模板

单源最短路

```
struct cmp {
    bool operator()(state ele_1, state ele_2) {
        return ele_1.second > ele_2.second;
    }
};
int dijkstra(int src, int *cost) {
    memset(vis, 0, sizeof vis);
    priority_queue<state, vector<state>, cmp> pq;
    cost[src] = 0;
    pq.emplace(src, cost[src]);
    while (!pq.empty()) {
        const auto temp = pq.top();
        const int cur_idx = temp.first;
        const int cur_cost = temp.second;
        pq.pop();
        if (vis[cur_idx]) continue;
        vis[cur_idx] = 1;
        for (auto next:edges[cur_idx]) {
            if (next.first == cur_idx) continue;
            const int new_idx = next.first;
            const int new_cost = cur_cost + next.second;
            if (new_cost < cost[new_idx]) {
                pq.emplace(new_idx,new_cost);
                cost[new_idx] = new_cost;
            }
        }
    }
    for (int i = 1; i <= n; i++)
        if (!vis[i]) return false;
    return true;
}
```

塔杨算法

```
无向图版本
void dfs(int pos, int pre) {
    vis[pos] = true;
    low[pos] = dfn[pos] = ++ts;
    s.push(pos);
    for (auto &nex:edges[pos]) {
        if (nex == pre) continue;
        if (!vis[nex]) {
            dfs(nex, pos);
            low[pos] = min(low[pos], low[nex]);
```

```
        } else {
            low[pos] = min(low[pos], dfn[nex]);
        }
    }
    if (low[pos] == dfn[pos] && vis[pos]) {
        ++cnt;
        while (!s.empty() && s.top() != pos) {
            belong[s.top()] = cnt;
            vis[s.top()] = false;
            s.pop();
        }
        if (!s.empty()) s.pop();
        belong[pos] = cnt;
        vis[pos] = false;
    }
}
```

有向图版本

```
void dfs(int pos, int pre) {
    vis[pos] = true;
    low[pos] = dfn[pos] = ++ts;
    s.push(pos);
    for (auto &nex:edges[pos]) {
        if (dfn[nex] == 0) {
            dfs(nex, pos);
            low[pos] = min(low[pos], low[nex]);
        } else if (vis[nex]) {
            low[pos] = min(low[pos], dfn[nex]);
        }
    }
    if (low[pos] == dfn[pos] && vis[pos]) {
        ++cnt;
        while (!s.empty() && s.top() != pos) {
            belong[s.top()] = cnt;
            vis[s.top()] = false;
            s.pop();
        }
        if (!s.empty()) s.pop();
        belong[pos] = cnt;
        vis[pos] = false;
    }
}
```

多元最短路

```
for(int k = 1;k <= n;k++)
    for(int i = 1;i <= n;i ++)
        for(int j = 1;j <= n;j ++)
        {
            if(dis[i][j] > max(dis[i][k],dis[k][j]))
                dis[i][j] = max(dis[i][k],dis[k][j]);
        }
```

SIMPLEX算法

```
int sgn(double x) {
    if (x < -EPS) return -1;
    return x > EPS ? 1 : 0;
}

void pivot(int r, int c) {
    swap(id[n + r], id[c]);
    double x = -a[r][c];
    a[r][c] = -1;
    for (int i = 0; i <= n; ++i) a[r][i] /= x;
    for (int i = 0; i <= m; ++i) {
        if (sgn(a[i][c]) && i != r) {
            x = a[i][c];
            a[i][c] = 0;
            for (int j = 0; j <= n; ++j) a[i][j] += x * a[r][j];
        }
    }
}

int simplex() {
    /* important: revert symbols of conditions */
    /* bug fixed thanks to TuoMianZiGan */
    for (int i = 1; i <= m; ++i) {
        for (int j = 1; j <= n; ++j) {
            a[i][j] *= -1;
        }
    }
    for (int i = 1; i <= n; ++i) id[i] = i;
    /* initial-simplex */
    while (true) {
        int x = 0, y = 0;
        for (int i = 1; i <= m; ++i) {
            if (sgn(a[i][0]) < 0) { x = i; break; }
        }
        if (!x) break;
        for (int i = 1; i <= n; ++i) {
            if (sgn(a[x][i]) > 0) { y = i; break; }
        }
        if (!y) return -1; // infeasible
        pivot(x, y);
    }
    /* solve-simplex */
    while (true) {
        int x = 0, y = 0;
        for (int i = 1; i <= n; ++i) {
            if (sgn(a[0][i]) > 0) { x = i; break; }
        }
        if (!x) break; // finished
        double w = 0, t = 0; bool f = true;
```

```
        for (int i = 1; i <= m; ++i) {
            if (sgn(a[i][x]) < 0) {
                t = -a[i][0] / a[i][x];
                if (f || t < w) {
                    w = t, y = i, f = false;
                }
            }
        }
        if (!y) { return 1; } // unbounded
        pivot(y, x);
    }
    for (int i = 1; i <= n; ++i) v[i] = 0;
    for (int i = n + 1; i <= n + m; ++i) v[id[i]] = a[i - n][0];
    return 0;
}
```

