图论入门

mip001

July 22, 2024

目录

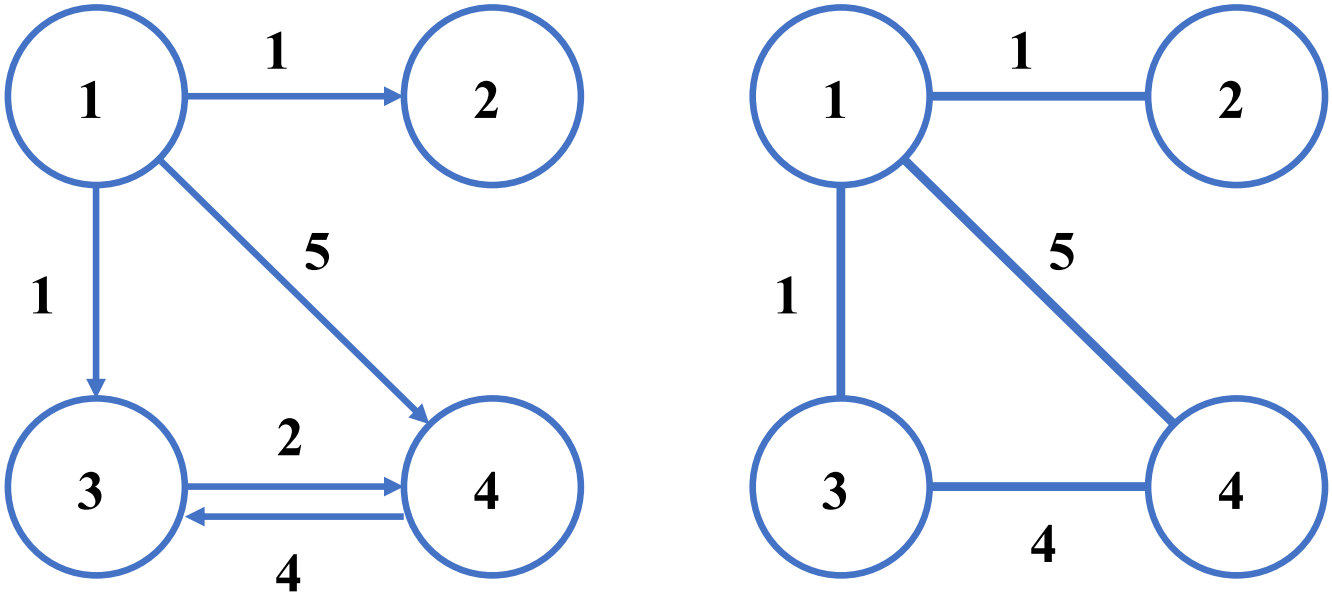
目录

图的基础知识

- 图是由 **点**（vertex）和 **边**（edge）构成的，每一条边连接两个点
- 如果一条边连接相同的两个点，称为 **自环**
- 如果两个边所连接的两个点都相同，称为 **重边**
- **简单图**：没有自环和重边的图
- 点和边都可以带权，称为 **点权**和 **边权**
- 一般情况下，我们用 n 表示一张图中的点数，用 m 来表示边数
- **稀疏图**：$m << n^2$
- **稠密图**：$m \approx n^2$

图的基础知识

左边的是 有向图，右边的是 无向图



- 度：对于无向图而言，与某个结点相连的结点数
- 出度：对于有向图而言，一个顶点向别的顶点连边的条数
- 入度：对于有向图而言，别的顶点连边到一个顶点的条数

树的基础知识

- 树是一种特殊的图
- 一个树有 n 个点，$n-1$ 条边
- 所有点连通
- 没有环
- 任何两个点间只有一条路径

邻接表存图

```cpp
const int maxn = 1e5 + 5;
int n, m, u, v;
vector<int> g[maxn];

// 添加一个 u 到 v 的边
g[u].push_back(v);

// 遍历 u 的出边
for (int v : g[u]) cout << v << " ";
```

邻接表存图（带权）

```cpp
struct edge
{
    int v, w;
};
const int maxn = 1e5 + 5;
int n, m, u, v, w;
vector<edge> g[maxn];

// 添加一个 u 到 v 的权值为 w 的边
g[u].push_back(edge{v, w});

// 遍历 u 的出边
for (auto ed : g[u])
    v = ed.v, w = ed.w;
```

链式前向星存图

```cpp
const int maxn = 1e5 + 5, maxm = 5e5 + 5;
int n, m;
int head[maxn], nxt[maxm], to[maxm], val[maxm], cnt;

// 添加从 u 到 v 的权值为 w 的边
void add(int u, int v, int w)
{
    nxt[++cnt] = head[u];
    head[u] = cnt;
    to[cnt] = v;
    val[cnt] = w;
}

// 遍历 u 的出边
for (int i = head[u]; i; i = nxt[i])
    int v = to[i], w = val[i];
```

目录

树的直径

SP1437 Longest path in a tree

给你一个无权无向的树。编写程序以输出该树中最长路径（从一个节点到另一个节点）的长度。在这种情况下，路径的长度是我们从开始到目的地的遍历边数。$0 < n \leq 10^4$

树的直径

SP1437 Longest path in a tree

给你一个无权无向的树。编写程序以输出该树中最长路径（从一个节点到另一个节点）的长度。在这种情况下，路径的长度是我们从开始到目的地的遍历边数。$0 < n \leq 10^4$

两次 DFS。

第一次从任意一个顶点 x 开始 DFS，找到离 x 最远的顶点 y。

第二次从 y 开始 DFS，找到离 y 最远的顶点 z，y 与 z 的距离即为这棵树的直径。

也可以用树形 DP 解决，可以应对边权为负的情况。

树的重心

定义

如果在无根树中选择某个节点并删除，这棵树将分为若干棵子树，统计子树节点数并记录最大值。取遍树上所有节点，使此最大值取到最小的节点被称为整个树的重心。

- 以树的重心为根时，所有子树的大小都不超过整棵树大小的一半。
- 树的重心如果不唯一，则至多有两个，且这两个重心相邻。
- 树中所有点到某个点的距离和中，到重心的距离和是最小的；如果有两个重心，那么到它们的距离和一样。
- 把两棵树通过一条边相连得到一棵新的树，则新树的重心在连接原来两棵树重心的路径上。
- 在一棵树上添加或删除一个叶子，那么它的重心最多只移动一条边的距离。

树的重心

```cpp
// 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)
        centroid[centroid[0] != 0] = cur; // 依照树的重心的定义统计
}
```

树的重心

还可以用换根 DP 实现

用 f_u 表示所有点到 u 的距离和，sze_u 表示以 1 为根节点时，u 的子树大小。

以 1 为根节点 DFS 一遍，可以求出 sze 和 f_1。

接下来考虑转移，对于所有 u 能到达的点 v，有：

$$f_v = f_u + (sze_1 - sze_v) - sze_v$$

例题：P1364 医院设置

树上公共祖先（LCA）

定义

两个节点的最近公共祖先，就是这两个点的公共祖先里面，离根最远的那个。记点 u 和 v 的 LCA 为 $\mathrm{LCA}(u,v)$

- $\mathrm{LCA}(u,u)=u$
- u 是 v 的祖先，当且仅当 $\mathrm{LCA}(u,v)=u$
- 若 u 不为 v 的祖先且 v 不为 u 的祖先，则 u,v 分别处于 $\mathrm{LCA}(u,v)$ 的两棵不同的子树中
- 两点的 LCA 必定处在树上两点间的最短路上。

LCA 有很多种求法，下面只介绍倍增算法。倍增算法的预处理复杂度为 $O(n\log n)$，单次查询复杂度为 $O(\log n)$

LCA

```cpp
int n, m, s;
vector<int> g[maxn];
int depth[maxn], fa[maxn][25];
void dfs(int u, int father)
{
    depth[u] = depth[father] + 1, fa[u][0] = father;
    for (int i = 1; i <= 20; i++) fa[u][i] = fa[fa[u][i - 1]][i - 1];
    for (int v : g[u]) if (v != father) dfs(v, u);
}
int lca(int u, int v)
{
    if (depth[u] <= depth[v]) swap(u, v);
    int tmp = depth[u] - depth[v];
    for (int i = 0; tmp; i++, tmp>>=1) if (tmp & 1) u = fa[u][i];
    if (u == v) return u;
    for (int i = 20; i >= 0 && u != v; i--)
        if (fa[u][i] != fa[v][i])
            u = fa[u][i], v = fa[v][i];
    return fa[u][0];
}
```

