

算法分析与设计

实

验

报

告

**姓名：栾佳锡**

**班级：计算机54班**

**学号：2150500131**

实验一 线性时间查找中位数

1. 问题描述

编写程序，实现先线性时间内选择n个元素的中位数算法，并对不同的n，测试平均时间效率。

1. 问题分析及理解

线性时间选择问题是第二章中的内容，需要采用**递归分治策略**。第一次以数组首元素为基准把数组划分，使得首元素a[p]左侧元素均小于它，右侧元素均大于它。记此时a[p]下标为i。设数组长度为n，则若i=n，则a[p]即为中位数；若i<n,则中位数定在a[p]的右侧出现；若i>n，则中位数定在a[p]左侧出现。故再对a[p]左侧或右侧元素按上述方法进行递归查找。这种形况下，平均时间复杂度为O(n)。

1. 算法程序

#include "stdlib.h"

#include <ctime>

#include <iostream>

#include <windows.h>

using namespace std;

template <class Type>

void Swap(Type &x,Type &y);

inline int Random(int x, int y);

template <class Type>

void BubbleSort(Type a[],int p,int r);

template <class Type>

int Partition(Type a[],int p,int r,Type x);

template <class Type>

Type Select(Type a[],int p,int r,int k);

int main()

{

//初始化数组

int n=40000,N=1;

cout<<"The Total Number Is:"<<n<<endl;//总共多少个数

int \*a = new int[n];

double t = 0;

double counts = 0;

LARGE\_INTEGER nFreq;

LARGE\_INTEGER nBeginTime;

LARGE\_INTEGER nEndTime;

QueryPerformanceFrequency(&nFreq);

QueryPerformanceCounter(&nBeginTime);//开始计时

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

//必须放在循环体外面,更改种子值，避免每次随机数列相等

srand((unsigned)time(0));

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

a[i] = Random(0,n);

if(n%2==1)

cout<<"Mid Is"<<Select(a,0,n-1,n/2)<<endl;

else cout<<"Mid Is"<<(Select(a,0,n-1,n/2-1)+Select(a,0,n-1,n/2))/2<<endl;

}

QueryPerformanceCounter(&nEndTime);//停止计时

t = (double)(nEndTime.QuadPart - nBeginTime.QuadPart) / (double)nFreq.QuadPart;//计算程序执行时间单位为s

cout << "Running Time Is " << t \* 1000 << "ms" << endl;

system("pause");

}

template <class Type>

void Swap(Type &x,Type &y)

{

Type temp = x;

x = y;

y = temp;

}

inline int Random(int x, int y)

{

int ran\_num = rand() % (y - x) + x;

return ran\_num;

}

//冒泡排序

template <class Type>

void BubbleSort(Type a[],int p,int r)

{

//记录一次遍历中是否有元素的交换

bool exchange;

for(int i=p; i<=r-1;i++)

{

exchange = false ;

for(int j=i+1; j<=r; j++)

{

if(a[j]<a[j-1])

{

Swap(a[j],a[j-1]);

exchange = true;

}

}

//如果这次遍历没有元素的交换,那么排序结束

if(false == exchange) break ;

}

}

template <class Type>

int Partition(Type a[],int p,int r,Type x)

{

int i = p-1,j = r + 1;

while(true)

{

while(a[++i]<x && i<r);

while(a[--j]>x);

if(i>=j) break;

Swap(a[i],a[j]);

}

return j;

}

template <class Type>

Type Select(Type a[],int p,int r,int k)

{

if(r-p<75)

{

BubbleSort(a,p,r);

return a[p+k-1];

}

//(r-p-4)/5相当于(n-5)/5

for(int i=0; i<=(r-p-4)/5; i++)

{

//将元素每5个分成一组，分别排序，并将该组中位数与a[p+i]交换位置

//使所有中位数都排列在数组最左侧，以便进一步查找中位数的中位数

BubbleSort(a,p+5\*i,p+5\*i+4);

Swap(a[p+5\*i+2],a[p+i]);

