**算法训练 结点选择**

时间限制：1.0s   内存限制：256.0MB

问题描述

有一棵 n 个节点的树，树上每个节点都有一个正整数权值。如果一个点被选择了，那么在树上和它相邻的点都不能被选择。求选出的点的权值和最大是多少？

输入格式

第一行包含一个整数 n 。

接下来的一行包含 n 个正整数，第 i 个正整数代表点 i 的权值。

接下来一共 n-1 行，每行描述树上的一条边。

输出格式

输出一个整数，代表选出的点的权值和的最大值。

样例输入

5  
1 2 3 4 5  
1 2  
1 3  
2 4  
2 5

样例输出

12

样例说明

选择3、4、5号点，权值和为 3+4+5 = 12 。

数据规模与约定

对于20%的数据， n <= 20。

对于50%的数据， n <= 1000。

对于100%的数据， n <= 100000。

权值均为不超过1000的正整数。

锦囊1

使用树型动态规划。

锦囊2

用F[i]表示从子树i中选择结点，且结点i必须被选择的最大值，用G[i]表示从子树i中选择结点，且结点i必须不被选择的最大值。

则F[i]=a[i]+\sum(G[j])，其中a[i]表示结点i的权值，j是i的子结点。

G[i]=\sum(max(F[j], G[j]))，其中j是i的子结点。

本题的C++参考代码如下：

#include<stdio.h>

const int NO=1000005;

int dp[NO][2];

int du[NO];

int first[NO],next[NO],v[NO],num=1;

bool mark[NO];

int n,a;

int t[NO],tip,top;

int max(int a,int &b){return a>b?a:b;}

void input(int &num)

{

num=0;

char ch=getchar();

while(ch<'0'||'9'<ch)

ch=getchar();

while('0'<=ch&&ch<='9')

{

num=10\*num+ch-'0';

ch=getchar();

}

}

void add(int &a,int &b)

{

v[num]=b;

next[num]=first[a];

first[a]=num++;

v[num]=a;

next[num]=first[b];

first[b]=num++;

}

int work()

{

int i;

while(tip<top)

{

a=t[tip++];

mark[a]=1;

for(i=first[a];i!=-1;i=next[i])

if(mark[v[i]])

{

dp[a][0]+=max(dp[v[i]][0],dp[v[i]][1]);

dp[a][1]+=dp[v[i]][0];

}

else if(--du[v[i]]==1)

t[top++]=v[i];

}

return max(dp[a][0],dp[a][1]);

}

int main()

{

int i,a,b;

scanf("%d",&n);

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

input(dp[i][1]),first[i]=-1;

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

input(a),input(b),add(a,b),du[a]++,du[b]++;

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

if(du[i]==1)

t[top++]=i;

printf("%d\n",work());

return 0;

}

本题的C参考代码如下：

#include <stdio.h>

#include <string.h>

#define \_Max 100010

#define max(a, b) a > b ? a : b

struct point

{

int v, next; //v指向这条边的另一个结点（父结点），next指向子结点

} edge[\_Max \* 2]; //一条边记录两次，分别以一个点做记录

int head[\_Max];

int M;

int dp[\_Max][2];

//添加一个边

void addEdge(int from, int to)

{

//from结点

edge[M].v = to;

edge[M].next = head[from]; //为-1则定位叶结点，否则，指向另外一条边

head[from] = M++; //指向他的一条边，增加结点

//to结点

edge[M].v = from;

edge[M].next = head[to]; //为-1则定位叶结点，否则，指向另外一条边

head[to] = M++; //指向他的一条边，增加结点

return ;

}

//深度遍历，先深入到叶子结点，然后一层一层往上回升，一直到根结点，即第一个结点（初始pre为－1是因为根结点没有父结点，用－1表示）

void dfs(int x, int pre)

{

int i = head[x], v;

for (; i != -1; i = edge[i].next) //i != -1说明有子结点，则遍历子结点，否则为叶子结点

{

v = edge[i].v;

if (pre == v) //如果指向的子结点和父结点重合，则说明这个结点是叶子结点，不需要进一步dp

{

continue;