树上差分

P3128 [USACO15DEC] Max Flow P

给定一棵有 n 个结点的树，初始点权都为 0。K 次操作，每次给 s_i 到 t_i 路径上的每一个点的权值加 1，问最后的最大点权。

$2 \leq N \leq 5 \times 10^4, 1 \leq K \leq 10^5$

树上差分

P3128 [USACO15DEC] Max Flow P

给定一棵有 n 个结点的树，初始点权都为 0。K 次操作，每次给 s_i 到 t_i 路径上的每一个点的权值加 1，问最后的最大点权。

$2 \leq N \leq 5 \times 10^4, 1 \leq K \leq 10^5$

对于一个路径 (u, v)，将 u, v 的权值各加 1，再将 $lca(u, v)$ 和 $lca(u, v)$ 的父亲的权值各减 1。

最后做一遍树上前缀和，即做一遍 DFS，从叶子结点开始，把每个点的权值都变成其子树的权值和。

例题

P2680 [NOIP2015 提高组] 运输计划

给你一个 n 个点的树，对于 $n-1$ 条边各有边权，给出 m 个点对 (x,y)，同时定义 $dis(x,y)$ 表示 x,y 两点间的树上距离。

现允许你将一条边的权值变为 0，请你最小化最大的 dis 值。$n,m \leq 3 \times 10^5$

例题

P2680 [NOIP2015 提高组] 运输计划

给你一个 n 个点的树，对于 $n-1$ 条边各有边权，给出 m 个点对 (x,y)，同时定义 $dis(x,y)$ 表示 x,y 两点间的树上距离。

现允许你将一条边的权值变为 0，请你最小化最大的 dis 值。$n,m \leq 3 \times 10^5$

提示：判定问题总是比求解问题更简便

例题

容易发现最后的答案有单调性，考虑二分答案。

每次只需要判断是否存在一条边的边权 w 不小于最长路径减去二分的答案，同时所有比二分值大的路径都覆盖这条边

路径长度可以用树上前缀和 + LCA 求得；对于树上路径覆盖，可以用树上差分。

代码：https://www.luogu.com.cn/paste/b7jp3aa7

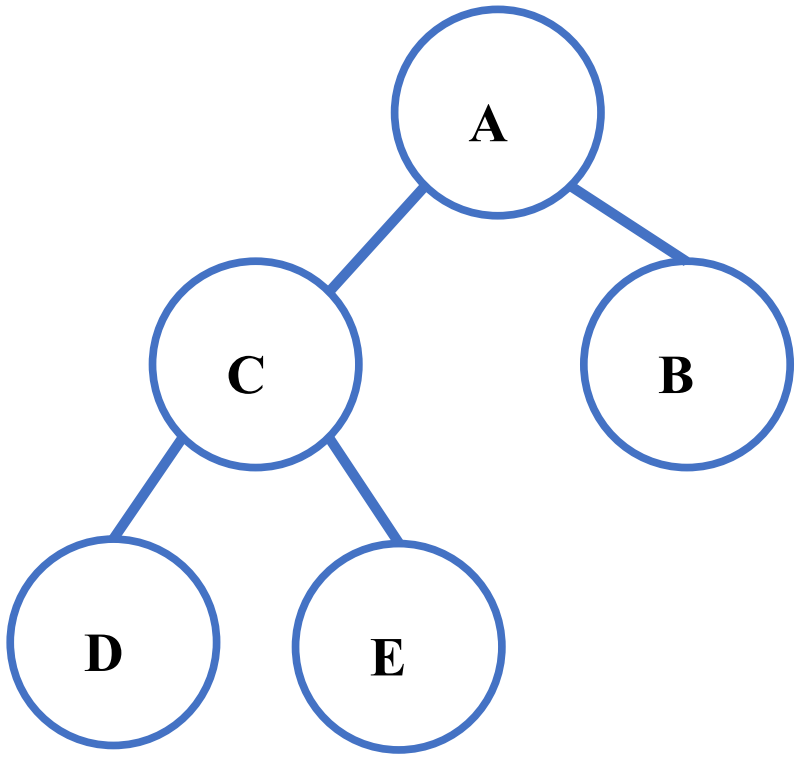
DFS 序

DFS 序指在一棵有根树中，结点在 DFS 中被访问的次序，如右图

结点编号：A C D E B

DFN：1 2 3 4 5

DFS 序使得任何子树在 DFS 序列中都是连续的一段，且该子树的根节点在这一段的开头。这使得对子树的操作可以转化为区间操作。从而可以使用一些数据结构来处理。



DFS 序

LOJ144 DFS 序 1

题目链接：https://loj.ac/p/144

给一棵有根树，这棵树由编号为 $1 \cdots n$ 的 n 个结点组成。根节点的编号为 R。每个结点都有一个权值，结点 i 的权值为 v_i。接下来又 m 组操作，操作分为两类：

- $1\ a\ x$，表示将结点 a 的权值增加 x；
- $2\ a$，表示求结点 a 的子树上所有结点的权值之和。

$n, m \leq 10^6$

DFS 序

```
void dfs(int x, int fa)
{
    st[x] = ++tim;
    for (int i = h[x]; ~i; i = ne[i]) if (e[i] != fa) dfs(e[i], x);
    ed[x] = tim;
    return;
}

void solve()
{
    while (q--)
    {
        int opt, x;
        scanf("%d%d", &opt, &x);
        if (opt == 1)
        {
            int y;
            scanf("%d", &y);
            update(st[x], y);
        }
        else printf("%lld\n", query(ed[x]) - query(st[x] - 1));
    }
}
```

目录

Floyd 算法

全源最短路算法，时间复杂度 (O^3)，实现简单。适用于任何存在最短路的图(无负环)。

```
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]);
        }
    }
}
```

Floyd 算法的实现（摘自 OI-Wiki）

我们定义一个数组 `f[k][x][y]`，表示只允许经过结点 1 到 k（也就是说，在子图 $V' = 1, 2, \ldots, k$ 中的路径，注意，x 与 y 不一定在这个子图中），结点 x 到结点 y 的最短路长度。

很显然，`f[n][x][y]` 就是结点 x 到结点 y 的最短路长度。接下来考虑如何求出 `f` 数组的值。

Floyd 算法的实现（摘自 OI-Wiki）

`f[0][x][y]`：x 与 y 的边权，或者 0，或者 $+\infty$（`f[0][x][y]` 什么时候应该是 $+\infty$？当 x 与 y 间有直接相连的边的时候，为它们的边权；当 $x = y$ 的时候为零，因为到本身的距离为零；当 x 与 y 没有直接相连的边的时候，为 $+\infty$）。

`f[k][x][y] = min(f[k-1][x][y], f[k-1][x][k]+f[k-1][k][y])`
（`f[k-1][x][y]`，为不经过 k 点的最短路径，而 `f[k-1][x][k]+f[k-1][k][y]`，为经过了 k 点的最短路）。

上面两行都显然是对的，所以说这个做法空间是 $O(N^3)$，我们需要依次增加问题规模（k 从 1 到 n），判断任意两点在当前问题规模下的最短路。

传递闭包

B3611 【模板】传递闭包

已知一个有向图中任意两点之间是否有连边，要求判断任意两点是否连通。

传递闭包

B3611 【模板】传递闭包

已知一个有向图中任意两点之间是否有连边，要求判断任意两点是否连通。

```
for (int k = 1; k <= n; k++)
    for (int i = 1; i <= n; i++)
        for (int j = 1; j <= n; j++)
            if(f[i][k]) f[i][j] |= f[k][j];

// 使用 bitset 优化，时间复杂度为 O(n^3/w)
std::bitset<maxn> f[maxn];
for (k = 1; k <= n; k++)
    for (i = 1; i <= n; i++)
        if (f[i][k]) f[i] = f[i] | f[k];
```

无向图最小环

P6175 无向图的最小环问题

给定一张无向图，求图中一个至少包含 3 个点的环，环上的节点不重复，并且环上的边的长度之和最小。

无向图最小环

P6175 无向图的最小环问题

给定一张无向图，求图中一个至少包含 3 个点的环，环上的节点不重复，并且环上的边的长度之和最小。

考虑这个最小环上编号最大的结点 u。

`f[u-1][x][y]` 和 (u,x)、(u,y) 共同构成了这个环。

于是只需要在 Floyd 的过程中枚举 u，计算这个和的最小值即可

Dijkstra 算法

P4779 【模板】单源最短路径（标准版）

给定一个 n 个点，m 条有向边的带非负权图，请你计算从 s 出发，到每个点的距离。

$1 \leq n \leq 10^5$，$1 \leq m \leq 2 \times 10^5$；

Dijkstra 算法

P4779 【模板】单源最短路径（标准版）

给定一个 n 个点，m 条有向边的带非负权图，请你计算从 s 出发，到每个点的距离。

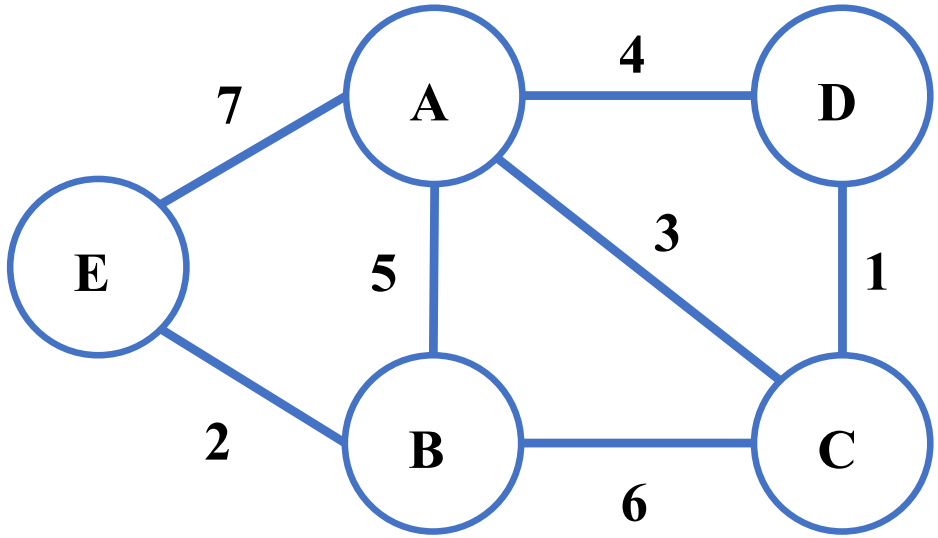
$1 \le n \le 10^5$，$1 \le m \le 2 \times 10^5$；

Dijkstra 算法只适用于非负权图，不同实现方式对应的时间复杂度并不完全相同。朴素实现为 $O(n^2)$，手写堆优化为 $O(m \log n)$，优先队列优化为 $O(m \log m)$

朴素实现适合稠密图，优先队列优化适合稀疏图。

Dijkstra 算法流程

1. 定义数组 dis 代表到目前为止从起点到各个点的最短路径长度。初始时将 dis_s 设置为 0，其它点设置为 $+\infty$。
2. 从所有没有被选择过的点中选择一个 dis 值最小的点，设为 u
3. 对于从 u 出发的每一条边 (u, v, w)，进行一次松弛操作。松弛指的是更新 $dis_v = \min(dis_v, dis_u + w)$。
4. 将点 u 标记为已经被选择过，回到步骤 2，直到不存在一条从已经选择过的点出发到达为选择过的点的边。



