- 链式前向星
- LCA(倍增)
- 求树重心
- Floyd
- Dijkstra(优先队列)
- Dijkstra(暴力)

链式前向星

```
// head[u] 和 cnt 的初始值都为 -1
void add(int u, int v) {
  nxt[++cnt] = head[u];  // 当前边的后继
  head[u] = cnt;         // 起点 u 的第一条边
  to[cnt] = v;           // 当前边的终点
}

// 遍历 u 的出边
for (int i = head[u]; ~i; i = nxt[i]) {  // ~i 表示 i != -1
  int v = to[i];
}
```

LCA(倍增)

```
#include <cstdio>
#include <cstring>
#include <vector>

#define MXN 40005
using namespace std;
std::vector<int> v[MXN];
std::vector<int> w[MXN];

int fa[MXN][31], cost[MXN][31], dep[MXN];
int n, m;
int a, b, c;

// dfs，用来为 lca 算法做准备。接受两个参数：dfs 起始节点和它的父亲节点。
void dfs(int root, int fno) {
  // 初始化：第 2^0 = 1 个祖先就是它的父亲节点，dep 也比父亲节点多 1。
  fa[root][0] = fno;
  dep[root] = dep[fa[root][0]] + 1;
  // 初始化：其他的祖先节点：第 2^i 的祖先节点是第 2^(i-1) 的祖先节点的第
  // 2^(i-1) 的祖先节点。
```

```
  for (int i = 1; i < 31; ++i) {
    fa[root][i] = fa[fa[root][i - 1]][i - 1];
    cost[root][i] = cost[fa[root][i - 1]][i - 1] + cost[root][i - 1];
  }
  // 遍历子节点来进行 dfs。
  int sz = v[root].size();
  for (int i = 0; i < sz; ++i) {
    if (v[root][i] == fno) continue;
    cost[v[root][i]][0] = w[root][i];
    dfs(v[root][i], root);
  }
}

// lca。用倍增算法算取 x 和 y 的 lca 节点。
int lca(int x, int y) {
  // 令 y 比 x 深。
  if (dep[x] > dep[y]) swap(x, y);
  // 令 y 和 x 在一个深度。
  int tmp = dep[y] - dep[x], ans = 0;
  for (int j = 0; tmp; ++j, tmp >>= 1)
    if (tmp & 1) ans += cost[y][j], y = fa[y][j];
  // 如果这个时候 y = x，那么 x，y 就都是它们自己的祖先。
  if (y == x) return ans;
  // 不然的话，找到第一个不是它们祖先的两个点。
  for (int j = 30; j >= 0 && y != x; --j) {
    if (fa[x][j] != fa[y][j]) {
      ans += cost[x][j] + cost[y][j];
      x = fa[x][j];
      y = fa[y][j];
    }
  }
  // 返回结果。
  ans += cost[x][0] + cost[y][0];
  return ans;
}

void Solve() {
  // 初始化表示祖先的数组 fa，代价 cost 和深度 dep。
  memset(fa, 0, sizeof(fa));
  memset(cost, 0, sizeof(cost));
  memset(dep, 0, sizeof(dep));
  // 读入树：节点数一共有 n 个，查询 m 次，每一次查找两个节点的 lca 点。
  scanf("%d %d", &n, &m);
  // 初始化树边和边权
  for (int i = 1; i <= n; ++i) {
    v[i].clear();
    w[i].clear();
  }
  for (int i = 1; i < n; ++i) {
    scanf("%d %d %d", &a, &b, &c);
    v[a].push_back(b);
    v[b].push_back(a);
    w[a].push_back(c);
    w[b].push_back(c);
  }
  // 为了计算 lca 而使用 dfs。
  dfs(1, 0);
```

```
  for (int i = 0; i < m; ++i) {
    scanf("%d %d", &a, &b);
    printf("%d\n", lca(a, b));
  }
}

int main() {
  int T;
  scanf("%d", &T);
  while (T--) Solve();
  return 0;
}
```

求树重心

```
// 这份代码默认节点编号从 1 开始，即 i ∈ [1,n]
int size[MAXN],  // 这个节点的「大小」（所有子树上节点数 + 该节点）
    weight[MAXN],  // 这个节点的「重量」，即所有子树「大小」的最大值
    centroid[2];  // 用于记录树的重心（存的是节点编号）

void GetCentroid(int cur, int fa) {  // cur 表示当前节点 (current)
  size[cur] = 1;
  weight[cur] = 0;
  for (int i = head[cur]; i != -1; i = e[i].nxt) {
    if (e[i].to != fa) {  // e[i].to 表示这条有向边所通向的节点。
      GetCentroid(e[i].to, cur);
      size[cur] += size[e[i].to];
      weight[cur] = max(weight[cur], size[e[i].to]);
    }
  }
  weight[cur] = max(weight[cur], n - size[cur]);
  if (weight[cur] <= n / 2) {  // 依照树的重心的定义统计
    centroid[centroid[0] != 0] = cur;
  }
}
```

Floyd

```
for (k = 1; k <= n; k++) {
  for (x = 1; x <= n; x++) {
    for (y = 1; y <= n; y++) {
      f[x][y] = min(f[x][y], f[x][k] + f[k][y]);
    }
  }
}
```

Dijkstra(优先队列)

```
struct edge {
  int v, w;
};

struct node {
  int dis, u;

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

vector<edge> e[maxn];
int dis[maxn], vis[maxn];
priority_queue<node, vector<node>, greater<node> > q;

void dijkstra(int n, int s) {
  memset(dis, 63, sizeof(dis));
  dis[s] = 0;
  q.push({0, s});
  while (!q.empty()) {
    int u = q.top().u;
    q.pop();
    if (vis[u]) continue;
    vis[u] = 1;
    for (auto ed : e[u]) {
      int v = ed.v, w = ed.w;
      if (dis[v] > dis[u] + w) {
        dis[v] = dis[u] + w;
        q.push({dis[v], v});
      }
    }
  }
}
```

Dijkstra(暴力)

```
struct edge {
  int v, w;
};

vector<edge> e[maxn];
int dis[maxn], vis[maxn];

void dijkstra(int n, int s) {
  memset(dis, 63, sizeof(dis));
```

```
  dis[s] = 0;
  for (int i = 1; i <= n; i++) {
    int u = 0, mind = 0x3f3f3f3f;
    for (int j = 1; j <= n; j++)
      if (!vis[j] && dis[j] < mind) u = j, mind = dis[j];
    vis[u] = true;
    for (auto ed : e[u]) {
      int v = ed.v, w = ed.w;
      if (dis[v] > dis[u] + w) dis[v] = dis[u] + w;
    }
  }
}
```