|  |
| --- |
|  |
| **课 程 实 验 报 告**  **课程名称：计算机算法基础**  **院 系 ：计算机科学与技术**  **专业班级 ： CS1409**  **学 号 ： U201414808**  **姓 名 ： 王林**  **指导教师 ：**  **2016 年 12 月 27 日** |

**华 中 科 技 大 学**

**目录**

[实验一 QUICKSORT 4](#_Toc471544446)

[1.实验题目 4](#_Toc471544447)

[2.实验目的与要求 4](#_Toc471544448)

[3.算法设计 5](#_Toc471544449)

[4.实验环境 6](#_Toc471544450)

[5.实验过程 6](#_Toc471544451)

[6.实验结果 8](#_Toc471544452)

[7.结果分析 9](#_Toc471544453)

[实验二 SELECT2 10](#_Toc471544454)

[1.实验题目 10](#_Toc471544455)

[2.实验目的与要求 10](#_Toc471544456)

[3.算法设计 10](#_Toc471544457)

[4.实验环境 11](#_Toc471544458)

[5.实验过程 11](#_Toc471544459)

[6.实验结果 13](#_Toc471544460)

[7.结果分析 13](#_Toc471544461)

[实验三 SHORTEST-PATHS 15](#_Toc471544462)

[1.实验题目 15](#_Toc471544463)

[2.实验目的与要求 15](#_Toc471544464)

[3.算法设计 15](#_Toc471544465)

[4.实验环境 16](#_Toc471544466)

[5.实验过程 16](#_Toc471544467)

[6.实验结果 18](#_Toc471544468)

[7.结果分析 18](#_Toc471544469)

[附件实验 NQUEENS 19](#_Toc471544470)

[1.实验题目 19](#_Toc471544471)

[2.实验目的与要求 19](#_Toc471544472)

[3.算法设计 19](#_Toc471544473)

[4.实验环境 20](#_Toc471544474)

[5.实验过程 20](#_Toc471544475)

[6.实验结果 22](#_Toc471544476)

[7.结果分析 22](#_Toc471544477)

实验一 QUICKSORT

1.实验题目

编程实现算法4.14：QUICKSORT的迭代模型，编制以下程序：

PARTITION

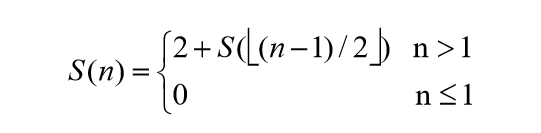
QUICJSIRT2

2.实验目的与要求

(1)栈可以用数组实现，但要有溢出检测

(2)编制测试数据，给出实验结果

(3)分析该算法的空间复杂度，特别是需要说明为什么其空间复杂度表达式是：



3.算法设计

开 始

A[n],j,top = 0;

p < q

j = q + 1

call PARTITION(p,j)

j-p < q-j

stack[top+1]=j+1

stack[top+2]=q

stack[top+1]=j+1

stack[top+2]=q

top = top + 2

top =0?

q = stack[top];p = stack[top-1]

top = top - 2

return

**图1-1 QUOCKSORT算法流程图**

4.实验环境

操作系统：Ubuntu 16.04 编译器：gcc 编辑器：vim 调试器：gdb 采用c语言实现。

5.实验过程

(1)依照算法设计，采用c语言实现算法，代码实现如下：

【quicksort.c】

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#include<stdio.h>

#include<stdlib.h>

#define MAX 20

#define NUM 10

int partition(int m,int p);

void quicksort(int p,int q);

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

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

int array[10] = {43,54,65,19,0,5,245,78,83,1};

int main(){

int i;

//打印排序前的数组

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

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

}

printf("\n\n");

//排序

quicksort(0,9);

//打印排序后的数组

printf("\n");

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

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

}

printf("\n");

return 0;

