2016级数据结构第七次上机解题报告

A

简单题，对于给出的数据依次检索即可。

参考代码：  
#include <iostream>

using namespace std;

int main() {

int N, H, D;

cin >> N >> D >> H;

int h, d, wasted = 0;

while (N-- && H > 0) {

cin >> d >> h;

if (d == D)

H -= h;

else

wasted++;

}

cout << wasted << "\n";

return 0;

}

B

使用一个队列记录“当前能看到哪些山峰”，枚举每一个山峰的状态，每次更新这个队列即可。更新的规则很简单，队列中的有元素必须依次递减。每次更新时，将当前元素入队，不符合条件的元素出队。

参考代码：

#include <iostream>

#include <stack>

using namespace std;

int main() {

int n, now;

long long cnt = 0;

stack<int> s;

while (!s.empty()) s.pop();

cin >> n;

while (n--) {

cin >> now;

if (s.empty())

s.push(now);

else {

cnt += s.size();

while (!s.empty() && now > s.top())

s.pop();

s.push(now);

}

}

cout << cnt;

return 0;

}

C

这道题的要求是根据完全二叉树的层次遍历来建立二叉树，并输出二叉树的中序遍历。我们在层次遍历输出一棵树的时候用到了队列，所以在建立一棵树的时候可以用同样的方法建立。

参考代码：  
#include <iostream>

#include <queue>

#include <fstream>

#define in std::cin

#define out std::cout

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 in\_order(Tree \*root) {

if (!root) return;

in\_order(root->left);

out << root->val << ' ';

in\_order(root->right);

}

int main() {

int x;

queue<Tree\*> q;

bool has\_left = false;

in >> x;

Tree \*root = new Tree(x);

q.push(root);

while (in >> x) {

Tree \*tree = q.front();

Tree \*temp = new Tree(x);

if (!has\_left) {

tree->left = temp;

q.push(temp);

has\_left = true;

} else {

tree->right = temp;

q.push(temp);

q.pop();

has\_left = false;

}

}

in\_order(root);

out << '\n';

return 0;

}

D

这道题要求是求出叶子节点的数目，我们可以遍历这棵树，如果在遍历的时候，某个节点非空，而且左右子树都是空的，那就说明这是一个叶子节点。

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 preOrder(Tree \*tree) {

if (tree) {

printf("%c", tree->val);

preOrder(tree->left);

preOrder(tree->right);

}

}

void deleteTree(Tree \*tree) {

if (tree) {

deleteTree(tree->left);

deleteTree(tree->right);

delete tree;

}

}

int countLeaf(Tree \*tree) {

if (!tree) return 0;

if (!tree->left and !tree->right) return 1;

return countLeaf(tree->left) + countLeaf(tree->right);

}

char a[1010];

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;

Tree \*root = nullptr;

createTree(root, i);

printf("%d\n", countLeaf(root));

deleteTree(root);

}

return 0;

}

E

输入时注意一下格式即可，要按行读入然后进行拆分，gets，getline皆可。

之后便是二叉树的四种遍历，书上原题，在此不进行赘述。

参考代码：

#include <iostream>

#include <stack>

#include <cstring>

#include <cstdio>

#include <string>

using namespace std;

struct BSTree

{

int data;

BSTree \*lchild, \*rchild;

BSTree() :data(0), lchild(NULL), rchild(NULL) {}

};

void InsertBST(BSTree \*&t, int key)

{

if (t == NULL)

{

t = new BSTree;

t->lchild = t->rchild = NULL;

t->data = key;

}

else

{

if (key<t->data)

InsertBST(t->lchild, key);

else

InsertBST(t->rchild, key);

}

}

void Print1(BSTree \*t)

{

if (t)

{

printf("%d ", t->data);

Print1(t->lchild);

Print1(t->rchild);

}

}

void Print2(BSTree \*t)

{

if (t)

{

Print2(t->lchild);

printf("%d ", t->data);

Print2(t->rchild);

}

}

void Print3(BSTree \*t)

{

if (t)

{

Print3(t->lchild);

Print3(t->rchild);

printf("%d ", t->data);

}

}

void Print4(BSTree \*T)

{

deque<BSTree\*> q\_first, q\_second;

q\_first.push\_back(T);

while (!q\_first.empty())

{

while (!q\_first.empty())

{

BSTree \*temp = q\_first.front();

q\_first.pop\_front();

cout << temp->data << " ";

if (temp->lchild)

q\_second.push\_back(temp->lchild);

if (temp->rchild)

q\_second.push\_back(temp->rchild);

}

q\_first.swap(q\_second);

}

}

void del(BSTree \*T)

{

if(T->lchild)

del(T->lchild);

if(T->rchild)

del(T->rchild);

delete T;

}

int main()

{

char s[60010];

int a[60010];

int cnt, cnt2;

int tt;

while (cin.getline(s,50010,'\n'))

{

cnt = 0;

cnt2 = 0;

BSTree \*t;

t = NULL;

tt=0;

while (s[cnt] != '\0')

{

if (s[cnt] != ' ')

{

tt\*=10;

tt+=s[cnt]-'0';

}

else

{

a[cnt2++]=tt;

tt=0;

}

cnt++;

}

a[cnt2++]=tt;

for(int i=0;i<cnt2;i++)

InsertBST(t, a[i]);

Print1(t);

cout << endl;

Print2(t);

cout << endl;

Print3(t);

cout << endl;

Print4(t);

cout << endl;

del(t);

memset(s, 0, sizeof(s));

}

return 0;

}

F

因为数据范围较大，所以要用map做个离散化，map[i]表示从i到i/2的路的花费，求最短路就是不断取v和u之中较大的除以2，直到两数相等就是它们的LCA，增加路段收费和求最短路收费都可以这么操作，前者是将前者的map值增加，后者是统计map值之和

参考代码：

#include <cstdio>

#include <map>

using namespace std;

map<long long,long long> mp;

int main()

{

int q,type;

long long v,u,w,ans;

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

{

while(q--)

{

scanf("%d",&type);

if(type==1)

{

scanf("%lld%lld%lld",&v,&u,&w);

while(v!=u)

{

if(v>u)

{

mp[v]+=w;

v/=2;

}

else

{

mp[u]+=w;

u/=2;

}

}

}

else

{

scanf("%lld%lld",&v,&u);

ans=0;

while(v!=u)

{

if(v>u)

{

ans+=mp[v];

v/=2;

}

else

{

ans+=mp[u];

u/=2;

}

}

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

}

}

mp.clear();

}

return 0;

}

G

dp，考虑以1-n分别为根节点，设dp[n][h]为n个结点高度为h的二叉搜索树的数量，1493717725(1)，然后可以做一次前缀和，这样就可以O(1)时间得到不小于h的解了

参考代码：

#include <cstdio>

long long ans[36][36];

int main()

{

int n,h;

ans[0][0]=1;

for(int i=1;i<=35;i++)

for(int j=1;j<=35;j++)

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

for(int l=0;l<j;l++)

{

ans[i][j]+=ans[k-1][j-1]\*ans[i-k][l];

if(l!=j-1)

ans[i][j]+=ans[i-k][j-1]\*ans[k-1][l];

}

for(int i=1;i<=35;i++)

for(int j=1;j<=35;j++)

ans[i][j]+=ans[i][j-1];

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

printf("%lld\n",ans[n][n]-ans[n][h-1]);

return 0;

}

广告：DH的博客 http://blog.csdn.net/m0\_37729344