Dijkstra 朴素实现

```cpp
struct edge { int v, w; };
vector<edge> e[maxn];
int dis[maxn], vis[maxn];
void dijkstra(int n, int s)
{
    memset(dis, 0x3f, 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;
        }
    }
}
```

堆优化 Dijkstra

```cpp
typedef pair<int, int> pii;
struct edge { int v, w;};
vector<edge> g[maxn];
priority_queue<pii, vector<pii>, greater<pii>> q;
int dis[maxn], vis[maxn];
void dijkstra(int n, int s) {
    memset(dis, 0x3f, sizeof(dis));
    q.push({dis[s] = 0, s});
    while (!q.empty()) {
        int u = q.top().second;
        q.pop();
        if (vis[u]) continue;
        vis[u] = 1;
        for (auto ed : g[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 算法正确性证明

前置证明

最短路径的子路径也是最短路

Dijkstra 算法正确性证明

前置证明

最短路径的子路径也是最短路

$x \to y \to z$ 是 $u \to v$ 的最短路径中的一个子路径，假设 $x \to y \to z$ 不是 $x \to z$ 的最短路，而 $x \to z$ 的最短路为 $x \to p \to z$，那么显然 $u \to x \to p \to z \to v$ 比 $u \to x \to y \to z \to v$ 更优，矛盾。证毕。

Dijkstra 算法正确性证明

前置证明

最短路径的边数不超过 $n-1$

Dijkstra 算法正确性证明

前置证明

最短路径的边数不超过 $n-1$

若最短路径的边数超过 $n-1$，则最短路径中必然有环，由于这是非负权图，显然有环就不优，矛盾。

Dijkstra 算法正确性证明

设已经确定了最短路长度的点构成集合 S（即已经被标记过的点），不在集合 S 里的 dis 最小的点是 u。

设到点 x 的正确最短路长度为 D_x，那么 $\forall x \in S, dis_x = D_x, dis_u \neq D_u$。

我们将从 s 点到 u 点的真正的最短路径拆成 $s \xrightarrow{p_1} x \to y \xrightarrow{p_2} u$，其中 y 是第一个满足 $y \notin S$ 的点，x 是 y 的前驱。(p_1, p_2 为含有若干条边的路径，当然有可能为空)

Dijkstra 算法正确性证明

由于 $s \xrightarrow{p_1} x \to y \xrightarrow{p_2} u$ 是一条最短路径，根据最短路径的子路径也是最短路，所以 $s \xrightarrow{p_1} x \to y$ 也是一条最短路，也就是说 $D_y = D_x + \mathrm{w}(x, y)$。因为 $x \in S$，所以 $dis_x = D_x$，同时也说明 (x, y) 这条边经过了松弛操作，那么 $dis_y \leq dis_x + \mathrm{w}(x, y) = D_x + \mathrm{w}(x, y) = D_y$。但是显然有 $dis_y \geq D_y$，所以 $dis_y = D_y$。

由于边权非负，所以 $dis_y = D_y \leq D_u \leq dis_u$，又 $dis_u \leq dis_y$，所以 $dis_u = dis_y$，那么 $dis_u = D_u$，与假设矛盾。证毕。

Bellman-Ford 算法

P3371 【模板】单源最短路径（弱化版）

给定一个无向图，请你计算从 s 出发，到每个点的距离。

$1 \leq n \leq 10^5$，$1 \leq m \leq 2 \times 10^5$；

Bellman-Ford 算法

P3371 【模板】单源最短路径（弱化版）

给定一个无向图，请你计算从 s 出发，到每个点的距离。

$1 \leq n \leq 10^5$，$1 \leq m \leq 2 \times 10^5$；

只建议在负权图中使用，需要重点注意判定负环存在的边界条件。时间复杂度 $O(nm)$

Bellman-Ford 算法流程

1. 定义数组 dis 代表到目前为止从起点到各个点的最短路径长度。初始时将 dis_s 设置为 0，其它点设置为 $+\infty$。
2. 对于每条边 (u, v, w)，进行一次松弛操作。松弛指的是更新 $dis_v = \min(dis_v, dis_u + w)$
3. 重复步骤 2，直到做了 $n-1$ 遍

算法正确性说明：

Bellman-Ford 算法流程

1. 定义数组 dis 代表到目前为止从起点到各个点的最短路径长度。初始时将 dis_s 设置为 0，其它点设置为 $+\infty$。
2. 对于每条边 (u, v, w)，进行一次松弛操作。松弛指的是更新 $dis_v = \min(dis_v, dis_u + w)$
3. 重复步骤 2，直到做了 $n-1$ 遍

算法正确性说明：

一条最短路径最多包含 $n-1$ 条边，设这些边为 $e_1, e_2, \cdots, e_k$，则在第 i 遍做步骤 2 时至少能够松弛边 e_i。

在 Bellman-Ford 中有大量无效的松弛操作，可以使用队列优化。即 SPFA 算法

SPFA 算法流程

1. 定义数组 dis 代表到目前为止从起点到各个点的最短路径长度。初始时将 dis_s 设置为 0，其它点设置为 $+\infty$。将 s 点放入队列中
2. 从队列中取出一个点 u，对于从 u 出发的每一条边 (u, v, w)，进行一次松弛操作。如果松弛成功且 v 不在队列里，把 v 放入队列中。松弛指的是更新 $dis_v = \min(dis_v, dis_u + w)$
3. 重复步骤 2，直到队列为空。

SPFA 在随机图上表现非常优秀，但是在菊花图中会被卡到 $O(nm)$。

在 NOI 2018 的 D1T1 中，出题人便卡掉了 SPFA。即梗“关于 SPFA，它死了”的来源

SPFA

```cpp
struct edge { int v, w; };
vector<edge> g[maxn];
int dis[maxn], cnt[maxn], vis[maxn];
queue<int> q;
void spfa()
{
    memset(dis, 0x3f, sizeof(dis));
    dis[s] = 0, vis[s] = 1; q.push(s);
    while (!q.empty())
    {
        int u = q.front(); q.pop();
        vis[u] = 0;
        for (auto x : g[u])
        {
            int v = x.v, w = x.w;
            if (dis[v] > dis[u] + w)
            {
                dis[v] = dis[u] + w, cnt[v] = cnt[u] + 1;
                if (cnt[v] >= n) return;
                if (!vis[v]) { q.push(v); vis[v] = 1; }
            }
        }
    }
}
```

目录

负环

P3385 【模板】负环

给定一个 n 个点的有向图，请求出图中是否存在从顶点 1 出发能到达的负环。

负环

P3385 【模板】负环

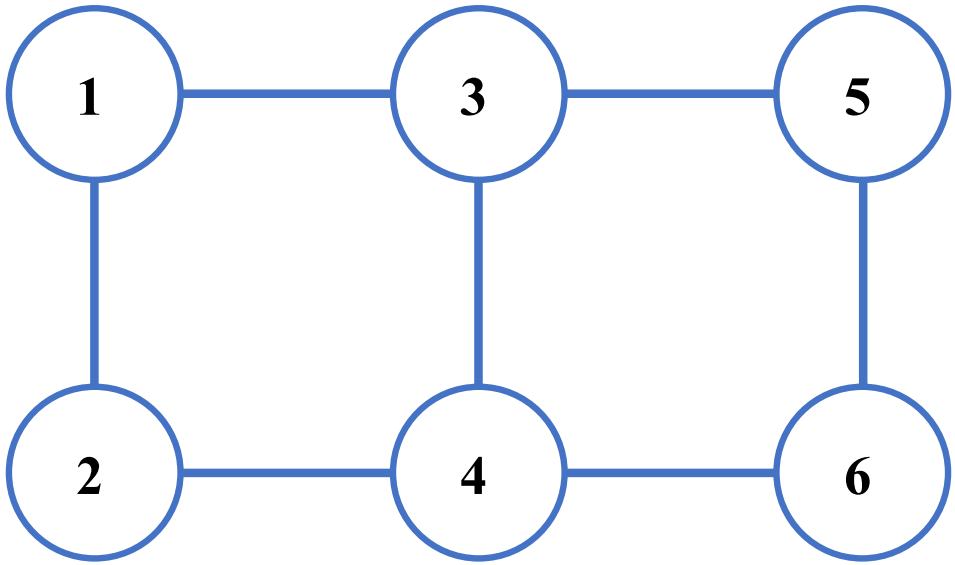
给定一个 n 个点的有向图，请求出图中是否存在从顶点 1 出发能到达的负环。

用 cnt_x 表示 1 到 x 的最短路包含的边数，$cnt_1 = 0$，每次松弛的时候顺便用 $cnt_u + 1$ 更新 cnt_v，若出现 $cnt_v \geq n$，则说明图中有负环。

无边权的最短路

无边权的最短路

直接用 BFS 解决即可。单源的时间复杂度为 $O(n+m)$



0-1 最短路

用于解决边权只有 0 或 1 的图上的单源最短路算法。

0-1 最短路

用于解决边权只有 0 或 1 的图上的单源最短路算法。

使用双端队列，把权值为 0 的边扩展到的点放在队首，把权值为 1 的边扩展到的点放在队尾。这样即可保证像普通 BFS 一样整个队列队首到队尾权值单调不下降。时间复杂度 $O(n+m)$。

拓扑排序

B3644 【模板】拓扑排序 / 家谱树

给定一个 DAG（有向无环图），按拓扑序输出点的编号。

拓扑排序

B3644 【模板】拓扑排序 / 家谱树

给定一个 DAG（有向无环图），按拓扑序输出点的编号。

每次找到入度为 0 的点，删掉与它相连的所有边，再继续相同的操作直到删完所有点。

DAG 的性质：

- 能拓扑排序的图，一定是 DAG
- DAG 一定能拓扑排序

拓扑排序