}

int partition(int m,int p){

int v,i,j;

i = m + 1;

j = p;

v = array[m];

int temp;

for(;;){

for(;(v >= array[i])&&(i <= p);i++);

for(;(v <= array[j])&&(j > -1);j--);

if(i < j) {

//change

temp = array[i];

array[i] = array[j];

array[j] = temp;

}

else{

break;

}

}

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

array[i-1] = v;

return i - 1;

}

void quicksort(int p,int q){

int stack[MAX];

int top = 0;

int j;

int n;

for(;;) {

while(p < q){

j = partition(p,q);

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

printf("%d\t",array[n]);

}

printf("\n");

if((j - p) < (q - j)){

stack[top ++] = j + 1;

//栈溢出检测

if(top == MAX){

printf("Overflow\n");

exit(-1);

}

stack[top ++] = q;

//栈溢出检测

if(top == MAX){

printf("Overflow\n");

exit(-1);

}

q = j - 1;

}

else{

stack[top ++] = p;

//栈溢出检测

if(top == MAX){

printf("Overflow\n");

exit(-1);

}

stack[top ++] = j - 1;

//栈溢出检测

if(top == MAX){

printf("Overflow\n");

exit(-1);

}

p = j + 1;

}

}

if(top == 0) return;

q = stack[--top];

p = stack[--top];

}

return ;

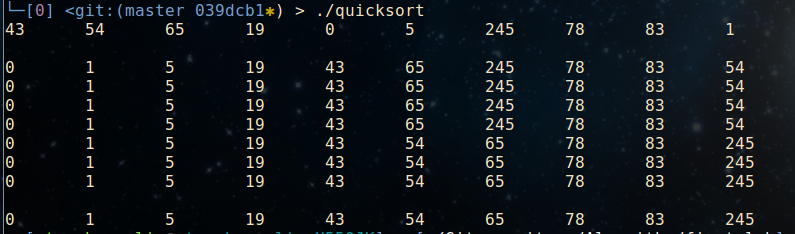
}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

(2)对源程序编译、链接、生成可执行程序，运行可执行程序，观察结果。

6.实验结果

运行quicksort，对写在程序中的数组进行排序



**图1-2 quicksort算法实验结果截图**

如图所示，第一行打印当前未排序的一串数据大小顺序随机，第二部分打印出一趟排序后的元素状态，最后一行打印出排序完成后的元素顺序。

通过对比分析 ，可以发现，无须的数组元素经过排序后变成非降序排列，并且大小顺序正确。则经过测试，实验正确。

7.结果分析

快速分类算法的最坏情况时间是O(n^2)而平均情况下时间是O(nlogn)。

现在来考虑递归所需的栈空间，在最坏情况下，递归的最大深度可以达到n-1,因此所需栈空间是O(n)。

实验二 SELECT2

1.实验题目

编程实现算法4.17，基于二次取中的选择算法，编制以下过程:

PARTITION

INSERTIONSORT

INTERCHANGE

SELECT2

2.实验目的与要求

(1)栈可以用数组实现，但要有溢出检测

(2)编制测试数据，给出实验结果

3.算法设计

SELECT2的SPARCK描述：

line procedure SEL(A,m,p,k)

//返回一个i，使得m<=i<=p,且A(i)是A(m:p)中第k小的元素，r是一个全局变量，其取值是大于1的整数。

gloable r

integer n,I,j

if p-m+1 <= r then call INSERTIONSORT(A,m,p)

return m-k+1

endif

loop

n = p-m+1 //元素数

for i=1 to n/r do //计算中间值

call INSERTIONSORT (A,m+(i-1)\*r,m + i\*r-1)