}

//找中位数的中位数

Type x = Select(a,p,p+(r-p-4)/5,(r-p-4)/10);

int i = Partition(a,p,r,x);

int j = i-p+1;

if(k<=j) return Select(a,p,i,k);

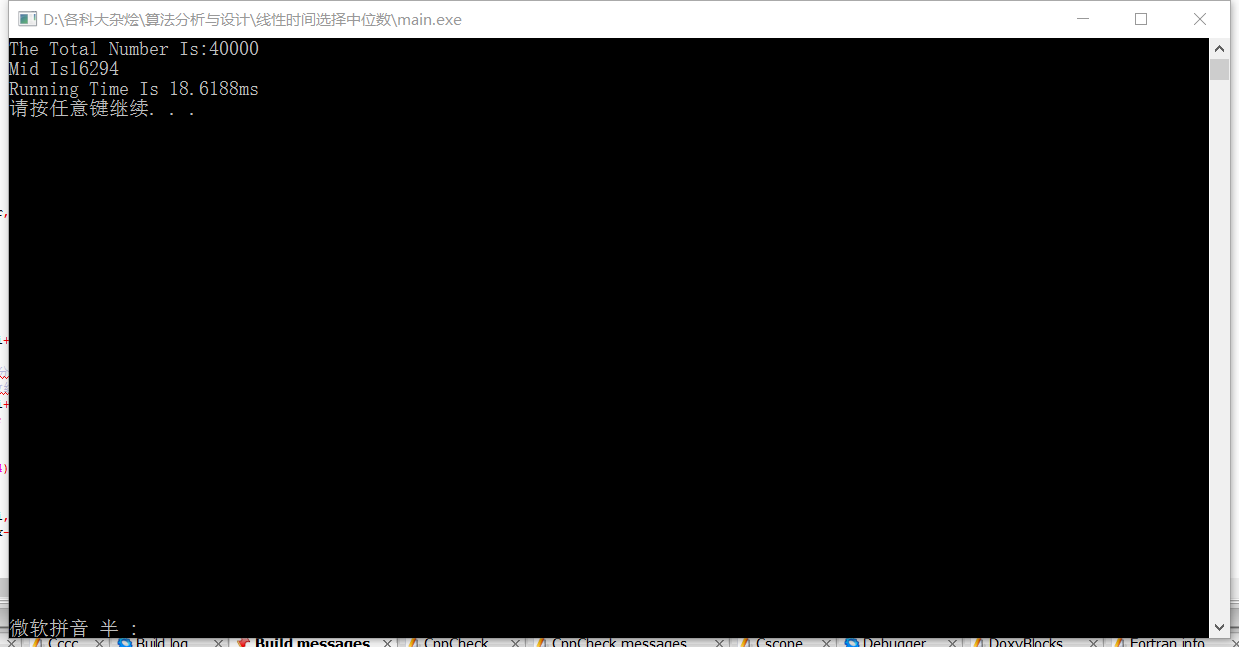
else return Select(a,i+1,r,k-j);

}

1. 与同学交流学习结果

程序主体式找到数组首元素位置，并用i记录该位置。然后判断i与n/2的大小关系进行递归运算。在主函数中，用字符s判断当输入“回车”时结束输入，并用c记录数组长度，算法实现较为简单。

1. 运行结果



实验二 最优二叉查找树

1. 问题描述

实现构造最优二叉树的算法

1. 问题分析

采用动态规划解题，如果一棵最优二叉查找树T有一棵包含关键字ki,..,kj的子树T'，那么这可子树T'对于关键字Ki,...,kj和虚拟键di-1,...dj的子问题也必定是最优的。根据最优子结构，寻找最优解：给定关键字ki,...,kj，假设kr(i<=r<=j)是包含这些键的一棵最优子树的根。其左子树包含关键字ki,...,kr-1和虚拟键di-1,...,dr-1，右子树包含关键字kr+1,...,kj和dr,...dj。我们检查所有的候选根kr,就保证可以找到一棵最优二叉查找树。递归解：定义e[i,j]为包含关键字ki,...,kj的最优二叉查找树的期望代价，最终要计算的是e[1,n]。当j = i - 1时，此时子树中只有虚拟键，期望搜索代价为e[i,i - 1] = qi-1。当j >= i时，需要从ki,...,kj中选择一个根kr，然后分别构造其左子树和右子树。下面需要计算以kr为根的树的期望搜索代价。然后选择导致最小期望搜索代价的kr做根。