```
int n, du[maxn];
vector<int> g[maxn];
queue<int> q;
void topo()
{
    for (int u = 1; u <= n; u++)
        for (int v : g[u])
            du[v]++;
    for (int i = 1; i <= n; i++) if (!du[i]) q.push(i);
    while (!q.empty())
    {
        int u = q.front();
        q.pop();
        cout << u << " ";
        for (int v : g[u])
        {
            du[v]--;
            if (!du[v]) q.push(v);
        }
    }
}
```

DAG 的最长（短）路

DAG 的最长（短）路

在 DAG 上可以用 DP 实现 $O(n+m)$ 求最长（短）路。求最长路的转移方程为：

$$dis_v = \max(dis_v, dis_u + w_{u,v})$$

按拓扑序 DP 即可。

拆点

找不到来源的题

有一张 n 个点，m 条边的有向图，点有点权，边有边权。定义一条路径的长度为这条路径经过的所有点的点权和加上经过的所有边的边权和。求 1 号点到 n 号点的最短路。

拆点

找不到来源的题

有一张 n 个点，m 条边的有向图，点有点权，边有边权。定义一条路径的长度为这条路径经过的所有点的点权和加上经过的所有边的边权和。求 1 号点到 n 号点的最短路。

将点拆成入点和出点，从入点向出点连权值为该点点权的边。直接跑一遍起点为 1 号点的入点，终点为 n 号点的出点的单源最短路即可。

分层图

P4822 [BJWC2012] 冻结

有 n 个点，m 条边的有向图，至多可以让 k 条边的权值变为原来的 50%，求 s 到 t 的最短路。$n \leq 5 \times 10^4, k \leq 50$

分层图

P4822 [BJWC2012] 冻结

有 n 个点，m 条边的有向图，至多可以让 k 条边的权值变为原来的 50%，求 s 到 t 的最短路。$n \leq 5 \times 10^4, k \leq 50$

把图分成 k 层，每层仍然是 n 个点，每层的连边方式也与原图相同。对于当前层，若有一个边 (u, v, w)，那么对于下一层，仍有一个边 (u', v', w)，并且，应该再连一条 $(u, v', \frac{w}{2})$ 的边。

然后只需要对 s 跑一遍全图的最短路，最后答案就是 t 点在每一层的最短路的最小值

分层图

P5060 旅行

有 n 个点，m 条边的有向图，求从 s 到 t 的路径上的边权和是 p 的倍数的最短路径的长度。$n \leq 5 \times 10^4, p \leq 50$

分层图

P5060 旅行

有 n 个点，m 条边的有向图，求从 s 到 t 的路径上的边权和是 p 的倍数的最短路径的长度。$n \leq 5 \times 10^4, p \leq 50$

仍然把图分成 p 层，但是每层不按原图连边，第 i 层表示到这一层的点时，边权和模 p 余 i。对于一条边 (u, v, w)，枚举 u 所在的层数 i，向第 $(i + w) \bmod p$ 层的 v' 连一条权值为 w 的边。

这样问题就被转换成了求第 0 层的 s 到 t 的最短路。

多源最短路

HDU2680 Choose the best route

题目链接：https://acm.hdu.edu.cn/showproblem.php?pid=2680

n 个点，m 条有向边，每条路都有时间花费，给一个终点，多个起点，问从起点到终点的最小时间花费。$n \leq 5 \times 10^3$

多源最短路

HDU2680 Choose the best route

题目链接：https://acm.hdu.edu.cn/showproblem.php?pid=2680

n 个点，m 条有向边，每条路都有时间花费，给一个终点，多个起点，问从起点到终点的最小时间花费。$n \leq 5 \times 10^3$

建反图，对终点跑一次单源最短路

建一个超级源点，向所有起点连一个边权为 0 的有向边，对超级源点跑一边单源最短路。

最短路计数

P1144 最短路计数

给出一个 N 个顶点 M 条边的无向无权图，顶点编号为 $1 \sim N$。问从顶点 1 开始，到其他每个点的最短路有几条。（请用 Dijkstra 做）

对于 100% 的数据，$1 \leq N \leq 10^6$，$1 \leq M \leq 2 \times 10^6$。

最短路计数

P1144 最短路计数

给出一个 N 个顶点 M 条边的无向无权图，顶点编号为 $1 \sim N$。问从顶点 1 开始，到其他每个点的最短路有几条。（请用 Dijkstra 做）

对于 100% 的数据，$1 \leq N \leq 10^6$，$1 \leq M \leq 2 \times 10^6$。

正常跑一遍 Dijkstra，记数组 cnt_i 表示从起点到 i 点的最短路径数。每次松弛操作时，若 $dis_v > dis_u + 1$，则用 $dis_u + 1$ 来更新 dis_v，并令 $cnt_v = cnt_u$；若 $dis_v = dis_u + 1$，则将 cnt_v 加上 cnt_u，因为在这种情况下，经过 u 的最短路一定也都是经过 v 的最短路。

二进制分组最短路

P5304 [GXOI/GZOI2019] 旅行者

给定 n 个点，m 条边的有向图，里面有 k 个特殊点，问这 k 个点之间两两最短路的最小值。$n \leq 5 \times 10^5, m \leq 5 \times 10^5$

二进制分组最短路

P5304 [GXOI/GZOI2019] 旅行者

给定 n 个点，m 条边的有向图，里面有 k 个特殊点，问这 k 个点之间两两最短路的最小值。$n \leq 5 \times 10^5, m \leq 5 \times 10^5$

假设我们把特殊点分成 A, B 两个集合，新建 s 连 A 集合的所有点，边权 0，新建 t 连接 B 集合里的所有点，边权 0，那么 s 到 t 的最短路就是 A, B 集合点之间的最短路的最小值。

那么对于 k 个特殊点，我们枚举二进制里的第 i 位，把二进制第 i 位是 0 的点放在 A，1 的点放在 B，用以上方法跑一个最短路。然后跑 $log\ n$ 次最短路之后，所有最短路的最小值就是最终答案。

原理是，假设 k 个特殊点里最近的是 x 和 y，那么 x 和 y 一定有一个二进制位不一样，那么他们肯定在那次分组的时候被放进了不同的集合，从而肯定被算进了最后的答案之中最短路。

线段树优化建图

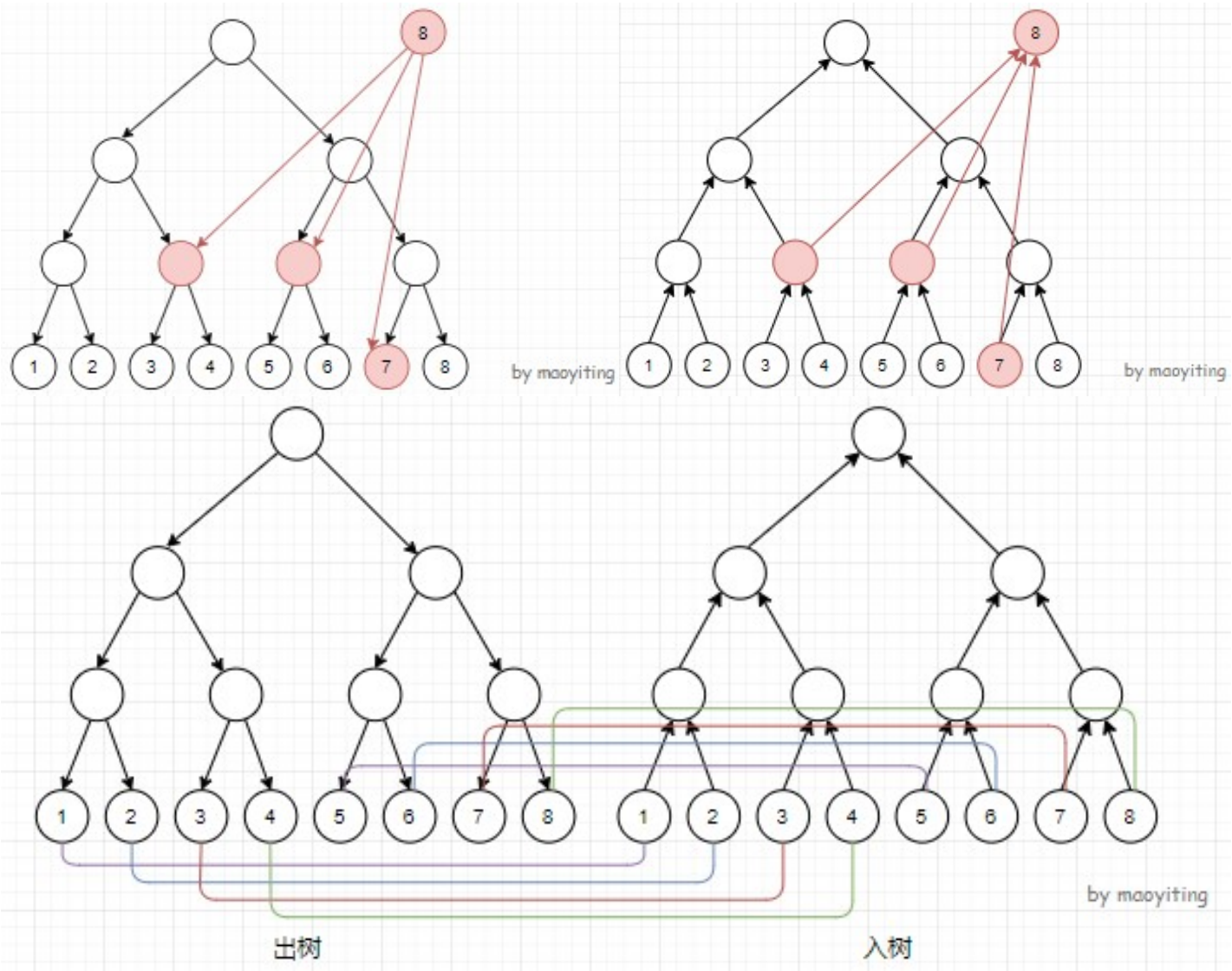
CF786B Legacy

有 n 个点、q 次操作。每一种操作为以下三种类型中的一种：

- 操作一：连一条 $u \to v$ 的有向边，权值为 w。
- 操作二：对于所有 $i \in [l, r]$ 连一条 $u \to i$ 的有向边，权值为 w。
- 操作三：对于所有 $i \in [l, r]$ 连一条 $i \to u$ 的有向边，权值为 w。

求从点 s 到其他点的最短路。$1 \le n, q \le 10^5, 1 \le w \le 10^9$。

线段树优化建图


8
1 2 3 4 5 6 7 8
by maoyiting
8
1 2 3 4 5 6 7 8
by maoyiting
1 2 3 4 5 6 7 8
1 2 3 4 5 6 7 8
by maoyiting
出树
入树

差分约束

P5960 【模板】差分约束

给出一组包含 m 个不等式，有 n 个未知数的形如：

$$\begin{cases} x_{c_1} - x_{c'_1} \leq y_1 \\ x_{c_2} - x_{c'_2} \leq y_2 \\ \cdots \\ x_{c_m} - x_{c'_m} \leq y_m \end{cases}$$

的不等式组，求任意一组满足这个不等式组的解。$1 \leq n, m \leq 5 \times 10^3$，$-10^4 \leq y \leq 10^4$，$1 \leq c, c' \leq n$，$c \neq c'$。

差分约束

考虑移项，把不等式 $x_{c_i} - x_{c'_i} \leq y_i$ 变形成 $x_{c_i} \leq x_{c'_i} + y_i$，容易发现这和单源最短路中的三角不等式 $dis_v \leq dis_u + w$ 非常相似。因此，我们可以把 x 看作图中的顶点，对于每个不等式 $x_{c_i} \leq x_{c'_i} + y_i$，从结点 c'_i 向结点 c_i 连一条长度为 y_i 的有向边。

新建一个 0 号点，设 $dis_0 = 0$，并向每个点连一条权值为 0 的边。这个操作相当于新增变量 x_0 和 n 个不等式 $x_i \leq x_0$，从而可以将所有变量都和 x_0 联系起来

以 0 为源点跑 SPFA，若图中存在负环，则给定的不等式组无解。否则 $x_i = dis_i$ 即为一组可行解。

习题

1. P1346 电车
2. P1119 灾后重建
3. P2829 大逃离
4. P2149 [SDOI2009]Elaxia 的路线
5. P2151 [SDOI2009]HH 去散步
6. CF1076D Edge Deletion
7. P3573 [POI2014]RAJ-Rally
8. P4745 [CERC2017]Gambling Guide
9. CF827F Dirty Arkady's Kitchen
10. P1993 小 K 的农场
11. P7515 [省选联考 2021 A 卷] 矩阵游戏
12. P3199 [HNOI2009] 最小圈
13. P2371 [国家集训队] 墨墨的等式

目录

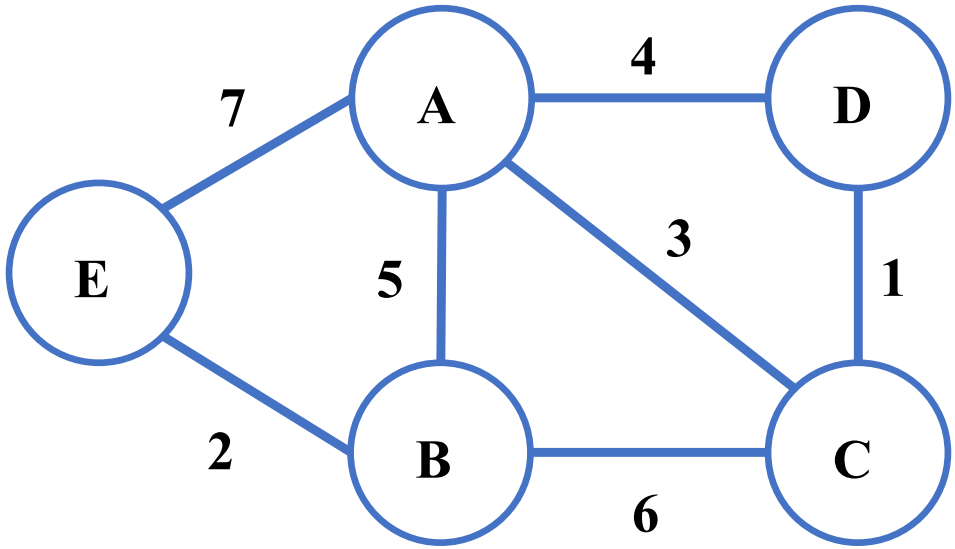
最小生成树

在一张 n 个点的图中，我们选择 $n-1$ 条边，使得这 n 个点与这 $n-1$ 条边构成一棵树，那么这棵树就叫做这个图的一棵生成树。

特别的，根据树的性质，上面的表述与以下表述等价：

1. 选择 $n-1$ 条边并且不出现环；
2. 选择 $n-1$ 条边并且连通。

当一棵生成树的所有边权之和最小时，称为最小生成树。



Prim 算法流程

1. 建立 dis 数组代表当前生成树中的点到该点的最短边长。
2. 将 1 号点加入生成树中。
3. 并且更新所有与刚刚加入生成树中的点相连的点的 dis 值。
4. 选择未加入生成树的点中 dis 值最小的，加入生成树中，然后回到步骤 3，直到所有点都被加入。

时间复杂度 $O(n^2)$。

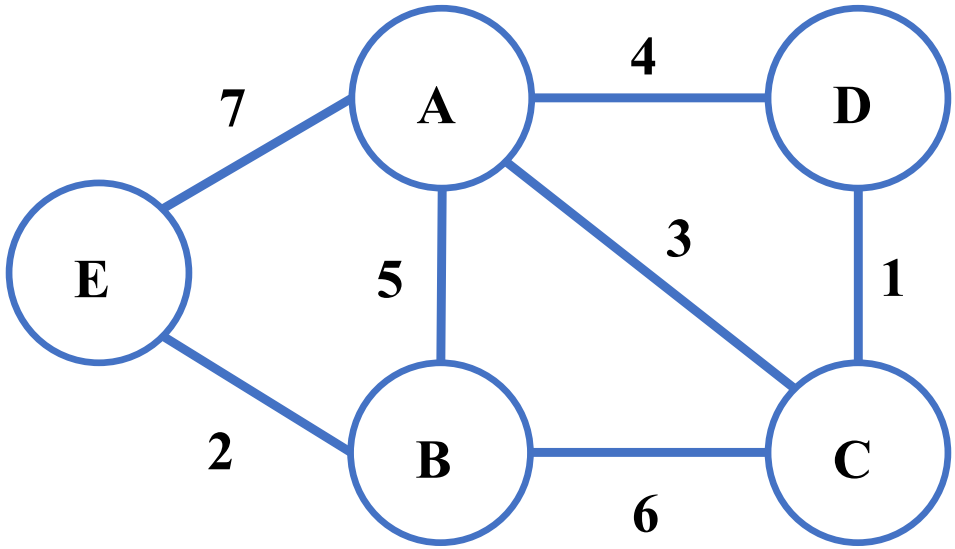
注意到 Prim 算法与 Dijkstra 算法非常相似。所以就不展示代码了。

Kruskal 算法流程

1. 把所有边按照边权从小到大排序。
2. 按顺序处理每一条边，如果加入这条边不会出现环，那么就加入这条边。
3. 直到已经加入 $n-1$ 条边

时间复杂度 $O(m \log m)$，瓶颈为按边权排序。

注意判环需要使用并查集。



Kruskal