//将中间值收集到A(m:p)的前部

call INSERTIONSORT(A(m+i-1),A(m+(i-1)\*r + r/2-1)

repeat

j = SEL(A,m,m+n/r-1,n/r/2的上界)

call INTERCHANGE(A(m),A(j))

j = p +1

call PARTION(m,j)

case

:j-m+1=k :return

:j-m+1>k: p=j-1

:else:k=k-(j-m+1);m=j+1

endcase

repeat

end SEL

4.实验环境

操作系统：Ubuntu 16.04 编译器：gcc 编辑器：vim 调试器：gdb 采用c语言实现。

5.实验过程

(1)依照算法设计，采用c语言实现算法，代码实现如下：

【select2.c】

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#include<stdio.h>

#define NUM 9 //进行Insertionsort 结束的下标

#define MUM 0 // 进行Insertionsort 开始的下标

#define r 3

void insertionsort(int \*array,int m,int n);

int select2(int \*array,int m,int p,int k);

int partition(int \*array,int m,int p);

int main(){

int i;

int array[NUM-MUM+1] = {5,4,6,2,1,9,8,0,3,7};

int rt\_num;

//打印排序前的数组

for(i = 0;i < NUM - MUM + 1;i ++){

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

}

printf("\n\n");

// insertionsort(array,MUM,NUM);

rt\_num = select2(array,MUM,NUM,5);

printf("%d\n\n",array[rt\_num]);

for(i = 0;i < NUM - MUM + 1;i ++){

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

}

printf("\n");

return 0;

}

void insertionsort(int \*array,int m,int n){

int i,j;

int temp;

for(j = m+1;j <= n;j ++){

temp = array[j];

i = j - 1;

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

array[i + 1] = array[i];

i = i - 1;

}

array[i + 1] = temp;

}

}

int select2(int \*array,int m,int p,int k){

int n,i,j;

int temp;

int rs;

if((p - m + 1 ) <= r){

insertionsort(array,m,p);

return m + k - 1;

}

for(;;){

n = p - m + 1;

for(i = 1;i <= n/r; i++){//分为n/r组

insertionsort(array,m + (i - 1) \* r,m + i \* r - 1);

//将中间值收集到array的前部

temp = array[m + i - 1] ;

array[m + i - 1] = array[m + (i - 1) \* r + r/2 ] ;

array[m + (i - 1) \* r + r/2] = temp;

}

//计算(n/r)/2的上界

rs = (n/r)%2;

if(rs) rs = (n/r)/2 + 1;

else rs = (n/r)/2;

//得到j

j = select2(array,m,m + n/r - 1,rs);

//交换Ai Aj

temp = array[m];

array[m] = array[j];

array[j] = temp;

j = p;

partition(array,m,j);

if((j - m + 1) == k) return j;

else if((j - m + 1) > k) p = j - 1;

else {

k = k-(j - m + 1);

m = j + 1;

}

}

}

int partition(int \*array,int m,int p){

int v,i,j;

i = m + 1;

j = p;

v = array[m];

int temp;

for(;;){

for(;(v >= array[i])&&(i <= p);i ++);

for(;(v <= array[j])&&(j > -1);j --);

if(i < j){

temp = array[i];

array[i] = array[j];

array[j] = temp;

}

else{

break;

}

}

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

array[i-1] = v;

return 0;

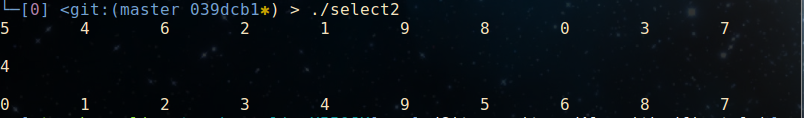
}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

(2)对源程序编译、链接、生成可执行程序，运行可执行程序，观察结果.