1. 算法程序

#include<cstdio>

#include<cfloat>

#include<iostream>

#define INF DBL\_MAX

#define m\_Size 100

using namespace std;

double m[m\_Size][m\_Size];int root[m\_Size][m\_Size];

void optimal\_BST(double p[],double q[],int n)

//p[1,n]表示实结点的概率,q[0,n]表示虚拟结点的搜索概率，n表示实结点的个数

{

double w[n+2][n+2];

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

m[i][i-1] = q[i-1];

w[i][i-1] = q[i-1];

}

for(int l =1 ; l<=n ; l++)

for(int i= 1 ; i<=n-l+1 ; ++i){

int j = i+l-1;

m[i][j] = INF;

w[i][j] = w[i][j-1] +p[j]+q[j];

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

double t = m[i][r-1]+m[r+1][j]+w[i][j];

if(t<m[i][j]){

root[i][j] = r;

m[i][j] = t;

}

}

}

}

// 输出最优二叉树

// ij代表首尾位置、初始时p置为0，表明当前无根结点

void out\_BST(int i,int j,int p)

{

//叶节点 判断其位置

if(i-j==1)

{

if(j<p) printf("b%d is a%d left child\n",j,p);

else printf("b%d is a%d right child\n",j,p);

return;

}

//根结点 输出根位置 迭代

if(p==0)

{

p = root[i][j];

printf("a%d is root\n",p);

out\_BST(i,p-1,p);

out\_BST(p+1,j,p);

} else if(root[i][j] <p)

{//根的左子树

printf("a%d is a%d left child\n",root[i][j],p);

p = root[i][j];

out\_BST(i,p-1,p);

out\_BST(p+1,j,p);

} else

{//根的右子树

printf("a%d is a%d right child\n",root[i][j],p);

p = root[i][j];

out\_BST(i,p-1,p);

out\_BST(p+1,j,p);

}

}

int main(){

double a[]={0,0.10,0.15,0.20,0.15,0.10};

double b[]={0.02,0.03,0.10,0.10,0.02,0.03};

int n=sizeof(a)/sizeof(a[0])-1;

cout<<"ture gailv a[1:5]: ";

for(int i=1;i<=n;i++) cout<<a[i]<<" "; cout<<endl;

cout<<"the probability of virtual dot is : b[0:5] ";

for(int i=0;i<=n;i++) cout<<a[i]<<" "; cout<<endl;

optimal\_BST(a,b,n);

cout<<endl<<"the shortest average distance is: "<<m[1][n]<<endl;

out\_BST(1,n,0);

cout<<endl<<"root tree:"<<endl;//根构造树

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

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

cout<<root[i][j]<<'\t';