```
struct edge
{
    int u, v, w;
    bool operator<(const edge &e) const { return w < e.w; }
} a[maxn];
int kruskal()
{
    int cnt = 0, sum = 0;
    sort(a + 1, a + m + 1);
    for (int i = 1; i <= m; i++)
    {
        if (findfa(a[i].u) != findfa(a[i].v))
        {
            merge(a[i].u, a[i].v);
            sum += a[i].w;
            cnt++;
            if (cnt == n - 1)
                break;
        }
    }
    return sum;
}
```

P1194 买礼物

你要购买 B 件原价为 A 元的物品，有若干促销活动，每个促销活动 $K_{i,j}$ 代表如果已经购买了第 i 件物品，那么再购买第 j 件物品只需要花 $K_{i,j}$ 元，如果 $K_{i,j}=0$，那么 i,j 两件物品间没有促销活动。

问购买所有物品的最小开销是多少元。$B \leq 500, A \leq 10^3, K_{i,j} \leq 10^3$

最小生成树

P1194 买礼物

你要购买 B 件原价为 A 元的物品，有若干促销活动，每个促销活动 $K_{i,j}$ 代表如果已经购买了第 i 件物品，那么再购买第 j 件物品只需要花 $K_{i,j}$ 元，如果 $K_{i,j}=0$，那么 i,j 两件物品间没有促销活动。

问购买所有物品的最小开销是多少元。$B \leq 500, A \leq 10^3, K_{i,j} \leq 10^3$

考虑根据各个物品间的促销活动来建图，边权代表优惠后的价格。例如，假如 $K_{i,j}=a$，那么就在 i 号点与 j 号点间连一条边权为 a 的无向边

最后再建一个源点，往其他每个点连一条边权为 A 的边。

这样建好图后，这个图的一棵生成树就代表一种购买方案。

求最小生成树即可。

Kruskal 重构树

P1967 [NOIP2013 提高组] 货车运输

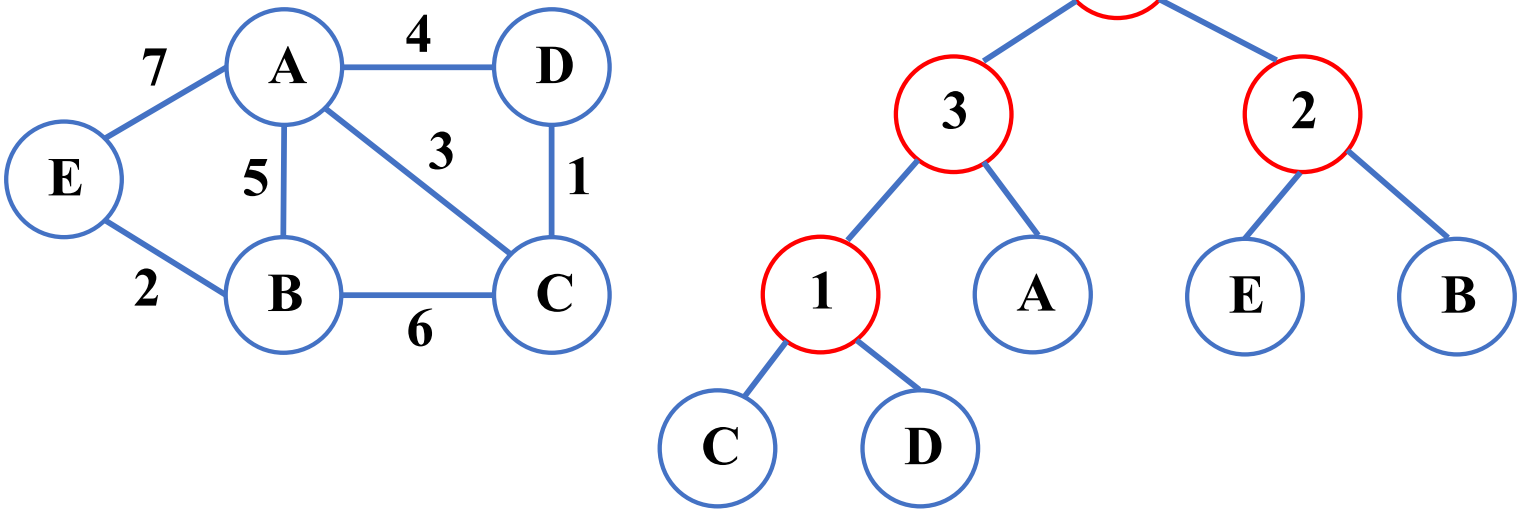
给定无向图，多次询问 u 到 v 路径上最大边权的最小值。

$n \leq 10^4, m \leq 5 \times 10^4, q \leq 3 \times 10^4$

Kruskal 重构树

在 Kruskal 的过程中，若当前边所连两点 u 和 v 不在一个集合内，则新建一个结点，这个点的点权为当前边的边权，然后分别连接这个点和 u, v 所在集合的根。重构出来的树有以下性质：

- 是大根堆，即祖先节点的权值一定大于后代（因为祖先节点后加入）
- 从 u 到 v 路径上最大边权的最小值为 $\mathrm{LCA}(u, v)$



Kruskal 重构树

P4768 [NOI2018] 归程

n 个点，m 条边，保证图联通，每条边有两个权值，一个长度，一个海拔，多组询问，告诉你起点和水位线，海拔小于等于水位线的边都会被淹没，只能走路，否则可以开车，只要下车了后面就都只能走路，问从当天起点到 1 号节点最少步行经过的长度，询问强制在线。$n \leq 2 \times 10^5, m \leq 4 \times 10^5, Q \leq 4 \times 10^5$

Kruskal 重构树

P4768 [NOI2018] 归程

n 个点，m 条边，保证图联通，每条边有两个权值，一个长度，一个海拔，多组询问，告诉你起点和水位线，海拔小于等于水位线的边都会被淹没，只能走路，否则可以开车，只要下车了后面就都只能走路，问从当天起点到 1 号节点最少步行经过的长度，询问强制在线。$n \leq 2 \times 10^5, m \leq 4 \times 10^5, Q \leq 4 \times 10^5$

以海拔为边权跑最大生成树，同时建立 Kruskal 重构树，这样就有一个对于海拔的小根堆。然后对于每一棵子树，如果询问中的水位线是低于子树根节点的点权的，那么此时这棵子树中的任意一个点都可以花费 0 的代价到达子树中其它点。

那么对于一次询问，我们找到这个重构树中的一个点 u，满足起点在以这个点 u 为根的子树中，且它的点权刚好大于给定的水位线。然后我们只需要找到这棵子树中所有叶子结点到 1 号点的最小花费，就是本次询问的答案。

可以使用倍增算法找到点 u。

习题

1. P1195 口袋的天空
2. P1550 [USACO08OCT]Watering Hole G
3. P2498 [SDOI2012] 拯救小云公主
4. P2700 逐个击破
5. P4180 [BJWC2010] 严格次小生成树
6. P3623 [APIO2008] 免费道路
7. CF76A Gift
8. CF609E Minimum spanning tree for each edge
9. P1967 [NOIP2013 提高组] 货车运输
10. P5633 最小度限制生成树

目录

一些概念

- 连通图：无向图 G 中，若任意两点之间都有路径，则该图为连通图
- 连通分量：无向图 G 的极大连通子图。连通图只有一个连通分量。
- 强连通图：有向图 G 中，若任意两点都可互相到达，则该图为强连通图
- **强连通分量（SCC）**：有向图 G 的极大强连通子图
- **割点**：无向连通图删除某点后，使整个图变为不连通的两部分的点
- **割边（桥）**：无向连通图删除某边后，使整个图变为不连通的两部分的边
- 点双连通图：不存在割点的图，即无向图中删掉任意一个结点图仍连通
- 边双连通图：不存在桥的图，即无向图中删掉任意一条边图仍连通
- **点双连通分量（V-DCC）**：无向图的极大点双连通子图
- **边双连通分量（E-DCC）**：无向图的极大边双连通子图

DFS 树

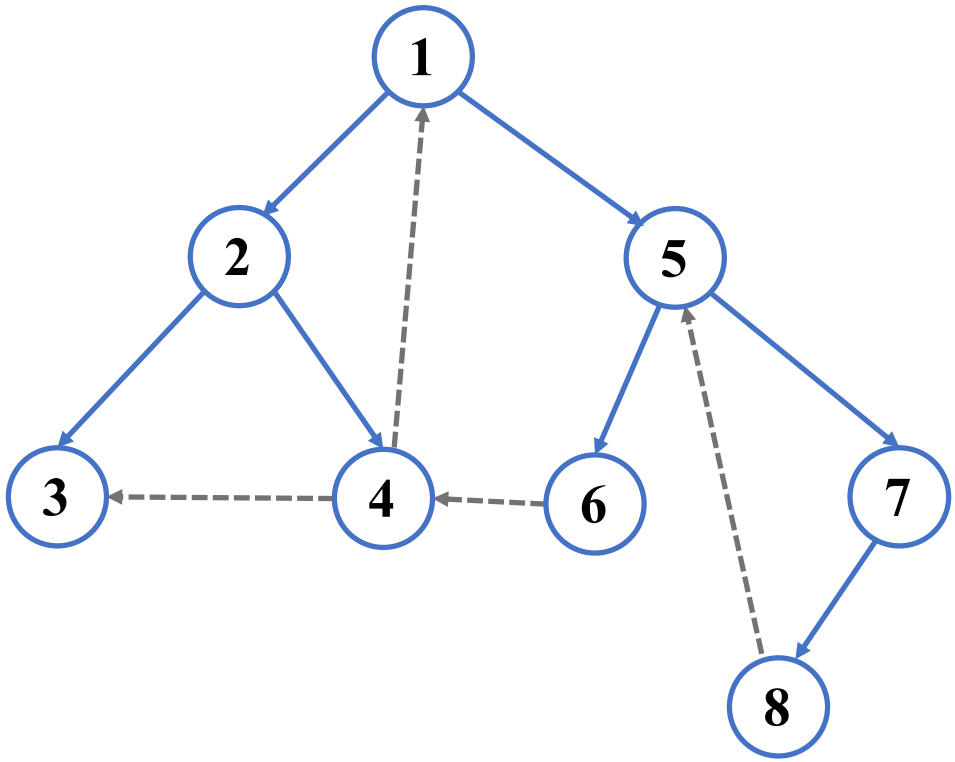
一张图的 DFS 树是在其进行深度优先遍历时，所形成的树结构。建立了 DFS 树后，图上的边可以分成几类：

- 树边：每个点到其所有孩子结点的边，也即每个点第一次被访问时经过的边
- 对于有向图：
  - 前向边：每个点到其后代的边，不包括树边
  - 返祖边（后向边）：每个点到其祖先的边
  - 横叉边：连接没有祖先关系的点的边
- 对于无向图：
  - 返祖边：连接有祖先关系的点的边
  - 无向图没有横叉边

DFS 树

返祖边和横叉边的起点的 DFS 序一定大于终点的 DFS 序

对于每一个强连通分量，存在一个点是其它所有点的祖先。考虑反证，假设有结点 v 在一个强连通分量中但不在以 u 为根的子树中，那么 u 到 v 的路径中一定有一条离开子树的边，且这个边只能是非树边，然而非树边只能指向被访问过的点，和 u 是第一个被访问的点矛盾。



Tarjan 算法求强连通分量

在 Tarjan 算法中为每个结点 u 维护了以下几个变量：

1. dfn_u：u 的 DFS 序。显然一个结点的子树内结点的 dfn 都大于该结点的 dfn。
2. low_u：在 u 的子树中能够回溯到的最早的已经在栈中的结点。low_u 定义为以下结点的 dfn 的最小值：u 的子树中的结点；从 u 的子树通过一条非树边能到达的结点。

从根开始的一条路径上的 dfn 严格递增，low 严格非降。

Tarjan 算法求强连通分量

按照 DFS 序进行搜索，每次让搜索到的结点入栈。每当找到一个强连通分量的根节点，就按照该元素包含结点数目让栈中元素出栈。在搜索过程中，对于结点 u 和它能够一步到达的结点 v 考虑 3 种情况：

1. v 未被访问：继续对 v 进行深度搜索。在回溯过程中，用 low_v 更新 low_u。因为存在从 u 到 v 的直接路径，所以 v 能够回溯到的已经在栈中的结点，u 也一定能够回溯到。
2. v 被访问过，已经在栈中：根据 low 值的定义，用 dfn_v 更新 low_u。
3. v 被访问过，已不在栈中：说明 v 已搜索完毕，其所在连通分量已被处理，不用进行操作。

Tarjan 算法求强连通分量