6.实验结果

运行select2，观察运行结果



**图2-1 select2算法实验结果截图**

如图所示，程序第一行打印出排序前的数组元素状态，处于无序状态，大小位置关系随机，第二行打印出要进行选择出的值，即第k小的元素，本次测试用例中，k为5，第三行打印出排序后的数组元素状态，通过对比发现，第5小的元素就是4，与实验结果一致，实验测试通过。

7.结果分析

r=5时，T(n)<=T(n/5)+T(3n/4)+cn，对于n<=24，T(n)显然是一线性时间，只要将上式中的c取得足够大，这线性时间就可以表示成

T(n)<=cn

至此，用归纳法证明对于n>=1，有

T(n)<=20cn

这一结论表明，在r=5的情况下，求解n个不同元素选择问题的散发SELECT2在最坏情况下的时间是O(n)。

SELECT2所需要的附加空间除了几个简单变量所需要的空间外，还需要作为递归的栈用的空间。递归的最大深度是logn,所以需要O(logn)的栈空间。

实验三 SHORTEST-PATHS

1.实验题目

(1)算法5.10(Dijkstra) SHORTEST-PATHS求出了v0至其它各结点 的最短路径，但是没有给出这些最短路径。

(2)补充算法5.10，使新算法在找出这些最短路径长度的同时， 也能求出路径上的结点序列。

2.实验目的与要求

1. 给出新算法的描述
2. 编写该算法的程序
3. 用书上的实例（或自行设计测试数据）测试程序，输出测试 结果。基本形式如下： start end length nodes list

v1 v2 20 v1v2

v1 v4 30 v1v4

v1 v6 80 v1 v2v3v5v6

3.算法设计

SHORT-PATH的SPARCK描述：

procedure SHORT-PATH(v,COST,DIST,n)

//G是一个n节点有向图，它由其他成本邻接矩阵COST(N,N)表示，DIST(j)被置以节点v到节点j的最短路径长度，这里1<=j<=n，DIST(v)被置0,node数组存储每次所选择的节点，用于输出路径，初始值为空//

boolean S(1:n);real COST(1:n,1:n),DIST(1:n)

integer u,v,n,num,I,w

for i=1 to n do //初始化集合//

S(i)=0;DIST(i)=COST(v,i)

repeat

S(v)=1;DIST(v)=0 //节点v计入S//

for num=2 to n-1 do //确定由节点v出发的n-1条路线

选取节点u，使得DIST(u)=min{dist(w)}

将u存入node数组

S(u)=1

for 所有的S(w)=0的节点 w do

DIST(w)=min{DIST(w),DIST(w)+COST(u,w)}

repeat

repeat

按格式打印出node的值

end SHORT-PATH

4.实验环境

操作系统：Ubuntu 16.04 编译器：gcc 编辑器：vim 调试器：gdb 采用c语言实现。

5.实验过程

(1)依照算法设计，采用c语言实现算法，代码实现如下：

【Dijkstra.c】

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#include<stdio.h>

#include<stdlib.h>

#include<stdbool.h>

//assume 0,1,2,3,4...

#define M 7

int min1(int \*array1,\_Bool \*array2);

int min2(int a,int b);

void print(int \*array);

int main(){

bool S[M];

//用于测试的临接矩阵

int COST[M][M] = {{0 ,20 ,50 ,30 ,1000,1000,1000},

{1000,0 ,25 ,1000,1000,70 ,1000},

{1000,1000,0 ,40 ,25 ,50 ,1000},

{1000,1000,1000,0 ,55 ,1000,1000},

{1000,1000,1000,1000,0 ,10 ,70 },

{1000,1000,1000,1000,1000,0 ,50 },

{1000,1000,1000,1000,1000,1000,0 },

};

int DIST[M]; //存储边的权

int node[M-1]; //存储经过的节点

int u,v,i,w,num;

v = 0;

num = 0;

//对DIST进行初始化

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

S[i] = 0;

DIST[i] = COST[v][i];

}

S[v] = 1; //设置第一个节点已经查看的节点

node[num++] = v + 1; //将第一个节点存储

for(i = 0; i < M;i ++){//计算导其他每个节点的最短长度

print(DIST);

u = min1(DIST,S); //选取到v最近的节点

node[num++] = u + 1;

for(w = 0;w < M;w++){

if(!S[w]){

DIST[w] = min2(DIST[w],DIST[u] + COST[u][w]);

