0 1
1 1
*/
```

Tarjan

```
int Stack[maxn], low[maxn], dfn[maxn], inStack[maxn], belong[maxn];
int now = 0, len = 0, cnt= 0;
// now:时间戳，len：栈的大小，cnt强连通的个数
void tarjan(int x) {
    // 打上标记，入栈
    low[x] = dfn[x] = ++now;
    Stack[++len] = x;
    inStack[x] = 1;
    for (int i = 0; i < (int)g[x].size(); ++i) {
        int y = g[x][i];
                // 没有访问过，继续递归
            // 在栈中表示可以形成一个强连通分量，更新根节点的low，继续找
        if (!dfn[y]) tarjan(y), low[x] = min(low[x], low[y]);
        else if (inStack[y]) low[x] = min(low[x], low[y]);
    }
        // 回溯，如果当前节点的dfn = low 表示栈中形成一个强连通分量
    if (dfn[x] == low[x]) {
        ++cnt; // 统计个数
        int top;
        while (Stack[len] != x) {
            top = Stack[len--];
            belong[top] = cnt;
            inStack[top] = 0;
        }
        top = Stack[len--];
        belong[top] = cnt; // 记录每个点的隶属关系
        inStack[top] = 0;
    }
}
for (int i = 1; i <= n; ++i) {
    if (!dfn[i])  tarjan(i);
}
```

KMP

```
int a[N], b[N], Next[N];        //从a数组里匹配b数组
void get_next(int m) {  //求Next数组
        Next[0] = -1;
        int i = 0, j = -1;
        while (i < m) {
                if(j == -1 || b[i] == b[j]) {
                        Next[++i] = ++j;        //赋值
                }else {
                        j = Next[j];    //回溯
                }
        }
}
int kmp(int n, int m) {
        get_next(m);    //求Next数组
        int i = 0, j = 0;
        int ans = 0;
        while (i < n) {
                if (j == -1 || a[i] == b[j]) {  //当前匹配成功进行下一个匹配
                        i++;
                        j++;
                }else {
                        j = Next[j];
                }
                if (j == m) {   //匹配成功
                        ans++;
                        j = Next[j];    //进行下一次匹配
                }
        }
        return ans;
}
```

最小表示法

```
int minRepresent(char *s, int len) {
    int i = 0, j = 1, k = 0;
    while (i < len && j < len && k < len) {
        int t = s[(i+k) % len] - s[(j+k) % len];
        if (t == 0) k++;
        else {
            if (t < 0)  j = max(j+k+1, i+1);
            else i = max(i+k+1, j+1);
            k = 0;
        }
    }
    return min(i, j);
}
```

Manacher

```
int RL[maxn << 1];
int Manacher(string s) {
        string t;
        for (int i = 0; i < (int)s.size(); ++i) {
                t += s[i];
                t += '#';
        }
        s = "#" + t;
        int MaxRight = 0, pos = 0, MaxLen = 0;
        for (int i = 0; i < (int)s.size(); ++i) {
                if (i < MaxRight)       RL[i] = min(RL[2 * pos - i], MaxRight - i +
                else    RL[i] = 1;
                int l = i - RL[i];
                int r = i + RL[i];
                while (l >= 0 && r < (int)s.size() && s[l] == s[r])     {
                        RL[i] += 1;
                        l = i - RL[i];
                        r = i + RL[i];
                }
                if (RL[i] + i - 1 > MaxRight) {
                        MaxRight = RL[i] + i - 1;
                        pos = i;
                }
                MaxLen = max(MaxLen, RL[i]);
        }
        return MaxLen - 1;
}

