Q1

Set an array containing childcount, position of parent and whether it is a leaf. When input, parent’s slot’s childcount will increase by one. If parent has parent and parent was originally a leaf, change parent into not a leaf and make grandparent’s count decrease by 1. Finally check whether all non-leaf slot have childcount greater than 2.

#include <stdio.h>

struct cell{

int childcount;

int parentpos;

int isroot;

};

int main(){

int input=0;

int input2=0;

int no=0;

scanf("%d",&input);

struct cell array[input+1];

for (int i=0;i<input+1;i++){

array[i].childcount=0;

array[i].parentpos=0;

array[i].isroot=0;

}

for (int i=2;i<input+1;i++){

scanf("%d",&input2);

array[i].parentpos=input2;

if (array[input2].isroot==0){

//printf("leaf become root!");

array[input2].isroot++;

array[array[input2].parentpos].childcount--;

}

array[input2].childcount++;

}

//for (int i=0;i<input+1;i++){

// printf("i is , %d %d %d",array[i].childcount,array[i].parentpos,array[i].isroot);

//}

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

if (array[i].isroot==1)

{

if (array[i].childcount<3) {no=1;};

}

}

if (no) printf("No"); else printf("Yes");

return 0;}

Q2

Construct a linked list and array of integer. Input will be stored in the list and the array. Loop the array, each time will loop search the linked list for position of same number and put the cell into the head of linked list, plus output the position.

#include <stdio.h>

typedef struct cellT{

int num;

struct cellT\* next;

struct cellT\* last;

} cellT;

void addcell(cellT \*first,int b){

cellT\* newcell;

newcell=(cellT\*)malloc(sizeof(cellT));

newcell->num=b;

if (first->next==NULL){

first->next=newcell;

first->last=newcell;

newcell->last=first;

newcell->next=first;

}

else {

first->last->next=newcell;

newcell->last=first->last;

newcell->next=first;

first->last=newcell;

}

};

int findcell(cellT \*first,int b){//output position, put to front.

cellT \*newcell;

newcell=first->next;

int k=1;

while (newcell->num!=b)

{

newcell=newcell->next;

k++;

}

//swapping here

newcell->last->next=newcell->next;

newcell->next->last=newcell->last;

first->next->last=newcell;

newcell->next=first->next;

newcell->last=first;

first->next=newcell;

return k;

};

int main(){

struct cellT \*first;

first=(cellT\*)malloc(sizeof(cellT));

first->num=0;

first->next=NULL;

first->last=NULL;

int m,n,b;

scanf("%d%d",&m,&n);

int input[n];

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

scanf("%d",&b);

addcell(first,b);

}

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

int j=0;

scanf("%d",&b);

j=findcell(first,b);

printf("%d ",j);

}

}

Q3

Set an array to store the parent and children’s position integer of each person. If the person’s parent is -1, start a recursion call to record the level. The call will give the person’s children the person’s level+1, and continue the call on the person’s children.

The highest level will be recorded in the call. Lastly output the highest level+1.

#include <stdio.h>

int highest=0;

typedef struct cellT{

int parent;

int level;

int \*child;

int childcount;

} cellT;

void givetreelevel(cellT array[],int pos){

if (array[pos].childcount!=0)

{

for (int i=0;i<array[pos].childcount;i++){

array[array[pos].child[i]].level=array[pos].level+1;

if (array[pos].level+1>highest) highest=array[pos].level+1;

givetreelevel(array,array[pos].child[i]);

}

}

}

int main(){

int n,m;

scanf("%d",&n);

cellT array[n];

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

array[i].level=0;

array[i].childcount=0;}

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

scanf("%d",&m);

array[i].parent=m;

if (m!=-1){ array[m-1].childcount++;

if (array[m-1].childcount==1) {array[m-1].child=(int\*)malloc(sizeof(int));}

array[m-1].child=(int\*)realloc(array[m-1].child,(sizeof(int)\*(array[m-1].childcount+1)));

array[m-1].child[array[m-1].childcount-1]=i;

}

}

//similiar to recursion hard can delete

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

if (array[i].parent==-1)

{

givetreelevel(array,i);

}

}

/\*testing print only

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

printf("%d %d %d\n ",array[i].parent,array[i].level,array[i].childcount);

for (int j=0;j<array[i].childcount;j++)

{

printf("child are %d\n",array[i].child[j]+1);

}

}

\*/

printf("%d",highest+1);

return 0;

}