**已知后序中序求前序**

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

char s1[101],s2[101];

int a[101];

int k;

char stack[101];

int numOf;

void fen(char \*s2,int l,int h);

int main()

{

int i,j;

while( scanf("%s %s",s1,s2)!= EOF){

int s1len=strlen(s1),s2len=strlen(s2);

k =s1len;

numOf=0;

for(i = 0;i < s1len;i++){

for(j = 0;j < s2len;j++){

if(s1[i] == s2[j]) a[i] = j;

}

}

fen(s1,0,strlen(s1) - 1);

for(int k=s1len-1;k>=0;k--)

{

printf("%c",stack[k]);

}

printf("\n");

}

return 0;

}

void fen(char \*s1,int l,int h){

int n;

if(h < l) return ;

if(h - l > 0) k--;

n = a[k];

if(l == h) {

//printf("%c %d ",s1[h],k);

stack[numOf++]=s1[h];

k--;

return ;

}

fen(s1,n+1,h);

fen(s1,l,n-1);

// printf("%c %d ",s1[n],k);

stack[numOf++]=s1[n];

}

**二叉树（建树）+层次遍历**

#include <iostream>

#include<stdio.h>

#include <queue>

using namespace std;

char fstr[27],mstr[27];

struct TNode{

TNode \*lchild,\*rchild;

char str;

TNode()

{

lchild=NULL;

rchild=NULL;

}

};

int search(char \*str,char x) //查找字符串，返回对应的数组下标

{

int i;

for(i=0;str[i]!='\0';i++)

if(str[i]==x) return i;

return -1;

}

void Fbit(TNode \*root,char x) //根据先序遍历数组，判断在目标结点的左边或右边。在叶节点处插入

{

int mindex,findex;

mindex=search(mstr,x);

findex=search(mstr,root->str);

if(mindex<findex)

{

if(root->lchild==NULL)

{

root->lchild=new TNode;

root->lchild->str=x;

}else Fbit(root->lchild,x);

}

else

{

if(root->rchild==NULL)

{

root->rchild=new TNode;

root->rchild->str=x;

}else Fbit(root->rchild,x);

}

}

void Bbit(TNode \*root) //后序遍历输出

{

if(root==NULL) return;

Bbit(root->lchild);

Bbit(root->rchild);

printf("%c",root->str);

}

void print(TNode \*T)

{

int parentSize=1,childSize=0;

TNode \*temp;

queue<TNode\*> q;

q.push(T);

do{

temp=q.front();

printf(" %c ",temp->str);

q.pop();

if(temp->lchild!=NULL)

{

q.push(temp->lchild);

childSize++;

}

if(temp->rchild!=NULL)

{

q.push(temp->rchild);

childSize++;

}

parentSize--;

if(parentSize==0)

{

parentSize=childSize;

childSize=0;

printf("\n");

}

}while(!q.empty());

}

int main()

{

while(gets(fstr))

{

gets(mstr);

int i;

TNode \*root = new TNode;

root->str=fstr[0];

for(i=1;fstr[i]!='\0';i++) //根据先序遍历数组，判断在目标结点的左边或右边。在叶节点处插入

{

Fbit(root,fstr[i]);

}

print(root);

Bbit(root);

printf("\n");

}

}

**死亡才分**

给定一个由整数组成的集合，集合中的整数各不相同，现在要将它分为两个子集合，使得这两个子集合的并为原集合、交为空集，同时在两个子集合的元素个数n1与n2之差的绝对值|n1-n2|尽可能小的前提下，要求它们各自的元素之和S1与S2之差的绝对值|S1-S2|尽可能大。

#include <iostream>

#include <cstdio>

#include <algorithm>

using namespace std;

typedef long long LL;

const int maxn=10000010;

int n;

LL A[maxn];

int main()

{

LL sum=0,sum1=0;

scanf("%d",&n);

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

{

scanf("%lld",&A[i]);

sum+=A[i];

}

nth\_element(A,A+n/2,A+n);

for(int i=0;i<n/2;i++)

{

sum1+=A[i];

}

printf("%d %lld\n",n%2,sum-sum1-sum1);

return 0;

}

**万妖**

#include <iostream>

#include <vector>

#include <queue>

#include <algorithm>

#include <stdio.h>

using namespace std;

const int MAX\_N=1001;

vector <int> G[MAX\_N],topOrder;

int vertex,edge,inDegree[MAX\_N]={0};

int num;

bool topologicalsort(){

num=0;

priority\_queue<int,vector<int>,greater<int> > q;

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

{

if(inDegree==0)

q.push(i);

}

while(!q.empty())

{

int u=q.top();

topOrder.push\_back(u);

q.pop();

for(int i=0;i<G[u].size();i++)

{

int v=G[u][i];

inDegree[v]--;

if(inDegree[v]==0)

q.push(v);

}

num++;

}

if(num==vertex) return true;

else return false;

}

int main()

{

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

{

if(vertex==0&&edge==0)

break;

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

{

int u,v;

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

inDegree[v]++;

G[u].push\_back(v);

}

if(topologicalsort()){

printf("YES\n");

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

{

cout<<topOrder[i];

if(i<vertex-1)

cout<<" ";

else printf("\n");

}

}

else {printf("NO\n");

printf("%d\n",vertex-num);}

}

return 0;

}

**大软和小软：**

征程 2016/3/16 22:38:54  
  
#include <iostream>  
#include<set>  
using namespace std;  
set<int>s1;  
int main() {  
int n;  
int j;  
while(cin>>n)  
{  
          while(n--)  
  {  
           cin>>j;  
   if(s1.count(j\*2)==1&&s1.count(j)==0)  
          s1.erase(j\*2);  
  s1.insert(j);  
  
  }  
  cout<<s1.size()<<endl;  
  s1.clear();  
}  
}