**实验四 减治法**

1. **实验目的**

1．了解减治法的概念和特点。

2．学会正确使用减治法。

3．熟悉掌握减治法时间效率的分析方法。

4．掌握减治法在实际问题中的应用。

1. **实验要求**
2. 熟悉减治法的基本思想。
3. 运用减治法解决实际问题，加深对减治法的理解和运用。
4. 能得到正确的程序运行结果。
5. **主要仪器及耗材**

计算机及相关软件

1. **实验内容**

1. 对于以邻接矩阵定义的图，用如下算法检测其连通性。

算法：Connected (A[0..n-1,0..n-1])

//输入：无向图的邻接矩阵A[0..n-1,0..n-1]

//输出：如果G是连通的，输出为1（true）,否则输出为0（false）

if n=1 return 1 //单一顶点的图显然是连通的

else

if not Connected(A[0..n-2,0..n-2]) return 0

else for j←0 to n-2 do

if A[n-1,j] return 1

return 0

**问题：**

①. 该算法是否对每个n>0个顶点的无向图都能够正确运行吗？**能**

②. 如果回答是，请说明最坏情况下的算法效率类型，并举出实例。如果回答否，请说明为什么？

**O(n\*n)**

③. 结合实验二中的求是否是完全图的程序，上机实现此算法，看两个算法有什么不同。

**实验二的代码可以用在这里。**

**完全图一定是连通图，连通图不一定是完全图。**

**#include<stdio.h>**

**#define N 100**

**int Connected(int array[][100],int n)**

**{**

**if(n==1) return 1;**

**else**

**{**

**if(!Connected(array,n-1))**

**return 0;**

**else**

**{**

**for(int j = 0;j <= n-1; j++)**

**if(array[n-1][j]) return 1;**

**return 0;**

**}**

**}**

**}**

**int main()**

**{**

**int a[N][N];**

**int n;**

**printf("输入你的矩阵的阶数：\n");**

**scanf("%d",&n);printf("\n");**

**printf("输入你的矩阵的元素：\n");**

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

**for(int j=0;j<n;j++)**

**{**

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

**}**

**printf("%d",Connected(a,n));**

**}**

1. 查阅数据结构课程中希尔排序的相关资料，任意选择一种语言实现希尔排序、直接插入排序、选择排序以及冒泡排序，然后对于序列大小为10n，n=2，，,,4,5,6的随机序列、升序序列和降序序列分别比较它们的性能。