}

}

}

node[num] = M;

print(node);

printf("start end length node list\n");

printf("V%-9dV%-9d%-10dV%d-V%d\n",node[0],node[M-1],DIST[M-1],node[0],node[M-1]);

return 0;

}

int min1(int \*array1,\_Bool \*array2){

int res;

int i;

int lable;

res = 1000;

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

if((res > array1[i])&&(!array2[i])) {

res = array1[i];

lable = i;

}

}

array2[lable] = 1;

return lable;

}

int min2(int a,int b){

return a<b?a:b;

}

void print(int \*array){

int i;

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

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

}

printf("\n\n");

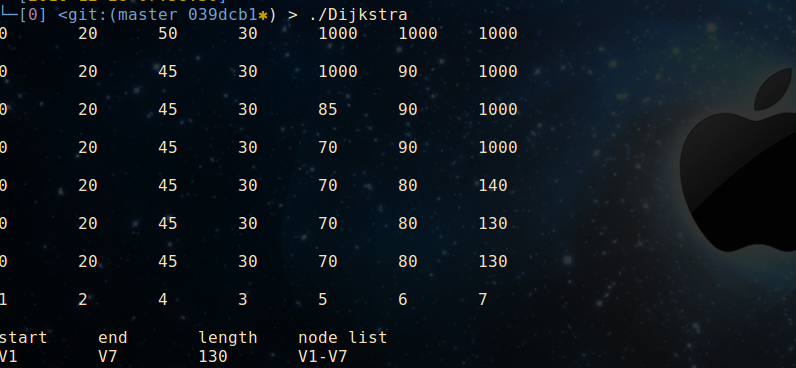
}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

(2)对源程序编译、链接、生成可执行程序，运行可执行程序，观察结果.

6.实验结果

运行Dijkstra，观察实验结果



**图3-1 Dijkstra算法实验结果截图**

如图所示，程序第一部分打印出每次选择后的节点路径的状态，每次更新后的节点路径如图，第二部分打印出节点序号，第三部分打印出对应路径的起点和终点，路径长度以及相应的路径。通过途中结果和分析理论结果对比，可以发现实验结果正确，实验测试成功。

7.结果分析

在n节点图上，算法所花费的时间是O(n^2)。第一行的for循环需要O(n)的时间，第5行的for循环则需要执行n-2次，而这个循环的每一次执行在第6行选择下一个节点并在第8~10行再一次修改DIS的值需要O(n)的时间，因此这个算法需要的总时间是O(n^2)。

附件实验 NQUEENS

1.实验题目

(1)算法8.5 NQUEENS求出了某一个n对应的n皇后问题的解，改进该算法，搜索有n皇后解得n 。

2.实验目的与要求

1. 给出新算法的描述
2. 编写该算法的程序
3. 用书上的实例（或自行设计测试数据）测试程序，输出测试 结果。

3.算法设计

PLACE用来判断此处能否放置一个皇后：

PLACE的SPARCK描述：

procedure PLACE(k)

//如果一个皇后能被放置在第k行和X(k)列，则返回true，否则返回false。X是一个全局数组，进入此过程时已置了k个值。ABS(r)返回r的绝对值 //

gloabae X(1:k);integer I,k

I = 1

while i<k do

if(X(i)=X(k) //在同一列有两个皇后//

or ABS(X(i)-X(k))=ABS(i-k) //在同一对角线有两个皇后

endif

I = I +1

repeat

return true

end PLACE

使用过程PLACE来实现n皇后的回溯算法：

NQUEENS的SPARCK描述：

procedure NQUEENS(n)

//此过程使用回溯算法求出在一个n\*n的棋盘上放置n和皇后，使其不能相互攻击的所有可能位置//

integer i,k,m,X(1:n)