int RL[maxn << 1];
char s[maxn], t[maxn << 1];
int Manacher(char *s) {
        if (s[strlen(s) - 1] == '\n')   s[strlen(s) - 1] = '\0';
        int lens = strlen(s), len = 0;
        t[len++] = '#';
        for (int i = 0; i < lens; ++i) {
                t[len++] = s[i];
                t[len++] = '#';
        }
        int MaxRight = 0, pos = 0, MaxLen = 0;
        for (int i = 0; i < len; ++i) {
                if (i < MaxRight)       RL[i] = min(RL[2 * pos - i], MaxRight - i +
                else    RL[i] = 1;
                int l = i - RL[i];
                int r = i + RL[i];
                while (l >= 0 && r < len && t[l] == t[r])       {
                        RL[i] += 1;
                        l = i - RL[i];
                        r = i + RL[i];
                }
                if (RL[i] + i - 1 > MaxRight) {
                        MaxRight = RL[i] + i - 1;
                        pos = i;
                }
                MaxLen = max(MaxLen, RL[i]);
        }
        return MaxLen - 1;
}
```

后缀自动机

```
#include <bits/stdc++.h>
#define LL long long
#define P pair<int, int>
#define lowbit(x) (x & -x)
#define mem(a, b) memset(a, b, sizeof(a))
#define mid ((l + r) >> 1)
#define lc rt<<1
#define rc rt<<1|1
using namespace std;
const int maxn = 1e5 + 5;
char s[maxn];
int n, m;
struct Sam{
    int trans[maxn<<1][26], slink[maxn<<1], maxlen[maxn<<1];
    int last, now, root, len;
    inline void newnode (int v) {
        maxlen[++now] = v;
    }
    inline void extend(int c) {
        newnode(maxlen[last] + 1);
        int p = last, np = now;
        while (p && !trans[p][c]) {
            trans[p][c] = np;
            p = slink[p];
        }
        if (!p) slink[np] = root;
        else {
            int q = trans[p][c];
            if (maxlen[p] + 1 != maxlen[q]) {
                newnode(maxlen[p] + 1);
                int nq = now;
                memcpy(trans[nq], trans[q], sizeof(trans[q]));
                slink[nq] = slink[q];
                slink[q] = slink[np] = nq;
                while (p && trans[p][c] == q) {
                    trans[p][c] = nq;
                    p = slink[p];
                }
            }else slink[np] = q;
        }
        last = np;
    }
    inline void build() {
        scanf("%s", s);
        len = strlen(s);
        root = last = now = 1;
        for (int i = 0; i < len; ++i) extend(s[i] - 'a');
    }
    inline LL num() {
        LL ans = 0;
        for (int i = root; i <= now; ++i) {
            ans += max(maxlen[i]- max(maxlen[slink[i]], m-1), 0);
        }
        return ans;
    }
}sam;
```

行列式

```
LL exgcd(LL a, LL b, LL &x, LL &y) {
    LL d = a;
    if (b == 0) x = 1,y = 0;
    else {
        d = exgcd(b, a%b, y, x);
        y -= a / b * x;
    }
    return d;
}
LL inv(LL a, LL mod) {
    LL x, y;
    LL d = exgcd(a, mod, x, y);
    return (d == 1) ? ((x + mod) % mod) : -1;
}

LL det(int n) {
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) {
            A[i][j] = (A[i][j] % mod + mod) % mod;
        }
    }
    LL tmp = 1;
    for (int i = 0; i < n; ++i) {
        for (int j = i + 1; j < n; ++j) {
            while (A[j][i]) {
                // LL t = A[i][i] * inv(A[j][i], mod);
                LL t = A[i][i] / A[j][i];
                for (int k = i; k < n; ++k) {
                    A[i][k] = (A[i][k] - t * A[j][k]) % mod;
                    // swap(A[i][k], A[j][k]);
                }
                swap(A[i], A[j]);
                tmp *= -1;
            }
        }
        if (!A[i][i]) return 0;
        tmp = A[i][i] * tmp % mod;
    }
    return (tmp + mod) % mod;
}
```

最小树形图

