**金属采集**

问题描述

人类在火星上发现了一种新的金属！这些金属分布在一些奇怪的地方，不妨叫它节点好了。一些节点之间有道路相连，所有的节点和道路形成了一棵树。一共有 n 个节点，这些节点被编号为 1~n 。人类将 k 个机器人送上了火星，目的是采集这些金属。这些机器人都被送到了一个指定的着落点， S 号节点。每个机器人在着落之后，必须沿着道路行走。当机器人到达一个节点时，它会采集这个节点蕴藏的所有金属矿。当机器人完成自己的任务之后，可以从任意一个节点返回地球。当然，回到地球的机器人就无法再到火星去了。我们已经提前测量出了每条道路的信息，包括它的两个端点 x 和 y，以及通过这条道路需要花费的能量 w 。我们想花费尽量少的能量采集所有节点的金属，这个任务就交给你了。

输入格式

第一行包含三个整数 n, S 和 k ，分别代表节点个数、着落点编号，和机器人个数。

接下来一共 n-1 行，每行描述一条道路。一行含有三个整数 x, y 和 w ，代表在 x 号节点和 y 号节点之间有一条道路，通过需要花费 w 个单位的能量。所有道路都可以双向通行。

输出格式

输出一个整数，代表采集所有节点的金属所需要的最少能量。

样例输入

6 1 3  
1 2 1  
2 3 1  
2 4 1000  
2 5 1000  
1 6 1000

样例输出

3004

样例说明

所有机器人在 1 号节点着陆。

第一个机器人的行走路径为 1->6 ，在 6 号节点返回地球，花费能量为1000。

第二个机器人的行走路径为 1->2->3->2->4 ，在 4 号节点返回地球，花费能量为1003。

第一个机器人的行走路径为 1->2->5 ，在 5 号节点返回地球，花费能量为1001。

数据规模与约定

本题有10个测试点。

对于测试点 1~2 ， n <= 10 ， k <= 5 。

对于测试点 3 ， n <= 100000 ， k = 1 。

对于测试点 4 ， n <= 1000 ， k = 2 。

对于测试点 5~6 ， n <= 1000 ， k <= 10 。

对于测试点 7~10 ， n <= 100000 ， k <= 10 。

道路的能量 w 均为不超过 1000 的正整数。

锦囊1

使用动态规划。

锦囊2

使用F[i][k]表示以i结点为根，有k条路还要继续向根的方向走时最少花费的能量。

首先计算所有i的子结点的F值，然后再使用动态规划来计算F[i][k]。此时令G[p][j]表示i的前p个子树，还有j个要向上走时最小花费的能量，可写出G[p]和G[p-1]以及对应的结点F值的关系。利用两重动态规划解决本题。

C++代码：

#include <iostream>

#include <cstdio>

using namespace std;

const int MAXN=100000+10,oo=100000000,MAXK=10+1;

typedef long long LL;

int N,S,K,fa[MAXN];

int g[MAXN],num[MAXN\*2],next[MAXN\*2],cost[MAXN\*2],tot=1;

LL f[MAXN][MAXK],sum;

inline void read(int &x)

{

char ch;

while (ch=getchar(),ch>'9' || ch<'0') ; x=ch-48;

while (ch=getchar(),ch<='9' && ch>='0') x=x\*10+ch-48;

}

inline void addedge(int a,int b,int c) { ++tot; num[tot]=b; next[tot]=g[a]; g[a]=tot; cost[tot]=c; }

void dfs(int x)

{

for (int i=g[x];i;i=next[i])

if (num[i]!=fa[x])

{

fa[num[i]]=x;

dfs(num[i]);

for (int a=K;a;--a)

for (int b=1;b<=a;++b)

f[x][a]=max(f[x][a],f[x][a-b]+f[num[i]][b]+(LL)(-b+2)\*cost[i]);

}

}

int main()

{

read(N); read(S); read(K);

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

{

int x,y,z;

read(x); read(y); read(z); sum+=z;

addedge(x,y,z); addedge(y,x,z);

}

sum=sum+sum;

dfs(S);

LL ans=oo; ans=ans\*ans;

for (int i=0;i<=K;++i) ans=min(ans,sum-f[S][i]);

cout << ans << endl;

return 0;

}

C代码：

#include<stdio.h>

#include<string.h>

#include<stdlib.h>

typedef struct node

{

int p;

int cost;

struct node \* next;