```
int dfn[N], low[N], dfncnt, s[N], in_stack[N], tp;
int scc[N], sc;  // 结点 i 所在 SCC 的编号
int sz[N];       // 强连通 i 的大小
void tarjan(int u) {
    low[u] = dfn[u] = ++dfncnt, s[++tp] = u, in_stack[u] = 1;
    for (int i = h[u]; i; i = e[i].nex) {
        const int &v = e[i].t;
        if (!dfn[v]) {
            tarjan(v);
            low[u] = min(low[u], low[v]);
        } else if (in_stack[v]) {
            low[u] = min(low[u], dfn[v]);
        }
    }
    if (dfn[u] == low[u]) {
        ++sc;
        while (s[tp] != u) {
            scc[s[tp]] = sc, in_stack[s[tp]] = 0;
            sz[sc]++; --tp;
        }
        scc[s[tp]] = sc, in_stack[s[tp]] = 0;
        sz[sc]++; --tp;
    }
}
```

缩点

P3387 【模板】缩点

给定一个 n 个点 m 条边有向图，每个点有一个权值，求一条路径，使路径经过的点权值之和最大。你只需要求出这个权值和。允许多次经过一条边或者一个点，但是，重复经过的点，权值只计算一次。

$1 \leq n \leq 10^4$，$1 \leq m \leq 10^5$，$0 \leq a_i \leq 10^3$。

缩点

P3387 【模板】缩点

给定一个 n 个点 m 条边有向图，每个点有一个权值，求一条路径，使路径经过的点权值之和最大。你只需要求出这个权值和。允许多次经过一条边或者一个点，但是，重复经过的点，权值只计算一次。

$1 \le n \le 10^4$，$1 \le m \le 10^5$，$0 \le a_i \le 10^3$。

对于同一个强连通分量里的点，由于它们都可以互相到达，显然可以贪心地把它们全部走完。

所以我们可以将每一个强连通分量都缩为一个点，点权为这个 SCC 里所有点的点权之和。对于原图中有连边的两个点 u 和 v，若它们分别处于不同的 SCC，就在这两个 SCC 缩成的两个点间连上一条相同的边。

这样之后按照 DAG 求最长路相似的方式进行 DP 即可。

Tarjan 算法求割点

还是定义两个数组 dfn 和 low。然后开始 DFS

如果对于某个顶点 u，存在至少一个儿子 v，使得 $low_v \geq dfn_u$，即不能不经过 u 点回到祖先节点，那么 u 为割点。

需要特殊考虑根节点。如果根节点有至少两个儿子，则它是割点，否则就不是。

板子题：P3388 【模板】割点（割顶）

Tarjan 算法求割点