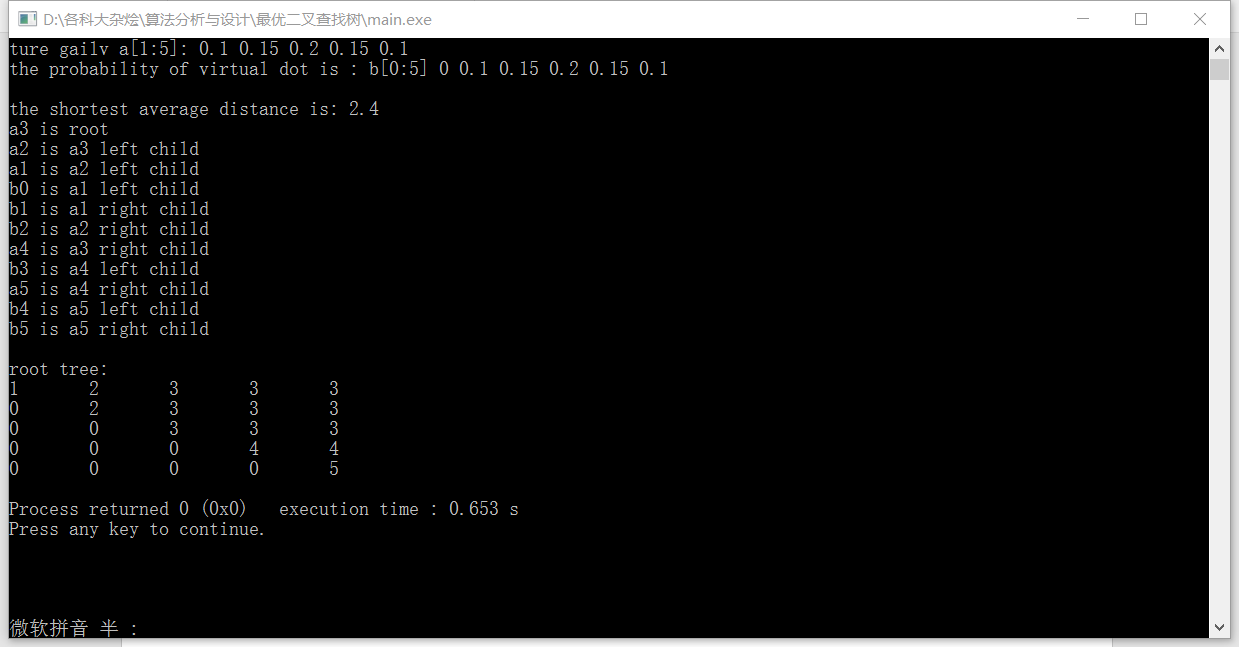
cout<<endl;

}

return 0;

}

1. 运行结果



实验三 贪心法最小删除方案

1. 问题描述

给定n位正整数，去掉其中任意k<n位，剩下的数字按原次序排列组成一个新正整数。对于给定的n位正整数和k，设计算法找出剩余数字组成的正整数最小的删除方案。

1. 问题分析

算法的原理应该是说从最高位开始，一次向低位搜索，一旦遇到前一位（高位）的数大于当前位，则删去前一位，直到删除k个数，如果到达末尾还没有删除k个，则说明现在这个数已经是从小到大排序了，则从最低位开始删除要求的位

1. 算法程序

//n表示输入整数的位数，k表示欲删除的位数

#include<iostream>

#include<math.h>

#include<cstdlib>

using namespace std;

int main()

{

long in\_num;

int n,k;

int i,tmp;

char choice;

bool flag=true;

while(flag){

cout<<"\*\*\* Please input an integer:";

cin>>in\_num;

cout<<"\*\*\* Please input the number of digits to be deleted:";

cin>>k;

n=log10(in\_num)+1;

if(k>=n) {

cout<<"<<< k cannot be greater than n, Please retry >>>"<<endl;

continue;

}

//将整数各位取至数组中

int a[n];

tmp=in\_num;

for(i=n-1;i>=0;i--){

a[i]=tmp%10;

tmp/=10;

}

//控制删去的个数

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

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

if(a[j]<a[j+1]) continue;

else{ a[j]=-1; break;}

}

//删掉的数的位置被赋值-1，现在把-1以后的数都往前挪

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

if(a[j]==-1){

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

a[j+1]=-1;

}

}

}

cout<<"\*\*\* After deletion: ";

for(i=0;i<n-k;i++)

cout<<a[i];

cout <<endl;

cout<<"\*\*\* Start over? (Y/N)";

cin>>choice;

cout<<endl;

if(choice=='Y'||choice=='y');

if(choice=='N'||choice =='n') flag=false;

