HDU 2196.md

Charlemagne

2020.07.07

Contents

;commend line : pandoc “HDU 2196.md” -o “./HDU 2196.pdf” –pdf-engine=xelatex –toc –toc-depth=4 –from markdown –template eisvogel –listings ; in mac if user snippts of markdown don’t work , tap option + Esc

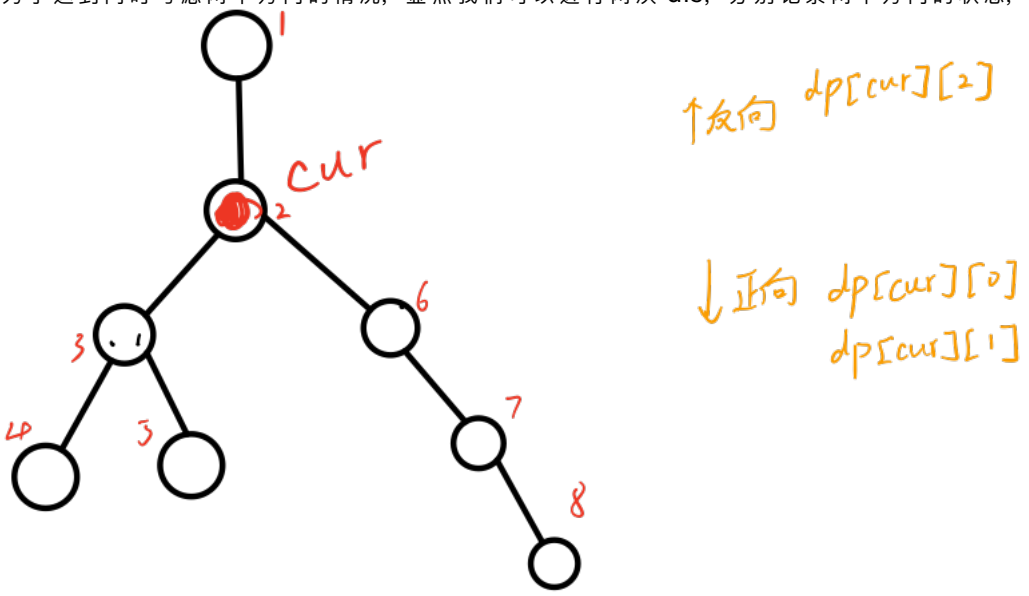
HDU 2196

链接 题意：n 个结点和的带权边组成的树，求树上每一结点到其他结点的距离的最大值。

一个正确思路（树上 dp）

一般的树上 dp 可以处理的就是类似这种树上最值问题，但是一般的树上 dp 往往只是单向处理，即从树根到叶子这个方向去记录状态转移，但是对于这一题，显然我们需要考虑两个方向，一个是以当前结点作为树根的子树方向，另一个就是总的树上不包含以当前结点作为树根的子树方向。用数学语言描述如下：设当前结点为：v_i，对于全结点集合有 $v_i \in V$，假设树为 T，以当前结点为树根的子树 $t_i \subset T$。则在进行状态转移时，我们应当同时考虑 $v_j \in t_i$ 和 $v_j' \in T/t_i$ 分别到结点 i 的最远距离。

为了达到同时考虑两个方向的情况，显然我们可以进行两次 dfs，分别记录两个方向的状态，并进行状态转移。



对于代码中反向结点状态转移这里给出解释：假设当前结点为 cur，考虑计算 6 的反向状态转移，显然结点 6 的状态转移应该由 cur 的反向和 cur 的部分正向（即 cur 子树中不包括结点 6 的那一部分）组成，通常来说，这个状态转移方程如下：

$$dp[6][2] = max(dp[cur][0], dp[cur][2]) + w$$

$dp[cur][0]$ 代表子树到达 cur 的最远距离，然鹅在这个例子我们发现结点 6 恰好就在 cur 取得 $dp[cur][0]$ 的路上，显然不符合结点 6 反向状态转移方程的组成，因此我们需要一个次最大距离，在这里次最大距离指向 3 结点的方向，不在 6 结点方向上，则可以安心状态转移。因此总的反向状态转移方程如下：

$$dp[v][2] = \begin{cases} max(dp[cur][0], dp[cur][2]) + w & id[cur]! = v \\ max(dp[cur][1], dp[cur][2]) + w & id[cur] == v \end{cases}$$

id 数组存储取得最大距离的结点。细节见代码：

```
/*
*   author : charlemagnescl
*   problem name: HDU 2196
*   solveing date : 2020.7.7
*   tag: dp on Tree
*/
#include <iostream>
#include <cstdio>
#include <cstring>
#include <set>
#include <cstdio>
#include <algorithm>
#include <queue>
#include <vector>
using namespace std;
typedef long long LL;
const int N = 1e4+100;
int head[N],tot,id[N];
LL dp[N][3];
struct node
{
    int to,next;
    LL w;
    node(){}
    node(int t,LL l,int n):to(t),w(l),next(n){}
}e[N<<1];
void add(int u,int v,LL w)
{
    e[tot] = node(v,w,head[u]);
    head[u]=tot++;
}
void dfs1(int cur,int pre)  // 记录以cur为根的子树状态（正向）
{
    for(int i=head[cur];i!=-1;i=e[i].next)
    {
        int v = e[i].to;
        LL w=e[i].w;
        if(v!=pre)
        {
            dfs1(v,cur);
```

```
41              LL x = dp[v][0] + w;
42              if(x>dp[cur][0])    // 记录最大距离
43              {
44                  dp[cur][0]=x;
45                  id[cur] = v;    // 记录取得最大距离的路径
46              }
47          }
48      }
49      for(int i=head[cur];i!=-1;i=e[i].next)
50      {
51          int v = e[i].to;
52          LL w=e[i].w;
53          if(v!=pre && v!=id[cur])    // 记录次最大距离
54          {
55              dfs1(v,cur);
56              LL x = dp[v][0] + w;
57              if(x>dp[cur][1])
58              {
59                  dp[cur][1]=x;
60              }
61          }
62      }
63  }
64  void dfs2(int cur, int pre)     // 记录反方向的状态转移
65  {
66
67      for(int i=head[cur]; i!=-1; i=e[i].next)
68      {
69          int v = e[i].to;
70          LL w = e[i].w;
71          if(v!=pre)
72          {
73              if(id[cur] == v)
74
75              {
76                  dp[v][2] = max(dp[cur][1],dp[cur][2]) + w;
77              }
78              else
79              {
80                  dp[v][2] = max(dp[cur][0],dp[cur][2]) + w;
81              }
82              dfs2(v,cur);
83          }
84      }
85  }
86  int main()
87  {
88      int n;
89      while(~scanf("%d",&n))
90      {
91          memset(head,-1,sizeof(head));
```

```
92          memset(dp,0,sizeof(dp));
93          tot=0;
94          for(int i=2;i<=n;i++)
95          {
96              int v;
97              LL w;
98              scanf("%d%lld",&v,&w);
99              add(i,v,w);
100             add(v,i,w);
101         }
102         dfs1(1,0);
103         dfs2(1,0);
104         for(int i=1;i<=n;i++)
105         {
106             printf("%lld\n",max(dp[i][2],dp[i][0]));     // 结果取两个方向的较大值
107         }
108     }
109     return 0;
110 }
```