```
void tarjan(int u, int fa)
{
    dfn[u] = low[u] = ++cnt;
    st.push(u);
    int son = 0, flag = false;
    for (int v : g[u])
    {
        if (!dfn[v])
        {
            ++son;
            tarjan(v, u);
            low[u] = min(low[u], low[v]);
            if (fa && low[v] >= dfn[u])
            {
                flag = true;
                while (st.top() != u) st.pop();
            }
        }
        else if (v != fa) low[u] = min(low[u], dfn[v]);
    }
    if (fa == 0 && son > 1) flag = true;
    if (flag) ++tot, ans.push_back(u);
}
```

Tarjan 算法求割边（桥）

过程和求割点差不多，只需要把 `low[v]>=dfn[u]` 改为 `low[v]>dfn[u]` 即可，且不需要再考虑根节点的问题。

双连通分量

- 点双连通：每条边恰好属于一个点双连通分量，每个点可能属于多个。分量之间由公共点（割点）连接
- 边双连通：每个点恰好属于一个边双连通分量，分量之间由割边连接，缩点后成为一棵树
- 边双连通具有传递性，即若 x, y 边双连通，y, z 边双连通，则 x, z 边双连通
- 点双连通不具有传递性，即 A, B 点双连通，B, C 点双连通，A, C 不一定点双连通

求双连通分量只需要在求割点或割边的同时，每次把出栈的点记录下来即可。

两个板子：P8435【模板】点双连通分量、P8436【模板】边双连通分量

习题

- P2002 消息扩散
- P3627 [APIO2009] 抢掠计划
- P2812 校园网络【[USACO]Network of Schools 加强版】
- P3243 [HNOI2015] 菜肴制作
- P5008 [yLOI2018] 锦鲤抄
- P1477 [NOI2008] 假面舞会
- P3469 [POI2008]BLO-Blockade
- P3225 [HNOI2012] 矿场搭建

目录

欧拉图

- 欧拉回路：通过图中每条边恰好一次的回路（起点和终点相同）
- 欧拉图：具有欧拉回路的图

判别：

1. 无向图是欧拉图当且仅当：
   - 非零度顶点是连通的
   - 顶点的度数都是偶数
2. 有向图是欧拉图当且仅当：
   - 非零度顶点是强连通的
   - 每个顶点的入度和出度相等

半欧拉图

- 欧拉通路：通过图中每条边恰好一次的通路
- 半欧拉图：具有欧拉通路但不具有欧拉回路的图

判别：

1. 无向图是半欧拉图当且仅当：
   - 非零度顶点是连通的
   - 恰有 2 个奇度顶点
2. 有向图是半欧拉图当且仅当：
   - 非零度顶点是弱连通的
   - 至多一个顶点的出度与入度之差为 1
   - 至多一个顶点的入度与出度之差为 1
   - 其他顶点的入度和出度相等

求欧拉路

直接 DFS 即可。

```
void dfs(int u)
{
    for (int v = minn; v <= maxn; v++)
    {
        if (g[u][v])
        {
            g[u][v]--, g[v][u]--;
            dfs(v);
        }
    }
    S.push(u); //记录路径
}
```

这里用的邻接矩阵，其实更推荐 vector 邻接表存图

习题

- P2731 骑马修栅栏
- P1127 词链
- P1333 瑞瑞的木棍
- P1341 无序字母对
- P6066 [USACO05JAN]Watchcow S
- P6628 [省选联考 2020 B 卷] 丁香之路

哈密顿图

- 通过图中所有顶点一次且仅一次的通路称为哈密顿通路。
- 通过图中所有顶点一次且仅一次的回路称为哈密顿回路。
- 具有哈密顿回路的图称为哈密顿图。
- 具有哈密顿通路而不具有哈密顿回路的图称为半哈密顿图。

已经证明“判断一个图存不存在哈密顿路径”是 NPC 的，即目前不存在算法能在多项式时间内判断。

二分图

定义：结点由两个集合组成，且两个集合内部没有边的无向图。

性质：

- 如果两个集合中的点分别染成黑色和白色，可以发现二分图中的每一条边都一定是连接一个黑色点和一个白色点。
- 二分图不存在长度为奇数的环。因为每一条边都是从一个集合走到另一个集合，只有走偶数次才可能回到同一个集合。

思考：树是不是二分图？

二分图染色

BFS 或 DFS 对图进行黑白染色。

对于结点 u，枚举与它相连点 v，如果 v 未被染色，则个它染上与 u 相反的颜色；如果 v 已染色，且颜色与 u 相同，则说明出现了奇环，二分图不存在。

所以二分图染色可以用来判定二分图。

二分图染色