}

dfs(v, x); //x可以理解为父结点

//深度遍历到最里面的叶子结点的父结点 如果父结点选择，则子结点不选择，否则子结点可能选择或者不选择，但是要比较两者哪个大选择哪个

dp[x][1] += dp[v][0]; // 父结点（选） ＋＝ 子结点（不选）

dp[x][0] += max(dp[v][0], dp[v][1]); // 父结点（不选） ＋＝ max（子结点（不选），子结点（选））

}

return ;

}

int main(int argc, const char \* argv[])

{

int i, n, s, t, tmp;

scanf("%d", &n);

M = 0;

memset(head, -1, sizeof(head)); //初始化每个结点都是独立的没有子结点

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

//输入权值，并且记录在dp[i][1]上，i表示第i个结点，1代表取了这个结点

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

{

scanf("%d", &dp[i][1]);

}

//输入边，并且添加edge，一个边添加两个edge

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

{

scanf("%d %d", &s, &t);

addEdge(s, t);

}

dfs(1, -1); //深度优先遍历，从第一个结点开始遍历

tmp = max(dp[1][0], dp[1][1]); //求出最大的权值和

printf("%d\n", tmp);

return 0;

}

本题的Java参考代码如下：

//package practice;

import java.io.\*;

import java.util.\*;

public class Main {

final static int MAX\_N = 100010;

//final static int MAX\_M = 200007;

final static long INF = (long)1e16;

class Edge {

int u, v, nxt;

Edge () {

}

Edge (int \_u, int \_v, int \_n) {

u = \_u;

v = \_v;

nxt = \_n;

}

}

int edgecnt;

int dp[][] = new int[MAX\_N][2];

Edge E[] = new Edge[MAX\_N \* 2];

int head[] = new int[MAX\_N];

int sta[] = new int[MAX\_N \* 2];

boolean vis[] = new boolean[MAX\_N];

void add(int u, int v) {

E[edgecnt] = new Edge(u, v, head[u]);

head[u] = edgecnt++;

}

void dfs(int x, int fa) {

Arrays.fill(vis, false);

int top = 0;

vis[x] = true;

sta[top++] = x;

while (top > 0) {

int u = sta[top - 1];

boolean Ed = false;

for (int i = head[u]; i + 1 != 0; i = E[i].nxt) {

int v = E[i].v;

if (vis[v]) continue;

Ed = true;

sta[top++] = v;

vis[v] = true;

}

if (Ed) continue;

--top;

for (int i = head[u]; i + 1 != 0; i = E[i].nxt) {

int v = E[i].v;

dp[v][0] += Math.max(dp[u][0], dp[u][1]);

dp[v][1] += dp[u][0];

}

}

}

void run() throws IOException {

int n = cin.nextInt();

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

dp[i][1] = cin.nextInt();

Arrays.fill(head, -1);

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

int u = cin.nextInt();

int v = cin.nextInt();

add(u, v);

add(v, u);

}

dfs(1, -1);

int ans = Math.max(dp[1][0], dp[1][1]);

out.println(ans);

out.close();

}

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

new Main().run();

}

Main() {

cin = new InputReader(System.in);

// cin = new Scanner(System.in);

out = new PrintWriter(System.out);

}

PrintWriter out;

InputReader cin;

//Scanner cin;

class InputReader {

InputReader(InputStream in) {

reader = new BufferedReader(new InputStreamReader(in));

// try {

// reader = new BufferedReader(new FileReader("input.txt"));

// } catch (FileNotFoundException ex) {

// }

tokenizer = new StringTokenizer("");

}

private String next() throws IOException {

while (!tokenizer.hasMoreTokens()) {

tokenizer = new StringTokenizer(reader.readLine());

}

return tokenizer.nextToken();

}

public Integer nextInt() throws IOException {

return Integer.parseInt(next());

}

private BufferedReader reader;

private StringTokenizer tokenizer;

}

}