C：

#include <stdio.h>

#include <string.h>

#include <math.h>

#include <stdlib.h>

struct node;

typedef struct node Node;

typedef Node \*PtrToNode;

typedef PtrToNode List;

typedef PtrToNode Position;

struct node

{

int n;

int val;

Position next;

};

int count=0;

int max=0;

int x;

int \*visit; //是否已遍历

Position Last(List l); //找出最后项

void Insert(int x,int q,List l,Position p); //在p后插入含x的项

void Dfs(int a,List l[]); //深度优先搜索

int Num(List l);

int main(void)

{

int n,u,v,q,a,b;

int i,j,k;

Node \*head;

List \*l,tmp;

Position p;

fscanf(stdin,"%d",&n);

head=(Node \*)malloc(sizeof(Node)\*(n+1));

l=(List \*)malloc(sizeof(List)\*(n+1));

visit=(int \*)malloc(sizeof(int)\*(n+1));

for(i=0;i<=n;i++) //初始化表头及链表

{

head[i].next=NULL;

l[i]=&head[i];

}

for(i=1;i<=n-1;i++) //建立无向图

{

fscanf(stdin,"%d%d%d",&u,&v,&q);

Insert(v,q,l[u],Last(l[u]));

Insert(u,q,l[v],Last(l[v]));

}

for(j=1;j<=n;j++)

visit[j]=0;

Dfs(1,l); //第一次遍历，找到点a，用全局变量x保存

for(j=1;j<=n;j++)

visit[j]=0;

count=0;

max=0;

Dfs(x,l); //第二次遍历，找到点b，用全局变量x保存，此时max为最大距离

printf("%d",max\*10+(max+1)\*max/2);

return 0;

}

Position Last(List l)

{

Position p;

for(p=l;p->next!=NULL;p=p->next);

return p;

}

void Insert(int x,int q,List l,Position p)

{

Position tmp;

tmp=(Position) malloc(sizeof(Node));

tmp->n=x;

tmp->val=q;

tmp->next=p->next;

p->next=tmp;

}

void Dfs(int a,List l[])

{

Position p;

visit[a]=1;

for(p=l[a]->next;p!=NULL;p=p->next)

if(!(visit[p->n]))

{

count+=p->val;

if(count>max)

{

max=count;

x=p->n;

}

Dfs(p->n,l);

count-=p->val;

}

}

int Num(List l)

{

int n=0;

Position p;

for(p=l->next;p!=NULL;p=p->next)

n++;

return n;

}

C++

#include <iostream>

#include <vector>

#include <cstdio>

using namespace std;

struct edge

{

int Number;

int Length;

edge\* next;

edge(int number, int length)

{

Number = number;

Length = length;

next = NULL;

}

};

struct vert

{

int Number;

int PathLen;

bool Isvisit;

edge\* First;

vert(int number)

{

Number = number;

Isvisit = false;

First = NULL;

PathLen = 0;

}

};

void dfs(vector<vert>& GList, int num)

{

edge\* p= GList[num].First;

GList[num].Isvisit = true;

for (; p!=NULL; p=p->next)

{

if(GList[p->Number].Isvisit == false)

{

GList[p->Number].PathLen =GList[num].PathLen + p->Length;

dfs(GList, p->Number);

}

}

}

int main()

{

vector<vert> GList;

int n, i, j;

int fir, sed, len;

cin>>n;

for (i=0; i<=n; i++)

{

GList.push\_back(vert(i));

}

for(j = 1; j < n; j++)

{

edge\* p1, \*p2;

scanf("%d%d%d",&fir,&sed,&len);

p1 = GList[fir].First;

p2 = GList[sed].First;

//无向图

if (p1==NULL)

{

GList[fir].First = new edge(sed, len);

}

else

{

while(p1->next!=NULL) //将边链接到顶点

{

p1 = p1->next;

}

p1->next = new edge(sed, len);

}

if (p2==NULL)

{

GList[sed].First = new edge(fir, len);

}

else

{

while(p2->next!=NULL) //将边链接到顶点

{

p2 = p2->next;

}

p2->next = new edge(fir, len);

}

}

int start, end, length = 0;

dfs(GList, 1);

for (j = 1; j<=n; j++)

{

if (GList[j].PathLen > length)//更新最长距离

{

start = i;

end = j;

length = GList[j].PathLen;

}

GList[j].PathLen = 0;

GList[j].Isvisit = false;

}

dfs(GList, end);

for (j = 1; j<=n; j++)

{

if (GList[j].PathLen > length)//更新最长距离

{

start = i;

end = j;

length = GList[j].PathLen;

}

GList[j].PathLen = 0;

GList[j].Isvisit = false;

}

int cost = length\*10;

for (i = 1; i <= length; i++)

{

cost+=i;

}

cout<<cost<<endl;

return 0;

}

Java：

import java.io.BufferedInputStream;

import java.io.IOException;

import java.util.ArrayList;

public class Main

{

private static BufferedInputStream in = new BufferedInputStream(System.in);

private static ArrayList<Node> n = new ArrayList<Node>();

private static Integer dis = 0;

private static Integer pow = 0;

public static void main(String[] args) throws IOException

{

int size = readInt();

for(int i=0; i<size; i++)

{

n.add(null);

}

for(int i=1; i<size; i++)

{

int x = readInt()-1;

int y = readInt()-1;

int d = readInt();

Node node = new Node();

node.nextSide = n.get(x);

node.con = y;

node.power = d;

n.set(x, node);

node = new Node();

node.nextSide = n.get(y);

node.con = x;

node.power = d;

n.set(y, node);

}

away(0, 0, -1);

pow = 0;

away(dis, 0, -1);

System.out.println(pow \* 10 + (1 + pow) \* pow /2);

}

private static void away(int index, int power, int from)

{

if(pow < power)

{

pow = power;

dis = index;

}

Node node = n.get(index);

while(node != null)

{

if(node.con == from)

{

node = node.nextSide;

continue;

}

away(node.con, power + node.power, index);

node = node.nextSide;

}

}

private static int readInt() throws IOException

{

int i,sum=0;

while(((i=in.read())&48) != 48 || i>57);

for(;(i&56) == 48 || (i&62) == 56; i=in.read())

sum = sum\*10 + (i&15);

return sum;

}

private static class Node

{

int power;

int con;

Node nextSide;

}

}