```
bool bfs(int x)
{
    queue<int> q;
    col[x] = 0;
    q.push(x);
    while (!q.empty())
    {
        int u = q.front();
        q.pop();
        for (int e = first[u]; e; e = nxt[e])
        {
            int v = to[e];
            if (col[v] == -1)
            {
                col[v] = col[u] ^ 1;
                q.push(v);
                continue;
            }
            if (col[v] == col[u]) return false;
        }
    }
    return true;
}
```

例题

CF741C

有 $2n$ 个人围成一圈坐在桌子边上，每个人占据一个位子，对应这 $2n$ 个人是 n 对情侣，要求情侣不能吃同一种食物，并且桌子上相邻的三个人的食物必须有两个人是不同的，只有两种食物（1 或者是 2），问一种可行分配方式。

例题

CF741C

有 $2n$ 个人围成一圈坐在桌子边上，每个人占据一个位子，对应这 $2n$ 个人是 n 对情侣，要求情侣不能吃同一种食物，并且桌子上相邻的三个人的食物必须有两个人是不同的，只有两种食物（1 或者是 2），问一种可行分配方式。

把“相邻的三个人的食物必须有两个人是不同的”转化一下，将 $2i$ 和 $2i-1$ 之间连边，表示他们两个吃的不一样。再在每对情侣之间连边。可以发现这样建出来的图一定是二分图。

我们假设 $2i$ 和 $2i-1$ 之间有另一条路径，因为一个点只会有一个对应的情侣，所以这条路径一定是由 x 条情侣之间的边和 $x-1$ 条相邻点之间的边组成的，再加上这条边，所以这个环就是偶环。

然后二分图染色即可。

目录

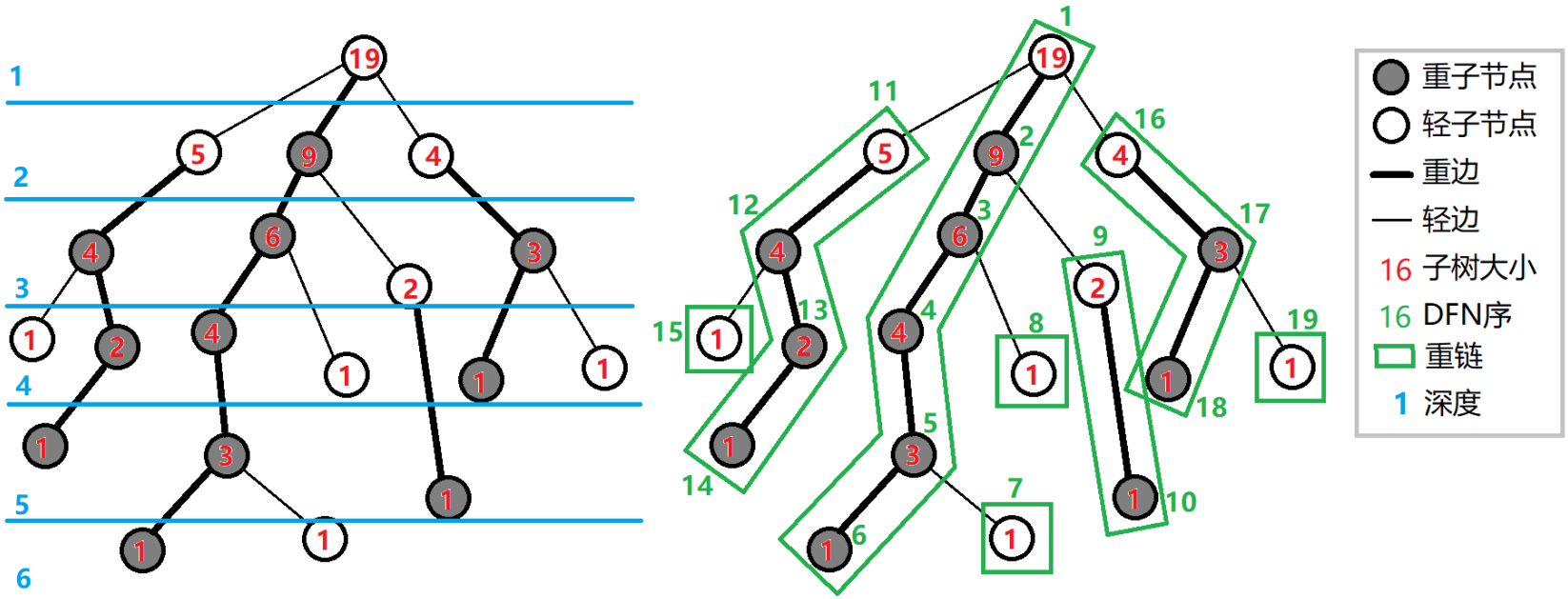
树链剖分

树链剖分是把一棵树分割成若干条链，以便于维护信息的一种方法，其中最常用的是重链剖分，所以一般提到树链剖分或树剖都是指重链剖分。

板子很简单，但是题不太简单。

一些概念

- 重儿子：子节点中子树节点个数最大的节点。如果有多个的话，取其中一个。如果没有子节点，那么就没有重儿子。
- 轻儿子：子节点中除了重儿子以外的其他节点。
- 重边：父亲节点与重儿子相连的边。
- 轻边：父亲节点和轻儿子相连的边。
- 重链：由多条重边连接的路径。注意：落单的节点一般算作单独一条重链。
- 轻链：由多条轻边连接的路径。



实现

```
//预处理出 fa[x], dep[x], siz[x], son[x]
//son[x] 用来记录结点 x 的重儿子
void dfs1(int o)
{
    son[o] = -1;
    siz[o] = 1;
    for (int j = h[o]; j; j = nxt[j])
        if (!dep[p[j]])
        {
            dep[p[j]] = dep[o] + 1;
            fa[p[j]] = o;
            dfs1(p[j]);
            siz[o] += siz[p[j]];
            if (son[o] == -1 || siz[p[j]] > siz[son[o]])
                son[o] = p[j];
        }
}
```

实现

```
// 求出 top[x], dfn[x], rnk[x]
// top[x] 表示结点 x 所在的重链的顶部结点
// rnk[x] 表示 dfs 序所对应的结点编号，有 rnk[dfn[x]]=x

void dfs2(int o, int t)
{
    top[o] = t;
    cnt++;
    dfn[o] = cnt;
    rnk[cnt] = o;
    if (son[o] == -1)
        return;
    // 优先对重儿子进行 DFS，可以保证同一条重链上的点 DFS 序连续
    dfs2(son[o], t);
    for (int j = h[o]; j; j = nxt[j])
        if (p[j] != son[o] && p[j] != fa[o])
            dfs2(p[j], p[j]);
}
```

重链剖分的性质

- 树上每个节点都属于且仅属于一条重链
- 所有的重链将整棵树 完全剖分
- 一颗子树内的 DFS 序是连续的
- 当我们向下经过一条轻边时，所在子树的大小至少会除以二
- 树上的每条路径都可以被拆分成不超过 $O(\log n)$ 条重链

树剖求 LCA

不断向上跳重链，当跳到同一条重链上时，深度较小的结点即为 LCA。

向上跳重链时需要先跳所在重链顶端深度较大的那个。

由于最多经过 $O(\log n)$ 条重链，因此查询的时间复杂度为 $O(\log n)$

```
int lca(int u, int v)
{
    while (top[u] != top[v])
    {
        if (dep[top[u]] > dep[top[v]])
            u = fa[top[u]];
        else
            v = fa[top[v]];
    }
    return dep[u] > dep[v] ? v : u;
}
```

例题

P2590 [ZJOI2008] 树的统计

对一棵有 n 个节点，节点带权值的静态树，进行三种操作共 q 次：

- 修改单个节点的权值；
- 查询 u 到 v 的路径上的最大权值；
- 查询 u 到 v 的路径上的权值之和。

保证 $1 \leq n \leq 30000$ $0 \leq q \leq 200000$。

例题

P2590 [ZJOI2008] 树的统计

对一棵有 n 个节点，节点带权值的静态树，进行三种操作共 q 次：

- 修改单个节点的权值；
- 查询 u 到 v 的路径上的最大权值；
- 查询 u 到 v 的路径上的权值之和。

保证 $1 \leq n \leq 30000$ $0 \leq q \leq 200000$。

弄一个能实现单点修改，区间查询最大值，区间查询和的线段树。

考虑查询路径。和求 LCA 一个思路，在往上跳的过程中，沿途查询区间信息。单次查询时间复杂度 $O(\log^2 n)$。具体实现看代码

例题

```
// st 是线段树结构体
int querymax(int x, int y)
{
    int ret = -inf, fx = top[x], fy = top[y];
    while (fx != fy)
    {
        if (dep[fx] >= dep[fy])
        {
            ret = max(ret, st.query1(1, 1, n, dfn[fx], dfn[x]));
            x = fa[fx];
        }
        else
        {
            ret = max(ret, st.query1(1, 1, n, dfn[fy], dfn[y]));
            y = fa[fy];
        }
        fx = top[x], fy = top[y];
    }
    if (dfn[x] < dfn[y])
        ret = max(ret, st.query1(1, 1, n, dfn[x], dfn[y]));
    else
        ret = max(ret, st.query1(1, 1, n, dfn[y], dfn[x]));
    return ret;
}
```

树剖习题

- P4211 [LNOI2014]LCA
- CF696E ...Wait for it...
- CF536E Tavas on the Path
- P4180 [BJWC2010] 严格次小生成树
- P5773 [JSOI2016] 轻重路径
- P6074 最小路径

dsu on tree（树上启发式合并）

U41492 树上数颜色

给出一棵 n 个节点以 1 为根的树，节点 u 的颜色为 c_u，现在对于每个结点 u 询问 u 子树里一共出现了多少种不同的颜色。$n \leq 2 \times 10^5$。

dsu on tree（树上启发式合并）

U41492 树上数颜色

给出一棵 n 个节点以 1 为根的树，节点 u 的颜色为 c_u，现在对于每个结点 u 询问 u 子树里一共出现了多少种不同的颜色。$n \leq 2 \times 10^5$。

我们可以先预处理出每个节点子树的大小和它的重儿子，这个过程显然可以 $O(n)$ 完成。我们用 cnt_i 表示颜色 i 的出现次数，ans_u 表示结点 u 的答案。

遍历一个节点 u，我们按以下的步骤进行遍历：

1. 先遍历 u 的轻（非重）儿子，并计算答案，但不保留遍历后它对 cnt 数组的影响；
2. 遍历它的重儿子，保留它对 cnt 数组的影响；
3. 再次遍历 u 的轻儿子的子树结点，加入这些结点的贡献，以得到 u 的答案。

可以证明，这样的时间复杂度为 $O(n \log n)$

实现

先预处理。

```
void dfs0(int u, int p)
{
    L[u] = ++totdfn;
    Node[totdfn] = u;
    sz[u] = 1;
    for (int v : g[u])
        if (v != p)
        {
            dfs0(v, u);
            sz[u] += sz[v];
            if (!big[u] || sz[big[u]] < sz[v])
                big[u] = v;
        }
    R[u] = totdfn;
}
```

实现

```
void add(int u) { if (cnt[col[u]] == 0) ++totColor; cnt[col[u]]++;}
void del(int u) { cnt[col[u]]--; if (cnt[col[u]] == 0) --totColor;}
int getAns() { return totColor; }

void dfs1(int u, int p, bool keep)
{
    for (int v : g[u]) // 计算轻儿子的答案
        if (v != p && v != big[u])
            dfs1(v, u, false);
    if (big[u]) // 计算重儿子答案并保留计算过程中的数据（用于继承）
        dfs1(big[u], u, true);
    for (int v : g[u])
        if (v != p && v != big[u])
            // 子树结点的 DFS 序构成一段连续区间，可以直接遍历
            for (int i = L[v]; i <= R[v]; i++)
                add(Node[i]);
    add(u);
    ans[u] = getAns();
    if (keep == false)
        for (int i = L[u]; i <= R[u]; i++)
            del(Node[i]);
}
```

dsu on tree 习题

- CF741D Arpa's letter-marked tree and Mehrdad's Dokhtar-kosh paths
- CF600E Lomsat gelral
- UOJ284 快乐游戏鸡
- CF1709E XOR Tree