2016级数据结构第十次上机解题报告

A

根据二叉树的前序遍历序列求出树的高度。

使用递归求解此题，假设当前执行到了节点p：

如果p节点为空，那么以p为根节点的子树高度为0

如果p节点不为空，p的下一个节点为p的左儿子，递归求出左子树高度，左子树后的第一个节点为p的右儿子，再递归求出右子树高度，取较大值再加1即为以p为根节点的子树的高度。

参考代码：

#include <iostream>

#include <string>

using namespace std;

string s;

int n, p;

int find() {

if (p >= s.length() || s[p++] == '#') return 0;

return max(find(), find()) + 1;

}

int main() {

cin >> n;

while (n--) {

cin >> s;

p = 0;

cout << find() << "\n";

}

}

B

模拟，刚开始位置为1，每次更新位置，超过t就不用管了，能到达t就是YES。

参考代码：

#include <cstdio>

int a[30001];

int main()

{

int n,t,now;

while(scanf("%d%d",&n,&t)!=EOF)

{

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

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

now=1;

while(now<t)

now+=a[now];

if(now==t)

printf("YES\n");

else printf("NO\n");

}

return 0;

}

C

图的深度优先遍历，因为一开始不知道所有点的值，所以在输入边的时候要把点保存记录下来。为了防止每次输入边都要判断点是否存在所以hash一下点的值即可，然后把点的值排序，进行DFS即可。

参考代码：

#include <cstdio>

#include <cstring>

#include <algorithm>

using namespace std;

int z[5001][5001], x[5001],judge[5001],p[5001];

void DFS(int a, int b)

{

x[a] = 1;

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

{

if (z[a][p[i]] && !x[p[i]])

{

printf(" %d", p[i]);

DFS(p[i], b);

}

}

}

int main()

{

int k, m, c, d, cnt;

while (scanf("%d%d",&k,&m)!=EOF)

{

cnt = 0;

memset(judge, 0, sizeof(judge));

memset(z, 0, sizeof(z));

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

{

scanf("%d%d", &c, &d);

z[c][d] = 1;

if (judge[c] == false)

{

p[cnt++] = c;

judge[c] = true;

}

if (judge[d] == false)

{

p[cnt++] = d;

judge[d] = true;

}

}

sort(p, p + cnt);

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

{

memset(x, 0, sizeof(x));

printf("%d", p[i]);

DFS(p[i], cnt);

printf("\n");

}

}

return 0;

}

D

剪枝bfs，如果操作后的数大于10^4这个上界，或者小于1这个下界就可以减掉，如果这个数之前出现过就可以减掉，也可以用数学方法a掉。

参考代码：

#include <cstdio>

#include <queue>

using namespace std;

queue<pair<int,int> > q;

bool used[10001];

int main()

{

//freopen("in.in","r",stdin);

//freopen("out.out","w",stdout);

int n,m,ans=0;

while(scanf("%d%d",&n,&m)!=EOF)

{

q.push(make\_pair(n,0));

while(!q.empty())

{

used[q.front().first]=true;

if(q.front().first==m)

{

ans=q.front().second;

break;

}

if(q.front().first\*2>=1&&q.front().first\*2<=10000&&!used[q.front().first\*2])

q.push(make\_pair(q.front().first\*2,q.front().second+1));

if(q.front().first-1>=1&&q.front().first-1<=10000&&!used[q.front().first-1])

q.push(make\_pair(q.front().first-1,q.front().second+1));

q.pop();

}

for(int i=1;i<=10000;i++)

used[i]=false;

while(!q.empty())

q.pop();

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

ans=0;

}

return 0;

}

E

这道题与求树的叶节点个数的思路有点相似，不过我们不止要求叶节点的个数，同时还要记录下每个叶节点的深度，然后求平均值即可。

参考代码：

#include <iostream>

#include <cstdio>

using namespace std;

struct Tree {

int val;

Tree \*left, \*right;

Tree() : val(0), left(nullptr), right(nullptr) {}

Tree(int x) : val(x), left(nullptr), right(nullptr) {}

};

void deleteTree(Tree \*tree) {

if (tree) {

deleteTree(tree->left);

deleteTree(tree->right);

delete tree;

}

}

int countLeaf(Tree \*tree, int depth, int& leaf) {

if (!tree) return 0;

// 如果是叶节点，便记录深度

if (!tree->left and !tree->right) {

leaf++;

return depth;

}

return countLeaf(tree->left, depth + 1, leaf) + countLeaf(tree->right, depth + 1, leaf);

}

char a[101];

void createTree(Tree \*&t, int& i) {

char p = a[i++];

if (p == '\0') {

return;

}

if (p != '#') {

t = new Tree(p);

createTree(t->left, i);

createTree(t->right, i);

}

}

int main() {

while (~scanf("%s", a)) {

int i = 0, leaf = 0, height = 0;

Tree \*root = nullptr;

createTree(root, i);

height = countLeaf(root, 1, leaf);

if (!leaf) printf("0.00\n");

else printf("%.2f\n", height \* 1.0 / leaf);

deleteTree(root);

}

return 0;

}

F

这道题与上一题思路一样，只不过是把叶节点改成了所有节点。

参考代码：

#include <iostream>

#include <cstdio>

using namespace std;

struct Tree {

int val;

Tree \*left, \*right;

Tree() : val(0), left(nullptr), right(nullptr) {}

Tree(int x) : val(x), left(nullptr), right(nullptr) {}

};

void deleteTree(Tree \*tree) {

if (tree) {

deleteTree(tree->left);

deleteTree(tree->right);

delete tree;

}

}

int countNode(Tree\* tree, int depth, int& node) {

if (!tree) return 0;

node++;

return countNode(tree->left, depth + 1, node) + countNode(tree->right, depth + 1, node) + depth;

}

char a[101];

void createTree(Tree \*&t, int& i) {

char p = a[i++];

if (p == '\0') {

return;

}

if (p != '#') {

t = new Tree(p);

createTree(t->left, i);

createTree(t->right, i);

}

}

int main() {

while (~scanf("%s", a)) {

int i = 0, node = 0, height = 0;

Tree \*root = nullptr;

createTree(root, i);

height = countNode(root, 1, node);

if (!node) printf("0.00\n");

else printf("%.2f\n", height \* 1.0 / node);

deleteTree(root);

}

return 0;

}

G

并查集，如果两个雪堆有一维坐标相同那就把他们所在的集合合并，最后的答案就是集合数减1，也可以暴力搜索过掉。

参考代码：

#include <cstdio>

int father[101];

int x[101],y[101];

int findfather(int x)

{

if(father[x]==x)

return x;

else return father[x]=findfather(father[x]);

}

int main()

{

int n,ans;

while(scanf("%d",&n)!=EOF)

{

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

scanf("%d%d",&x[i],&y[i]);

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

father[i]=i;

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

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

if(x[i]==x[j]||y[i]==y[j])

father[findfather(i)]=findfather(j);

ans=0;

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

if(father[i]==i)

ans++;

printf("%d\n",ans-1);

}

return 0;

}