}

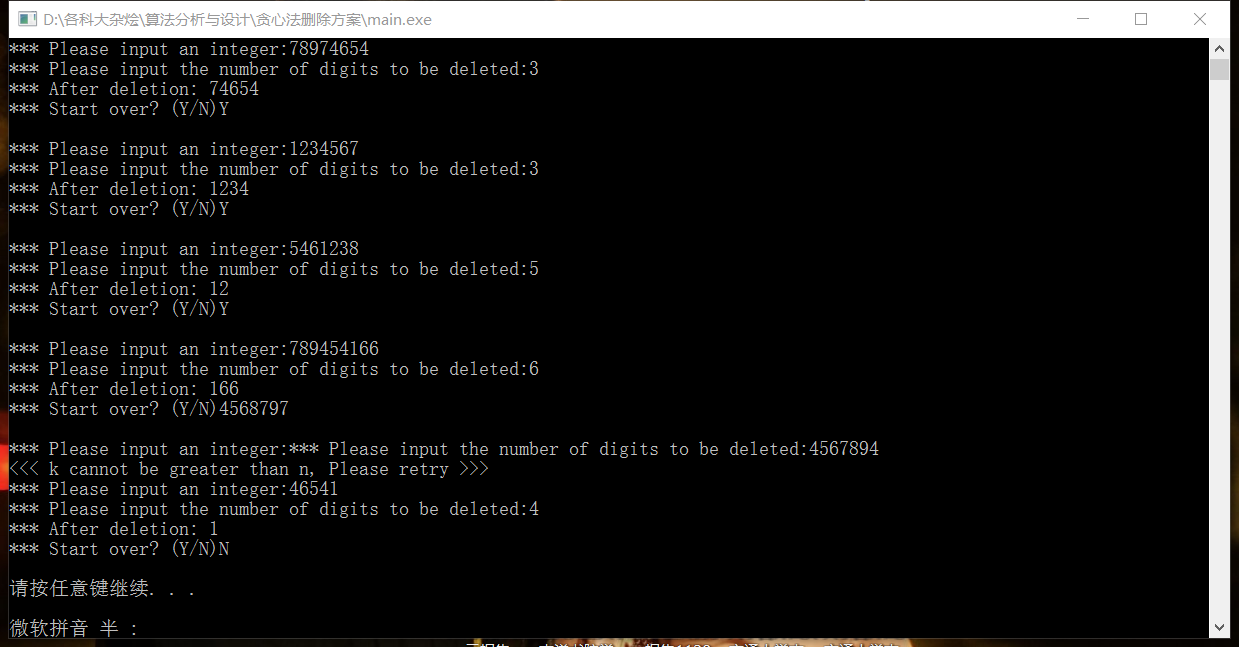
system("pause");

return 0;

}

}

1. 实验结果



时间复杂度：0（k\*(n-k)）

一共有n-k数字，求每个数字要用的区间大小是k

实验四 多机调度问题

1. 问题描述

设有n个独立的作业{1,2,…,n},由m台相同的机器进行加工处理。作业i所需要的处理时间为ti。约定，每个作业均可在任何一台机器上加工处理，但未完工前不允许中断处理，作业不能拆分为更小的子作业。给出一种作业调度方案，使所给的n个作业在尽可能短的时间内由m台相同的机器加工完成。

1. 问题分析

(1)分支限界法求解；假设有n个任务和k个机器，首先将问题转化为一颗解；分支限界法的本质是对解空间树的BFS（广度优先）；为了实现剪枝，我们需要定义一个处理时间上界up，；

(2)采用优先队列容器；在进行BFS搜索时，需要用队列来存储解空间树的各；

(3)搜索过程中动态申请\\释放数组；在BFS搜索过程中，解空间树的每个状态节点所记录的各机器完成时间都采用动态数组记录。

三、 算法程序

#define task 10

#define machine 3

#include <iostream>

#include <climits>

using namespace std;

void BackTrack(int j);

int getTime(int time\_Machine[]);

void output\_Assignment(int best\_x[]);

