长链剖分

长链剖分本质上就是另外一种链剖分方式。

定义 **重子节点** 表示其子节点中子树深度最大的子结点。如果有多个子树最大的子结点，取其一。如果没有子节点，就无重子节点。

定义 **轻子节点** 表示剩余的子结点。

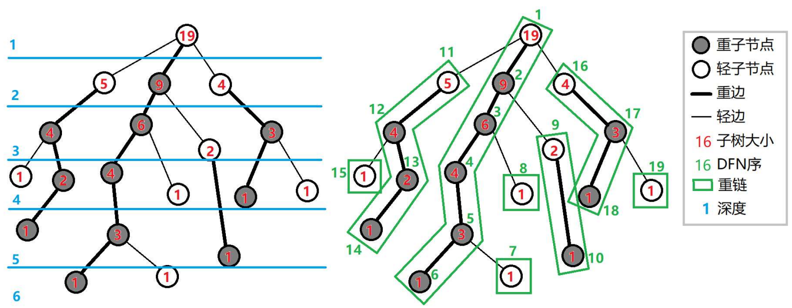
从这个结点到重子节点的边为 **重边**。

到其他轻子节点的边为 **轻边**。

若干条首尾衔接的重边构成 **重链**。

把落单的结点也当作重链，那么整棵树就被剖分成若干条重链。

如图（这种剖分方式既可以看成重链剖分也可以看成长链剖分）：



长链剖分实现方式和重链剖分类似，这里就不再展开。

常见应用

首先，我们发现长链剖分从一个节点到根的路径的轻边切换条数是 $\sqrt{n}$ 级别的。

如何构造数据将轻重边切换次数卡满

我们可以构造这么一颗二叉树 T：

假设构造的二叉树参数为 D。

若 $D \neq 0$, 则在左儿子构造一颗参数为 $D-1$ 的二叉树，在右儿子构造一个长度为 $2D-1$ 的链。

若 $D=0$, 则我们可以直接构造一个单独叶节点，并且结束调用。

这样子构造一定可以将单独叶节点到根的路径全部为轻边且需要 D^2 级别的节点数。

取 $D=\sqrt{n}$ 即可。

长链剖分优化 DP

一般情况下可以使用长链剖分来优化的 DP 会有一维状态为深度维。

我们可以考虑使用长链剖分优化树上 DP。

具体的，我们每个节点的状态直接继承其重儿子的节点状态，同时将轻儿子的 DP 状态暴力合并。

CF 1009F [http://codeforces.com/contest/1009/problem/F]

我们设 $f_{i,j}$ 表示在子树 i 内，和 i 距离为 j 的点数。

直接暴力转移时间复杂度为 $O(n^2)$

我们考虑每次转移我们直接继承重儿子的 DP 数组和答案，并且考虑在此基础上进行更新。

首先我们需要将重儿子的 DP 数组前面插入一个元素 1, 这代表着当前节点。

然后我们将所有轻儿子的 DP 数组暴力和当前节点的 DP 数组合并。

注意到因为轻儿子的 DP 数组长度为轻儿子所在重链长度，而所有重链长度和为 n。

也就是说，我们直接暴力合并轻儿子的总时间复杂度为 $O(n)$。

注意，一般情况下 DP 数组的内存分配为一条重链整体分配内存，链上不同的节点有不同的首位置指针。

DP 数组的长度我们可以根据子树最深节点算出。

例题参考代码：

```cpp
#include <bits/stdc++.h>
using namespace std;
const int N = 1000005;
struct edge {
  int to, next;
} e[N * 2];
int head[N], tot, n;
int d[N], fa[N], mx[N];
int *f[N], g[N], mxp[N];
int dfn[N];
void add(int x, int y) {
  e[++tot] = (edge){y, head[x]};
  head[x] = tot;
}
void dfs1(int x) {
  d[x] = 1;
  for (int i = head[x]; i; i = e[i].next)
    if (e[i].to != fa[x]) {
      fa[e[i].to] = x;
      dfs1(e[i].to);
      d[x] = max(d[x], d[e[i].to] + 1);
      if (d[e[i].to] > d[mx[x]]) mx[x] = e[i].to;
    }
}
void dfs2(int x) {
  dfn[x] = ++*dfn;
  f[x] = g + dfn[x];
  if (mx[x]) dfs2(mx[x]);
  for (int i = head[x]; i; i = e[i].next)
    if (e[i].to != fa[x] && e[i].to != mx[x]) dfs2(e[i].to);
}
void getans(int x) {
  if (mx[x]) {
    getans(mx[x]);
    mxp[x] = mxp[mx[x]] + 1;
  }
  f[x][0] = 1;
  if (f[x][mxp[x]] <= 1) mxp[x] = 0;
  for (int i = head[x]; i; i = e[i].next)
    if (e[i].to != fa[x] && e[i].to != mx[x]) {
      getans(e[i].to);
      int len = d[e[i].to];
      For(j, 0, len - 1) {
        f[x][j + 1] += f[e[i].to][j];
        if (f[x][j + 1] > f[x][mxp[x]]) mxp[x] = j + 1;
        if (f[x][j + 1] == f[x][mxp[x]] && j + 1 < mxp[x])
mxp[x] = j + 1;
      }
    }
}
int main() {
  scanf("%d", &n);
  for (int i = 1; i < n; i++) {
    int x, y;
    scanf("%d%d", &x, &y);
    add(x, y);
    add(y, x);
  }
  dfs1(1);
  dfs2(1);
  getans(1);
  for (int i = 1; i <= n; i++) printf("%d\n", mxp[i]);
}
```


当然长链剖分优化 DP 技巧非常多，包括但是不仅限于打标记等等。这里不再展开。

参考 租酥雨的博客 [https://www.cnblogs.com/zhoushuyu/p/9468669.html]。

长链剖分求 k 级祖先

即询问一个点向父亲跳 k 次跳到的节点。

首先我们假设我们已经预处理了每一个节点的 2^i 级祖先。

现在我们假设我们找到了询问节点的 2^i 级祖先满足 $2^i < k < 2^{i+1}$。

我们考虑求出其所在重链的节点并且按照深度列入表格。假设重链长度为 d。

同时我们在预处理的时候找到每条重链的根节点的 1 到 d 级祖先，同样放入表格。

根据长链剖分的性质，$k - 2^i < 2^i \le d$, 也就是说，我们可以 $O(1)$ 在这条长链的表格上求出的这个节点的 k 级祖先。

预处理需要倍增出 2^i 次级祖先，同时需要预处理每条重链对应的表格。

预处理复杂度 $O(n\log n)$, 询问复杂度 $O(1)$。