**算法训练3-5**

**3算法训练 K好数**

问题描述

如果一个自然数N的K进制表示中任意的相邻的两位都不是相邻的数字，那么我们就说这个数是K好数。求L位K进制数中K好数的数目。例如K = 4，L = 2的时候，所有K好数为11、13、20、22、30、31、33 共7个。由于这个数目很大，请你输出它对1000000007取模后的值。

输入格式

输入包含两个正整数，K和L。

输出格式

输出一个整数，表示答案对1000000007取模后的值。

样例输入

4 2

样例输出

7

数据规模与约定

对于30%的数据，KL <= 106；

对于50%的数据，K <= 16， L <= 10；

对于100%的数据，1 <= K,L <= 100。

**题目分析:**

1**理解”L位K进制数中的K好数 的数目”**

K进制数，取值范围为**[0，K-1]**； “L位K进制数”，如：**“2位4进制数”、 “3位7进制数”；**要求是**任意相邻的两位不是相邻的数字**。

**2 “由于数很大……输出取模后的值”这一条件？**

在数据规范与约定中，我们知道1\leqslantK\leqslant100，1\leqslant L\leqslant​​​​​​​100。假设K=100，L=100，即100位100进制数中100好数的情况下，它的数目约就有100100，肯定超过了int，long long。所以通过取模来减小运算量。

**3 举例分析 “2位4进制数”和“3为4进制数”**

满足条件的2位4进制数：11、13 、20 、22 、 30、 31、 33，共7个。 满足条件的3位4进制数：111、113、130、131 、133、200、 202、 203、220、 222、300、 302、 303、311、 313、330、 331、 333，共18个。

**解题思路1:**

1 使用**动态规划**，通过组合子问题的解来求解原问题。

2 **用F[i][j]表示长为i，最后一位数字是j的K好数的个数，则F[i][j]= F[i][j]+sum F[i-1][k]，其中|j-k|!=1。**

当前位置的数总数=当前位置的数的数目+前一个位置的数的总数。

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **i j** | **0** | **1** | **2** | **3** | **…** | **100** |
| **0** | **~~1~~ 0** | **1** | **1** | **1** |  |  |
| **1** | **2** | **2** | **1** | **2** |  |  |
| **2** | **5** | **4** | **3** | **6** |  |  |
| **…** |  |  |  |  |  |  |
| **100** |  |  |  |  |  |  |

**解题思路2:**

在F[i][j]中，**i 代表所在的位数**（i=0代表个位，i=1代表十位，以此类推）。**j 代表取K进制数的一个数**(如果是4进制数，那么j取[0,3]中的一个数)。F[i][j]表示当前位置的数K好数的个数，则F[i][j]= F[i][j]+sum F[i-1][k]，其中|j-k|!=1。

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **i j** | **0** | **1** | **2** | **3** | **…** | **100** |
| **0** | **1** | **1** | **1** | **1** |  |  |
| **1** | **3** | **2** | **2** | **3** |  |  |
| **2** | **8** | **5** | **5** | **8** |  |  |
| **…** |  |  |  |  |  |  |
| **100** |  |  |  |  |  |  |

**参考代码:**

#include <iostream>

using namespace std;

int main(int argc, char\*\* argv) {

int k,l;

cin>>k>>l;

long long table[100][100];

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

{ table[0][i]=1ll;

}

table[0][0]=0ll;

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

{ for(int j=0;j<k;j++)

{

long long x=0;

for(int y=0;y<k;y++)

{

if(y!=j+1 && y!=j-1)

{

x+=table[i-1][y];

}

}

table[i][j]=x%1000000007ll;

}

}

long long count=0;

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

{

count+=table[l-1][i];

}

cout<<(count%1000000007ll);

return 0;

}

**4算法训练 结点选择**

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  
 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。

**题目分析与解题思路:**

这是一道和树有关的问题，需要考虑如何去遍历树？（用到DFS），然后还需要用到动态规划，由此，我们做这道题，解题方案也就是**树形动态规划**了！

Step1:首先构造一棵树。（边与点配合，一个点的两条边都要标记）

Step2：动态规划的变化。

动态规划----构造状态转移方程

**对于叶子结点：**

dp[k][0] = 0;

dp[k][1] = k点权值;

**对于非叶子结点：**

dp[i][0] = max(dp[j][0], dp[j][1]); (j是i的儿子)

dp[i][1] = i点权值 + dp[j][0]; (j是i的儿子)

**最大权值即为：**

max(dp[0][0], dp[0][1])。（要么不包括根结点，要么包括根结点）

**参考代码1:**

#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;

}

**解题思路2：**

1 : 使用树型动态规划。

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。

**参考代码2:**

#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;

}

**5算法训练 最短路**

问题描述

给定一个n个顶点，m条边的有向图（其中某些边权可能为负，但保证没有负环）。请你计算从1号点到其他点的最短路（顶点从1到n编号）。

输入格式

第一行两个整数n, m。

接下来的m行，每行有三个整数u, v, l，表示u到v有一条长度为l的边。

输出格式

共n-1行，第i行表示1号点到i+1号点的最短路。

样例输入

3 3  
1 2 -1  
2 3 -1  
3 1 2

样例输出

-1  
-2

数据规模与约定

对于10%的数据，n = 2，m = 2。

对于30%的数据，n <= 5，m <= 10。

对于100%的数据，1 <= n <= 20000，1 <= m <= 200000，-10000 <= l <= 10000，保证从任意顶点都能到达其他所有顶点。

锦囊1: 使用最短路算法。

锦囊2：使用Dijkstra算法，此图的边数比点数的平方要少很多，因此应该使用带堆优化的Dijkstra。

参考代码：

#include<iostream>

#include<cstdio>

#include<cmath>

#include<algorithm>

#include<queue>

#include<cstring>

#define ENT putchar('\n')

using namespace std;

const int maxn=100000+10,INF=-1u>>1;

struct Tedge{int x,y,w,next;}adj[maxn\*2];int fch[maxn],ms=0;

void AddEdge(int u,int v,int w){

adj[++ms]=(Tedge){u,v,w,fch[u]};fch[u]=ms;return;

}

bool vis[maxn];int dist[maxn],n,m;

void SPFA(){

queue<int>Q;memset(vis,false,sizeof(vis));

for(int i=1;i<=n+1;i++)dist[i]=INF;dist[1]=0;

Q.push(1);vis[1]=true;

while(!Q.empty()){

int u=Q.front();Q.pop();vis[u]=false;

for(int i=fch[u];i;i=adj[i].next){

int v=adj[i].y;

if(dist[v]>dist[u]+adj[i].w){

dist[v]=dist[u]+adj[i].w;

if(!vis[v]) vis[v]=true,Q.push(v);

}

}

} return ;

}

inline int read(){

int x=0,sig=1;char ch=getchar();

while(!isdigit(ch)){if(ch=='-') sig=-1;ch=getchar();}

while(isdigit(ch)) x=10\*x+ch-'0',ch=getchar();

return x\*=sig;

}

inline void write(int x){

if(x==0){putchar('0');return;}if(x<0) putchar('-'),x=-x;

int len=0,buf[15];while(x) buf[len++]=x%10,x/=10;

for(int i=len-1;i>=0;i--) putchar(buf[i]+'0');return;

}

void init(){

n=read();m=read();int a,b,c;

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

a=read();b=read();c=read();

AddEdge(a,b,c);

} return;

}

void work(){

SPFA();

return;

}

void print(){

for(int i=2;i<=n;i++) write(dist[i]),ENT;

return;

}

int main(){

init();work();print();return 0;

}