//所有数组下标从1开始

int x[task + 1];//x[i]表示给任务i分配机器x[i]

int best\_x[task+1];//存储最优分配方案

int min\_t=INT\_MAX;//执行任务所需的最小时间

int t[task + 1] = {0,2,11,6,3,1,5,2,7,8,4};//每个任务所需时间

int time\_Machine[machine + 1] = {0};//每个机器运行结束的时间

int main() {

BackTrack(1);

cout << "the time of the task is:" << endl;

for (int i = 1; i <= task; i++) cout << t[i] << " ";

cout << endl;

cout << "the shortest time is:"<<min\_t << endl;

output\_Assignment(best\_x);

return 0;

}

void BackTrack(int j) {

if (j > task) {

int cur\_time = getTime(time\_Machine);//当前已分配任务的完成时间

if (cur\_time < min\_t) { //剪枝

min\_t = cur\_time;

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

best\_x[i] = x[i];

}

}

else{

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

x[j] = i;

time\_Machine[i] += t[j];

if(time\_Machine[i]<min\_t)

BackTrack(j+1);

time\_Machine[i] -= t[j];

}

}

}

int getTime(int time\_Machine[]) {

int max\_time=time\_Machine[1];

for (int i = 2; i <= machine; i++) {

if (time\_Machine[i] > max\_time)

max\_time = time\_Machine[i];

}

return max\_time;

}

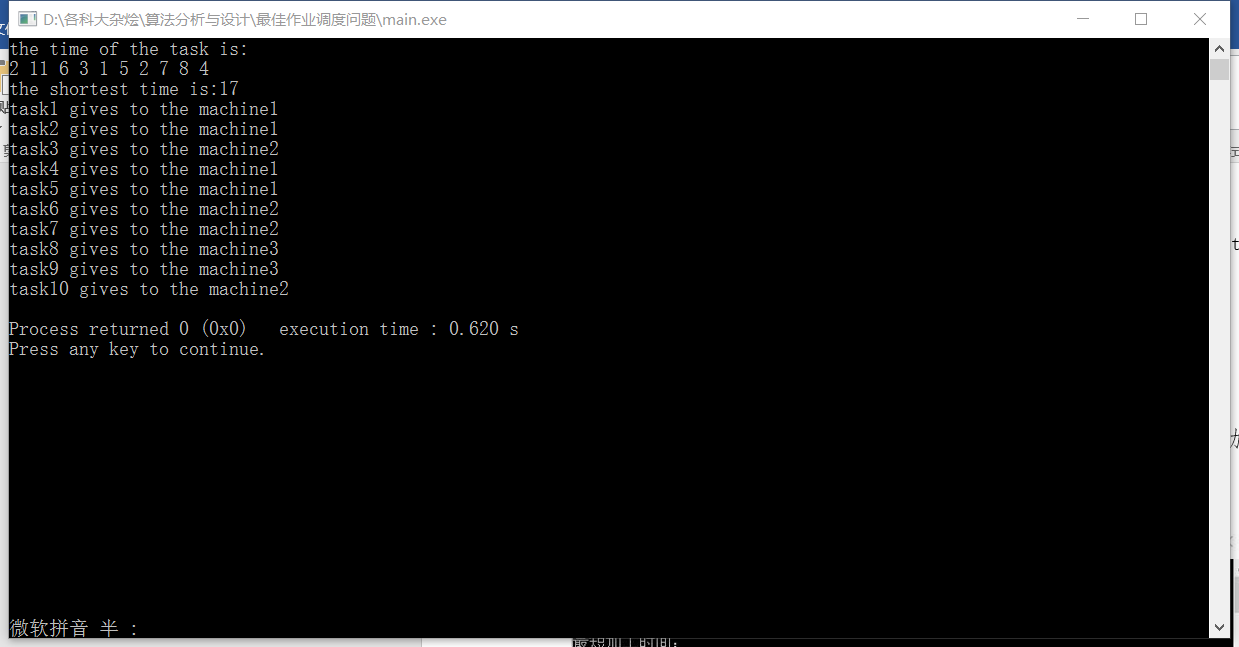
void output\_Assignment(int best\_x[]) {

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

cout << "task" << i << " gives to the machine" << best\_x[i] << endl;

}

1. 运行结果



当作业数小于机器数时，最短加工时间即为最长加工时间。