**希尔排序：**

**#include<iostream>**

**using namespace std;**

**void shell\_sort(int a[],int n);**

**int main()**

**{**

**int n,a[10000];**

**cin>>n;**

**for(int y=0;y<n;y++)**

**cin>>a[y];**

**shell\_sort(a, n);**

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

**cout<<a[i]<<" ";**

**cout<<endl;**

**}**

**void shell\_sort(int a[], int n)**

**{**

**int gap;//定义增量；**

**for(gap = 3; gap >0; gap--)//设置初始增量，递减；**

**{**

**for(int i=0; i<gap; i++)//按增量分组；**

**{**

**for(int j = i+gap; j<n; j=j+gap)//每组分别比较大小；**

**{**

**if(a[j]<a[j-gap])**

**{**

**int temp = a[j];**

**int k = j-gap;**

**a[j]=a[k];**

**a[k]=temp;**

**}**

**int temp=a[j];**

**int k=j-gap;**

**if(j!=i-k)**

**{**

**a[k+gap] = temp;**

**}**

**}**

**}**

**}**

**}**

**直接插入排序：**

**#include <stdio.h>**

**#include <stdlib.h>**

**int main()**

**{**

**int i,j;**

**int a[10];**

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

**{**

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

**}**

**for(i=1; i<10; i++)**

**{**

**int t;**

**t=a[i];//设置哨兵**

**for(j=i-1; a[j]>t&&j>=0; j--)//寻找插入位置**

**a[j+1]=a[j];//前一个大于当前，当前等于前一个**

**a[j+1]=t;**

**}**

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

**printf("%d\n",a[i]);**

**return 0;**

**选择排序：**

**#include <stdio.h>**

**#include <stdlib.h>**

**int main()**

**{**

**int i,j,k;**

**int a[10];**

**int t;**

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

**scanf("%d",&a[i]);//输入10个待排序的数**

**for(i=0;i<9;i++)**

**{**

**k=i;//默认待排序数中的第一个数为最小的**

**for(j=i+1;j<10;j++)**

**if(a[j]<a[k])//若下一个数比这个数更小**

**k=j;//k 始终指示出现的较小的元素的位置**

**if(i!=k)//当前的数不是最小的，将它与最小的数交换**

**{**

**t=a[i];**

**a[i]=a[k];**

**a[k]=t;**

**}**

**}**

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

**printf("%d\n",a[i]);**

**return 0;**

**冒泡排序：**

**#include <stdio.h>**

**#include <stdlib.h>**

**int main()**

**{**

**int t;**

**int i,j;**

**int a[10]= {9,8,7,6,5,4,3,2,1,0};**

**for(j=0; j<9;)**

**{**

**for(i=0; i<9-j; i++)**

**{**

**if(a[i]>a[i+1])**

**{**

**t=a[i];**

**a[i]=a[i+1];**

**a[i+1]=t;**

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

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

**printf("\n");**

**}**

**}**

**}**

**j++;**

**return 0;**

**}**

**比较性能采用随机数的方式进行比较即可。**

**随机序列：希尔>插入>冒泡>简选**

**升序序列：冒泡>插入>希尔>简选**

**降序序列：希尔>插入>冒泡>简选**

1. 任选一种语言实现这两种拓扑排序算法并做一个实验来比较它们的运行时间。

# **两种算法分别为：基于Kahn算法和dfs算法**

**Kahn算法：**

**public class KahnTopological**

**{**

**private List<Integer> result; // 用来存储结果集**

**private Queue<Integer> setOfZeroIndegree; // 用来存储入度为0的顶点**

**private int[] indegrees; // 记录每个顶点当前的入度**

**private int edges;**

**private Digraph di;**

**public KahnTopological(Digraph di)**

**{**

**this.di = di;**

**this.edges = di.getE();**

**this.indegrees = new int[di.getV()];**

**this.result = new ArrayList<Integer>();**

**this.setOfZeroIndegree = new LinkedList<Integer>();**

**// 对入度为0的集合进行初始化**

**Iterable<Integer>[] adjs = di.getAdj();**

**for(int i = 0; i < adjs.length; i++)**

**{**

**// 对每一条边 v -> w**

**for(int w : adjs[i])**

**{**

**indegrees[w]++;**

**}**

**}**

**for(int i = 0; i < indegrees.length; i++)**

**{**

**if(0 == indegrees[i])**

**{**

**setOfZeroIndegree.enqueue(i);**

**}**

**}**

**process();**

**}**

**private void process()**

**{**

**while(!setOfZeroIndegree.isEmpty())**

**{**

**int v = setOfZeroIndegree.dequeue();**

**// 将当前顶点添加到结果集中**

**result.add(v);**

**// 遍历由v引出的所有边**

**for(int w : di.adj(v))**

**{**

**// 将该边从图中移除，通过减少边的数量来表示**

**edges--;**

**if(0 == --indegrees[w]) // 如果入度为0，那么加入入度为0的集合**

**{**

**setOfZeroIndegree.enqueue(w);**

**}**

**}**

**}**

**// 如果此时图中还存在边，那么说明图中含有环路**

**if(0 != edges)**

**{**

**throw new IllegalArgumentException("Has Cycle !");**

**}**

**}**

**public Iterable<Integer> getResult()**

**{**

**return result;**

**}**

**}**

**基于DFS算法：**

**#include <stdio.h>**

**#include<stdlib.h>**

**#define maxSize 100**

**typedef struct EdgeNode{//边结点**

**int adjvex;**

**struct EdgeNode\* next;**

**}EdgeNode;**

**typedef struct Vnode{//顶点结点**

**int data;**

**int in;//新增的入度个数**

**EdgeNode \*firstEdge;**

**}Vnode,Adjvex[maxSize];**

**typedef struct AGraph{//图结点**

**Adjvex adjvex;**

**int n,e;**

**}AGraph;**

**void createGraph(AGraph &G){//邻接表创建图**

**EdgeNode \*p;**

**int i,j,k;**

**printf("请输入图的顶点数和边数\n");**

**scanf("%d %d",&G.n,&G.e);**

**printf("请输入图的顶点\n");**

**for(i=0;i<G.n;i++){//输入图的顶点**

**scanf("%d",&G.adjvex[i]);**

**G.adjvex[i].firstEdge=NULL;**

**G.adjvex[i].in=0;//所有的顶点入度初始化为0**

**}**

**printf("请输入边的下标i j\n");**

**for(k=0;k<G.e;k++){//输入图的边**

**scanf("%d %d",&i,&j);**

**p=(EdgeNode \*)malloc(sizeof(EdgeNode));**

**p->adjvex=j;**

**p->next=G.adjvex[i].firstEdge;**

**G.adjvex[i].firstEdge=p;**

**G.adjvex[j].in++; //j顶点入度自增长**

**}**

**}**

**int topSort(AGraph G){//拓扑排序**

**EdgeNode \*p;**

**int i,j,k,count=0;**

**int \*que;//栈里面存放的是顶点下标**

**que=(int \*)malloc(G.n\*sizeof(int));**

**int top=-1;//创建并初始化栈**

**for(i=0;i<G.n;i++){**

**if(!G.adjvex[i].in){//若入度为0的,则入栈**

**que[++top]=i;**

**}**

**}**

**while(top!=-1){//若栈不为空**

**k=que[top--];//栈顶出栈**

**printf("%d ",G.adjvex[k].data);**

**count++;//计数访问了多少顶点**

**for(p=G.adjvex[k].firstEdge;p!=NULL;p=p->next){//遍历出栈顶点的边表**

**G.adjvex[p->adjvex].in--;//与之相邻接的顶点的入度自减**

**if(G.adjvex[p->adjvex].in==0){//把自减后入度为0的顶点入栈**

**que[++top]=p->adjvex;**

**}**

**}**

**}**

**if(count<G.n){//若有顶点未访问到,则拓扑排序失败**

**return 0;**

**}else{**

**return 1;**

**}**

**}**

**int main(int argc, char\*\* argv) {**

**AGraph G;**

**createGraph(G);**

**topSort(G);**

**return 0;**

**}**

1. 设计一个只使用两路比较的折半查找的版本，例如只用<=和=,可以任选一种语言来实现，并认真调试。对于你设计的两路比较算法，分析其时间效率。

**#include<stdio.h>**

**int TBSearch(int A[],int q,int K)**

**{**

**int l=0,r=q-1;**

**int m;**

**while(l<r)**

**{**

**m=(l+r)/2;**

**if(K<=A[m]) r=m;**

**else l=m+1;**

**}**

**if(K==A[l]) return l;**

**else return -1;**

**}**

**int main()**

**{**

**int a[10]={0,2,3,4,5,6,7,8,9,10};**

**int k;**

**printf("输入你想查找的数：\n");**

**scanf("%d",&k);**

**printf("这个数的下标是：%d",TBSearch(a,10,k));**

**}**

**时间复杂度为O（）**

1. 为俄式乘法编写伪代码，并上机实现。

**伪代码：**

**sum=0**

**while m!=1 do**

**if m%2=0**

**m=m/2 n=n\*2**

**else**

**sum=sum+n m=m/2 n=n\*2**

**sum=sum+n**

**return sum**

**上机实现的代码：**

**#include<math.h>**

**int multi(int n,int m){**

**int sum=0;**

**while(n>1){**

**if(n%2==1)**

**sum+=m;**

**m=m\*2;**

**n=ceil(n/2);**

**printf("当前的n和m是：%d %d\n",n,m);**

**}**

**sum+=m;**

**return sum;**

**}**

**int main(){**

**int n,m;**

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

**printf("%d",multi(n,m));**

**}**

6. 应用快速选择算法求9,12,5,17,20,30,8的中位数。

**#include<stdio.h>**

**int a[7]= {9,12,5,12,20,30,8};**

**int flag=3;**

**//flag作为数组的中间值的数组下标(low+high)/2;**

**//若是排序一趟之后i==j&&i==flag则找到中位数**

**int quickSortMidSearch(int array[],int low,int high) {**

**if (low < high) {**

**int i = low;**

**int j = high;**

**int temp = array[low];**

**while (i < j) {**

**while(i < j && array[j] >= temp) {**

**j--;**

**}**

**if(i < j) {**

**array[i++] = array[j];**

**}**

**while(i < j && array[i] < temp) {**

**i++;**

**}**

**if(i < j) {**

**array[j--] = array[i];**

**}**

**}**

**array[i] = temp;**

**if(i==flag)**

**printf("%d",array[i]);;**

**if(i>flag)**

**quickSortMidSearch(array, low, i - 1);**

**if(i<flag)**

**quickSortMidSearch(array, i + 1, high);**

**}**

**}**

**int main() {**

**quickSortMidSearch(a,0,6);**

**}**

1. **注意事项**
2. 注意减治算法的递归设计。

2. 注意减治算法的时间复杂度的分析。