输入n

循环，从1->n,执行以下过程

X(1)=0;k=1; //k是当前行，X(k)是当前列//

while K >0 do //对所有的行执行以下语句//

X(K)=X(K)+1 //移到下一列//

while X(k)<=n and not PLACE(k) do //判断此处能否放置这个皇后//

X(k) = X(k) +1

repeat

if X(k)<=n then //找到一个位置//

if k = n //判断是否是一个完整的j解

then print(X) //是就打印出这个解//

elsek = k +1;X(k)=0 //转到下一行//

endif

else k = k-1 //回溯//

end if

repeat

en NQUEENS

4.实验环境

操作系统：Ubuntu 16.04 编译器：gcc 编辑器：vim 调试器：gdb 采用c语言实现。

5.实验过程

(1)依照算法设计，采用c语言实现算法，代码实现如下：

【Nqueens.c】

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#include<stdlib.h>

#include<math.h>

#include<stdio.h>

#define M 10

int place(int \*queen,int n);

void nqueens(int \*queen,int n);

void print(int \*queen,int n);

int num;

int main(){

int queue[M] ;

int n;

int i;

scanf("%d",&n);

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

num = 0;

nqueens(queue,i);

printf("The total selution for %d queens problem is :%d\n",i,num);

}

return 0;

}

int place(int \*queen,int n){

int i;

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

if((queen[i] == queen[n])||(abs(queen[i]-queen[n]) == abs(i - n)))

return 0;

}

return 1;

}

void nqueens(int \*queen,int n){

int k;

queen[0] = -1;

k = 0;

while(k>=0){

queen[k] = queen[k] + 1;

while((queen[k] < n)&&(!place(queen,k))) {

queen[k] = queen[k] +1;

}

if(queen[k] < n){

if(k == n - 1){

print(queen,n);

num ++;

}

else{

k = k + 1;

queen[k] = -1;

}

}

else{

k = k - 1;

}

}

}

void print(int \*queen,int n){

int i;

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

printf("%d\t",queen[i]);

}

printf("\n");

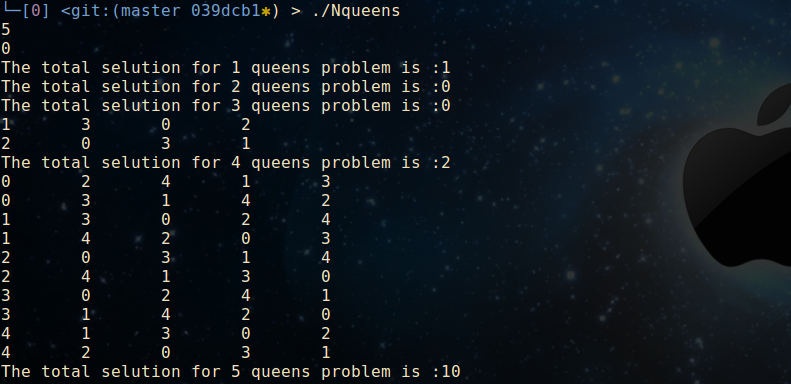
}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

(2)对源程序编译、链接、生成可执行程序，运行可执行程序，观察结果.

6.实验结果

运行Nqueens，观察实验结果



**图4-1 Nqueens算法实验结果截图**

如图所示，程序接受的输入是5，则依次打印出1~5的皇后问题解，n=1是一个解，n=2，n=3时没有解，n=4时2个解，n=5时10个解，通过对应n的解去验证，发现结果正确并且n皇后问题的解得个数与预期相符，则实验结果正确。

7.结果分析

过程PLACE返回一个布尔值，当第k个皇后能放置在X(k)的当前值时，返回职位真，否则为false，这个过程测试两种情况，即皇后在同一列或在对角线上。该过程的计算时间是O(k-1)。

过程NQUEENS只允许把皇后放置在不同的行和列上，因此之多需要作n!次检查。