网络流

```
struct Edge {
    int from, to, cap, flow;

    Edge(int u, int v, int c, int f) : from(u), to(v), cap(c), flow(f)
{};
};

struct dinic {
    int n, m, s, t;
    vector<Edge> edges;
    vector<int> G[maxn];
    int d[maxn], cur[maxn], vis[maxn];

    void addedge(int from, int to, int cap) {
        edges.push_back(Edge(from, to, cap, 0));
        edges.push_back(Edge(to, from, 0, 0));
        m = edges.size();
        G[from].push_back(m - 2);
        G[to].push_back(m - 1);
    }

    int bfs() {
        memset(vis, 0, sizeof(vis));
        vis[s] = 1;
        queue<int> q;
        q.push(s);
        while (!q.empty()) {
            int temp = q.front();
            q.pop();
            for (auto nex:G[temp]) {
                Edge &e = edges[nex];
                if (!vis[e.to] && e.cap > e.flow) {
                    vis[e.to] = 1;
```

```
                    d[e.to] = d[temp] + 1;
                    q.push(e.to);
                }
            }
        }
        return vis[t];
    }

    int dfs(int x, int a) {
        if (x == t || a == 0) return a;
        int flow = 0, f;
        for (int i = cur[x]; i < G[x].size(); i++) {
            Edge &e = edges[G[x][i]];
            if (d[x] + 1 == d[e.to] && (f = dfs(e.to, min(a, e.cap -
e.flow))) > 0) {
                e.flow += f;
                edges[G[x][i] ^ 1].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 ans = 0;
        while (bfs()) {
            memset(cur, 0, sizeof(cur));
            ans += dfs(s, INF);
        }
        return ans;
    }

} graph;
```

二分图匹配

```
bool find(int x) {
    for (auto &y : edges[x]) {                          // 遍历每一个可能
的匹配
        if (not used[y]) {                              // 如果我们这一次
没有尝试修改过y
            used[y] = true;
            if (belong[y] == 0 or find(belong[y])) {    // 注意OR运算具有
短路性质，如果y已经匹配过了，递归地尝试给他的现任换一个新的匹配
                belong[y] = x;                          // 没有匹配或者更
换成功，则x和y构成新的匹配
                return true;
```

```
            }
        }
    }
    return false;                                   // 一个都没成功
}

int hungarian() {
    int total = 0;
    for (int i = 0; i < n; ++i) {
        memset(used, 0, sizeof(used));              // 每一次查询前清
空标记
        if (find(i)) {
            ++total;
        }
    }
    return total;
}
```

并查集

```
int find_set(int x)
{
    if(x != par[x])
        par[x] = find_set(par[x]);
    return par[x];
}

void make_set(int x)
{
    par[x] = x;
    rk[x] = 0;
}

void make_union(int x,int y)
{
    if(find_set(x) == find_set(y))
        return;
    else {
        int a = find_set(x);
        int b = find_set(y);
        if(rk[a] < rk[b])
        {
            par[b] = a;
        } else{
            par[a] = b;
            if(rk[a] == rk[b]) rk[b]++;
        }
    }
}
```

MST

```
Kruska
先排序，用并查集，然后遍历所有的边
Prim
使用优先队列，维护到已访问节点的所有点的距离
```