```
struct ac{
    int u, v, w;
};
vector<ac> g(maxn);
int pre[maxn], vis[maxn], id[maxn], in[maxn];
int zhuliu(int rt, int n, int m) {
    int ans = 0, u, v, w;
    while (1) {
        for (int i = 0; i < n; ++i) in[i] = inf;
        for (int i = 0; i < m; ++i) {
            u = g[i].u; v = g[i].v; w = g[i].w;
            if (u != v && w < in[v]) {
                pre[v] = u;
                in[v] = w;
                // if (u == rt) pos = i; // 记录前驱，输出序号最小的根
            }
        }
        for (int i = 0; i < n; ++i) {
            if (i != rt && in[i] == inf) return -1;
        }
        int cnt = 0;
        mem(id, -1);
        mem(vis, -1);
        in[rt] = 0;
        for (int i = 0; i < n; ++i) {
            ans += in[i];
            u = i;
            while (vis[u] != i && id[u] == -1 && u != rt) {
                vis[u] = i;
                u = pre[u];
            }
            if (u != rt &&id[u] == -1) {
                v = pre[u];
                while (v != u) {
                    id[v] = cnt;
                    v = pre[v];
                }
                id[u] = cnt++;
            }
        }
        if (cnt == 0) break;
        for (int i = 0; i < n; ++i) {
            if (id[i] == -1) id[i] = cnt++;
        }
        for (int i = 0; i < m; ++i) {
            v = g[i].v;
            g[i].u = id[g[i].u];
            g[i].v = id[g[i].v];
            if (g[i].u != g[i].v) g[i].w -= in[v];
        }
        n = cnt;
        rt = id[rt];
    }
    return ans;
}
```

次小生成树

```
// Kruskal
int n, m;
struct ac{
        int u, v, w, flag;
        bool operator <(ac t) {
                return w < t.w;
        }
}g[maxn*maxn];
vector<int> son[maxn];
int pre[maxn], dis[maxn][maxn];
int find (int x) {
        return (pre[x] == x) ? x : pre[x] = find(pre[x]);
}
void Kruskal() {
        for (int i = 0; i <= n; ++i) {
                son[i].clear();
                son[i].push_back(i);
                pre[i] = i;
        }
        sort(g, g+m);
        int sum = 0;
        int cnt = 0;
        for (int i = 0; i < m; ++i) {
                if (cnt == n+1) break;
                int fx = find(g[i].u);
                int fy = find(g[i].v);
                if (fx == fy) continue;
                g[i].flag = 1;
                sum += g[i].w;
                cnt++;
                int lenx = son[fx].size();
                int leny = son[fy].size();
                if (lenx < leny) {
                        swap(lenx, leny);
                        swap(fx, fy);
                }
                // 更新两点的距离最大值
                for (int j = 0; j < lenx; ++j) {
                        for (int k = 0; k < leny; ++k) {
                                dis[son[fx][j]][son[fy][k]] = dis[son[fy][k]][son[f
                        }
                }
                pre[fy] = fx;
                //合并子树
                for (int j = 0; j < leny; ++j) {
                        son[fx].push_back(son[fy][j]);
                }
                son[fy].clear();
        }
        int ans = inf;
        for (int i = 0; i < m; ++i) {
                if (g[i].flag) continue;
                ans = min(ans, sum + g[i].w - dis[g[i].u][g[i].v]);
        }
        printf("%d %d\n", sum, ans);
}


// Prim
int n, m;
int g[maxn][maxn], val[maxn], vis[maxn], dis[maxn];
int pre[maxn], maxd[maxn][maxn];
bool used[maxn][maxn];

void prim(int s) {
    mem(maxd, 0);
    mem(vis, 0);
    mem(used, 0);
    for (int i = 1; i <= n; ++i) {
        dis[i] = g[s][i];
        pre[i] = s;
    }