}node;

node \* table[100005];

int dp[100005][12];

int n, s, k;

int visit[100005];

void dfs(int pos);

int main()

{

int i;

int x, y, w;

node \* tmp;

scanf("%d %d %d", &n, &s, &k);

memset(table, 0, sizeof(table));

memset(dp, 0, sizeof(dp));

memset(visit, 0, sizeof(visit));

for(i = 0; i < n-1; i++)

{

scanf("%d %d %d", &x, &y, &w);

tmp = (node \* )malloc(sizeof(node));

tmp->p = y;

tmp->cost = w;

tmp->next = table[x];

table[x] = tmp;

tmp = (node \* )malloc(sizeof(node));

tmp->p = x;

tmp->cost = w;

tmp->next = table[y];

table[y] = tmp;

}

dfs(s);

printf("%d", dp[s][k]);

return 0;

}

void dfs(int pos)

{

int p;

node \* tmp;

int i, j;

int cost;

visit[pos] = 1;

for(tmp = table[pos]; tmp; tmp = tmp->next)

{

p = tmp->p;

if(visit[p])

continue;

cost = tmp->cost;

dfs(p);

for(i = k; i >= 0; i--)

{

dp[pos][i] = dp[pos][i-0] + dp[p][0] + 2 \* cost;//如果子节点不留，则说明全部返回了上层节点，那此时最有的结果就是，从上层只有一个往下走

for(j = i; j > 0 ; j--)

{

dp[pos][i] = (dp[pos][i] < (dp[pos][i-j] + dp[p][j] + cost\*j))? dp[pos][i] : (dp[pos][i-j] + dp[p][j] + cost\*j);

}

}

}

}

JAVA代码：

import java.io.IOException;

import java.io.InputStream;

import java.util.Arrays;

import java.util.Stack;

class canner {

private InputStream is = System.in;

public int nextInt() {

try {

int i;

while ((i = is.read()) < 45 || i > 57) {

}

int mark = 1, temp = 0;

if (i == 45) {

mark = -1;

i = is.read();

}

while (i > 47 && i < 58) {

temp = temp \* 10 + i - 48;

i = is.read();

}

return temp \* mark;

} catch (IOException e) {

e.printStackTrace();

}

return -1;

}

}

public class Main {

static class Edge {

public int to;

public int next;

public int w;

}

private static int head[];

private static Edge[] edge;

private static int top;

private static int dp[][];

static int k;

static boolean[] flag;

public static void main(String[] args) {

canner sc = new canner();

int n = sc.nextInt();

int s = sc.nextInt();

k = sc.nextInt();

init(n);

dp = new int[n + 1][k + 1];

for (int i = 1; i < n; i++) {

int a = sc.nextInt();

int b = sc.nextInt();

int c = sc.nextInt();

addEdge(a, b, c);

addEdge(b, a, c);

}

dfs(s);

System.out.println(dp[s][k]);

}

private static void dfs(int root) {

Stack<Integer> stack = new Stack<Integer>();

int temp = root;

stack.push(root);

flag[root] = true;

while (!stack.isEmpty()) {

root = stack.peek();

int num = find(root);

if (num == -1) {

stack.pop();

if (!stack.isEmpty()) {

int peek = stack.peek();

int to = root;

int w = find2(root, peek);

for (int j = k; j >= 0; j--) {

dp[peek][j] += dp[to][0] + 2 \* w;

for (int l = 1; l <= j; l++) {

dp[peek][j] = Math.min(dp[peek][j], dp[peek][j - l]

+ dp[to][l] + l \* w);

}

}

}

} else {

flag[num] = true;

stack.push(num);

}

}

}

private static int find2(int root, int peek) {

for (int i = head[peek]; i != -1; i = edge[i].next) {

if (edge[i].to == root)

return edge[i].w;

}

return peek;

}

private static int find(int root) {

int num = -1;

for (int i = head[root]; i != -1; i = edge[i].next) {

if (flag[edge[i].to])

continue;

num = edge[i].to;

break;

}

return num;

}

private static void addEdge(int to, int next, int w) {

edge[top] = new Edge();

edge[top].to = next;

edge[top].w = w;

edge[top].next = head[to];

head[to] = top++;

}

private static void init(int n) {

flag = new boolean[n + 1];

int size = (n - 1) \* 2;

edge = new Edge[size];

head = new int[size];

top = 0;

Arrays.fill(head, -1);

}

}