## 蓝桥杯赛前培训 2017年1月30日

真言：有的时候当你一头雾水的时候，强迫让自己冷静，冷静，再冷静！然后试着去读代码。你会发现柳暗花明又一村。。。

题目一：

求一个数组的连续和最大子数组，这里给大家介绍四种解法

共享代码：

double maxl(double a,double b,double c){

return a>b?(a>c?a:c):(b>c?b:c);

}

double maxl(double a,double b){

return a>b?a:b;

}

1:传统迭代

//时间复杂度O(N^2)

//传统迭代方法，用sum记录当前子数组的和，去sum和currentMax比较的最大值

double MaxItera(double \*a,int N){

double currentMax=INF;

double sum;

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

sum=0;

for (int j=i; j<N; j++) {

sum+=a[j];

currentMax=maxl(sum,currentMax);

}

}

return currentMax;

}

2: //联想斐波那契的最大子数组求和方法

//开辟一个数组用来记录a(0...N)的和maxL[i]=a[0]+a[2]+...+a[i];

//从数组a(i...j)的和sum=maxL[j]-maxL[i]

double MaxIteFebi(double \*a,int N){

double max=INF;

double \*maxL=new double[N];

maxL[0]=a[0];

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

maxL[i]=a[i]+maxL[i-1];

}

double sum;

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

for (int j=i; j<N; j++) {

sum=maxL[j]-maxL[i];

max=maxl(max, sum);

}

}

return max;

}

3: 采用分治的方法解决最大子数组的问题

double MaxDivQue(double \*a,int N,int low,int high){

if(low>high){return INF;}

if(low==high){return a[low];}

int mid=(low+high)/2;

double maxLeft,maxRight,sum;

sum=0;

maxLeft=INF;

for (int i=mid;i>=low;i-- ) {

sum+=a[i];

maxLeft=maxl(maxLeft, sum);

}

sum=0;

maxRight=INF;

for (int j=mid+1; j<=high; j++) {

sum+=a[j];

maxRight=maxl(maxRight, sum);

}

return maxl(maxLeft+maxRight, MaxDivQue(a, N, mid+1, high),MaxDivQue(a, N, low, mid) );

}

4: //采用启发式淘汰算法求解最大子数组的问题（感兴趣可以搜搜，虽然名字很好听，难度，呵呵哒！）

double MaxInspire(double \*a,int N){

double max\_end\_sum=0;

double currentMax\_so\_far=INF;

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

max\_end\_sum+=a[i];

if(max\_end\_sum<0){max\_end\_sum=0;}

currentMax\_so\_far=maxl(max\_end\_sum, currentMax\_so\_far);

}

return currentMax\_so\_far;

}

题目二：

**问题描述，按F向前跑97米，按B向后跑127米，问最少按F B多少次可以使向前跑的长度为1米**

#include <stdio.h>

int main(int argc, const char \* argv[]) {

int i,j,count[100]={0},m=0;

int min;

for(i=1;i<101;i++){

for(j=1;j<101;j++){

if((97\*i-j\*127)==1){

count[m++]=i+j;

}

}

}

min=count[0];

for(i=0;i<10;i++){

if(count[i]&&count[i]<min){

min=count[i];

}

}

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

return 0;

}

题目三：

（终于到了激动人心的数据结构）

题目描述：求一棵二叉树中两个节点的最近公共祖先节点

struct Node

{

struct Node \*left, \*right;

int key;

};

struct Node\* newNode(int key)

{

struct Node \*temp = newNode;

temp->key = key;

temp->left = temp->right = NULL;

return temp;

}

// 返回n1和n2的 LCA的指针

// 假设n1和n2都出现在树中

struct Node \*findLCA(struct Node\* root, int n1, int n2)

{

if (root == NULL) return NULL;

// 只要n1 或 n2 的任一个匹配即可

// (注意：如果 一个节点是另一个祖先，则返回的是祖先节点。因为递归是要返回到祖先的 )

if (root->key == n1 || root->key == n2)

return root;

// 分别在左右子树查找

struct Node \*left\_lca = findLCA(root->left, n1, n2);

struct Node \*right\_lca = findLCA(root->right, n1, n2);

// 如果都返回非空指针 Non-NULL, 则说明两个节点分别出现了在两个子树中，则当前节点肯定为LCA

if (left\_lca && right\_lca) return root;

// 如果一个为空，在说明LCA在另一个子树

return (left\_lca != NULL)? left\_lca: right\_lca;

}

题目四：

求一个矩阵中的马鞍点。（鞍点：矩阵中的某一点，这点在所处行最小，在所处列最大）

#define Max\_row 3

#define Max\_col 4

void SearchS(int a[][Max\_col]){

int i,j,k,iMin,t,cnt=0;

int row,col;

col=Max\_col; /\*求出二维数组的列数\*/

row=Max\_row; /\*求出二维数组的行数\*/

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

iMin=a[i][0];

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

if (a[i][j]<iMin) {

iMin=a[i][j]; /\* 找到每行的最小值，将其值赋给iMin \*/

}

}

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

if (!(a[i][j]-iMin)) { /\*找到该行最小值，并记录其下标，判断它是不是所在列的最大值\*/

k=j;

t=0;

while (t<row&&(iMin>=a[t][k])) {

t++;

}

if (!(t-row)) {

cnt++;

printf("第%d个鞍点是：(%d,%d),其值为%d",cnt,i+1,k+1,a[i][k]);

}

}

}

}

if (!cnt) {

printf("该矩阵无鞍点!");

}

}

void display(int a[][Max\_col]){

int i,j;

int row,col;

col=Max\_col; /\*求出二维数组的列数\*/

row=Max\_row; /\*求出二维数组的行数\*/

printf("您的矩阵是：\*\*\*\*\*\*\*\*\*\*\*\*\n");

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

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

printf("%d ",a[i][j]);

}

printf("\n");

}

}