    vis[s] = 1;
    int sum = 0, cnt = 0;
    for (int i = 1; i < n; ++i) {
        int u = -1, MIN = inf;
        for (int j = 1; j <= n; ++j) {
            if (vis[j]) continue;
            if (MIN > dis[j]) {
                MIN = dis[j];
                u = j;
            }
        }
        if (u == -1) break;
        vis[u] = 1;
        sum += MIN;
        cnt++;
        used[pre[u]][u] = used[u][pre[u]] = 1;
        maxd[u][pre[u]] = maxd[pre[u]][u] = MIN;
        for (int j = 1; j <= n; ++j) {
            if (j == u) continue;
            if (vis[j]) {
                maxd[u][j] = maxd[j][u] = max(maxd[pre[u]][j], MIN);
            }
            if (vis[j] == 0 && dis[j] > g[u][j]) {
                dis[j] = g[u][j];
                pre[j] = u;
            }
        }
    }
    if (cnt != n-1) {
        puts("No way");
    }
    int ans = inf;
    for (int i = 1; i <= n; ++i) {
        for (int j = i+1; j <= n; ++j) {
            if (used[i][j]) continue;
            ans = min(ans, sum + g[i][j] - maxd[i][j]);
        }
    }
    printf("%d %d\n", sum, ans);
}
```

费用流

```
int path[maxn], dis[maxn], head[maxn];
bool vis[maxn];
int n, m, k, cnt;
struct ac{
        int v, c, cost, pre;
}edge[maxn<<7];
void addedge(int u, int v, int c, int cost) {
        edge[cnt].v = v;
        edge[cnt].c = c;
        edge[cnt].cost = cost;
        edge[cnt].pre = head[u];
        head[u] = cnt++;
}
bool spfa (int s, int e) {
        mem(vis, false);
        mem(dis, inf);
        mem(path, -1);
        queue<int> que;
        que.push(s);
        dis[s] = 0;
        vis[s] = true;
        while (!que.empty()) {
                int u = que.front();
                que.pop();
                vis[u] = false;
                for (int i = head[u]; i != -1; i = edge[i].pre) {
                        int v = edge[i].v;
                        int c = edge[i].c;
                        int cost = edge[i].cost;
                        if (dis[v] > dis[u] + cost && c > 0) {
                                dis[v] = dis[u] + cost;
                                path[v] = i;
                                if (!vis[v]) {
                                        vis[v] = true;
                                        que.push(v);
                                }
                        }
                }
        }
        if (dis[e] == inf) return false;
        else return true;
}

int MincostMaxflow(int s, int e, int &cost) {
        int maxflow = 0;
        int flow = inf;
        while (spfa(s, e)) {
                for (int i = path[e]; i != -1; i = path[edge[i^1].v])
                        flow = min(flow, edge[i].c);
                for (int i = path[e]; i != -1; i = path[edge[i^1].v]) {
                        edge[i].c -= flow;
                        edge[i^1].c += flow;
                        cost += flow * edge[i].cost;
                }
                maxflow += flow;
        }
        return maxflow;
}
```

Dinic

```cpp
const int maxn = 411;
const int inf = 0x3f3f3f3f;
struct ac{
    int v, c, pre;
}edge[maxn<<6];
int s, e;
int head[maxn], dis[maxn], curedge[maxn], cnt;
void add(int u, int v, int c) {
    edge[cnt] = {v, c, head[u]};
    head[u] = cnt++;
}
bool bfs() {
    queue<int> que;
    que.push(s);
    mem(dis, 0);
    dis[s] = 1;
    while (!que.empty()) {
        int f = que.front();
        que.pop();
        for (int i = head[f]; i != -1; i = edge[i].pre) {
            if (dis[edge[i].v] || edge[i].c == 0) continue;
            dis[edge[i].v] = dis[f] + 1;
            que.push(edge[i].v);
        }
    }
    return dis[e] > 0;
}

int dfs(int now, int flow) {
    if (now == e || flow == 0) return flow;
    for (int &i = curedge[now]; i != -1; i = edge[i].pre) { // 当前弧优化
        if (dis[edge[i].v] != dis[now] + 1 || edge[i].c == 0) continue;
        int d = dfs(edge[i].v, min(flow, edge[i].c));
        if (d > 0) {
            edge[i].c -= d;
            edge[i^1].c += d;
            return d;
        }
    }
    dis[now] = -1; // // 炸点优化
    return 0;
}
int Dinic() {
    int sum = 0, d;
    while (bfs()) {
        for (int i = 0; i <= e; ++i) curedge[i] = head[i];
        while (d = dfs(s, inf)) sum += d;
    }
    return sum;
}
```