实验报告

题目：求约瑟夫环问题的出列顺序

班级：信卓1901班 姓名：刘可立 学号：U201913498

一、需求分析

1. 模拟整个约瑟夫问题的过程，依次输出出列序号。数据密码m可以任意，作为报数的上限值。m由用户输入，每个人的密码在1~m中随机产生，不事先存储。 (若m为负则取m mod n)
2. 程序运行时，从用户输入获取报数上限值m、总人数n，并在完成输出出列序号后，输出运行的时间。
3. 用链表和顺序存储两种方式实现，并分别分析两种途径的时间复杂度。
4. 用大量的数据（n万级别）运行程序，分别得到各自的运行时间，画出时间和人数n的曲线，拟合出函数f(n)。
5. 测试数据：m的初值为20，n = 7, 七个人的密码依次为：3, 1, 7, 2, 4, 8, 4。此时出列顺序应为6, 1, 4, 7, 2, 3, 5。

二、概要设计

为实现上述程序功能，分别使用链式存储和顺序存储的方法。

1. 所使用的线性表的抽象数据结构定义为：

ADT LinearList {

数据对象：D = {ai | ai ∈ ElemSet, i = 1, 2, ..., n, n>=0}

数据关系：R = {< ai-1 , ai > | ai-1 , ai ∈ D, i = 2, ..., n}

基本操作：

Init(&L)

构造一个线性表并对其初始化

DestroyList(&L)

L已存在

销毁线性表L，释放分配的空间

ListEmpty(L)

若L为空表，返回True，否则返回False

ListLength(L)

返回L中数据元素个数

GetElem(L, i, &e)

线性表L存在，1<=i<=ListLength(L)

将L第i个元素值赋给e

ListDelete(&L, i, &e)

线性表L存在，1<=i<=ListLength(L)

删除L的第i个数据，将其值赋给e，L长度减1

}

1. 本程序包含四个模块：
2. 主程序模块：

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

初始化数据;

接受用户输入;

while (还有人未出列){

处理命令;

输出出列序号;

}

输出运行时间;

}

1. 实现顺序结构线性表的抽象数据类型
2. 实现链式结构线性表的抽象数据类型
3. 链式结构线性表的结点结构

三、详细设计

1. 采用顺序结构的线性表：记录剩余人数，存储每一个人的密码，并增加一个数据域存储每个人的序号。定义如下：

class LinearList {

public:

int \* member; //人员序号

int \* data; //密码

int last;

LinearList(int n): last(n - 1){

对member, data分配相应的存储空间并初始化

}

~LinearList(){

释放分配的存储空间

}

int & operator[](int n){

返回data[n]的引用

}

void Delete(int n){

合法性判断

删除member[i]和data[i],并将其后的元素前移

last--

}

};

1. 采用链式结构的线性表：存储头指针。采用双向链表实现，耗费空间略多，但代码更加直观，操作比较方便。每个节点有两个数据域和两个指针域，分别存储序号、密码、前驱、后继。类定义如下：

class LNode {

public:

int idx;

int data;

LNode \* prior;

LNode \* next;

LNode(int n): idx(n){

对data随机赋值

}

};

class LinkedList {

public:

LNode \* head;

LinkedList(int n){

分配第一个结点的存储空间

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

分配下一个结点的存储空间

设置prior, next

}

}

void Delete(LNode \* p){

删除结点并输出其序号

}

};

1. 主函数的伪代码算法

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

初始化

获取用户输入m, n,并由用户指定顺序存储或链式存储

构造约瑟夫环(自动调用构造函数)

开始计时

do {

计算下一个出列序号

Delete(i);

更新密码

} while (还有人未出列);

计时结束

输出时间间隔

return 0;

}

1. 函数调用关系

在主函数中创建类的对象，执行类的各个成员函数

四、调试分析

1. 设计过程总结反思

本程序设计过程总体上比较合理。但实现方法为暴力算法，时间复杂度较高。编程过程中，发现使用new运算符分配大量存储空间时容易出错，改用malloc()函数后问题解决，但未找到具体原因。

1. 算法的时空分析
2. 空间复杂度

算法在原地工作，使用的储存空间为常数。

1. 时间复杂度

采用实验验证。用大量的数据（n万级别）运行程序，分别得到各自的运行时间。

1. 预计结果：

进行理论分析，若采用顺序结构存储，共n人需出列，每出列一人，需要将他之后的所有人前移一位，总共时间复杂度应为O(n2)。若采用链式存储，共需n人出列，删除一个结点的时间复杂度为常数，但由于链式存储无法随机读取，寻找下一个出列对象需要一定时间，这个时间应与m、n有关。

1. 实验过程：

对顺序存储结构：

顺序结构支持随机存取，根据之前的分析，时间复杂度应只与n有关。先将m定为500，取不同的n值进行实验，并用Excel拟合出曲线。

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| n(\*104) | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| f(n) | 0.09 | 0.36 | 0.81 | 1.43 | 2.17 | 3.11 | 4.28 | 5.55 | 7.01 | 8.68 |



得到y = 0.0908x1.9787，R² = 1。可见，拟合度极好，时间复杂度为O(n2)。为验证此时的时间复杂度与m无关，再将m改为100000进行测试

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| n(\*104) | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| f(n) | 0.10 | 0.37 | 0.80 | 1.41 | 2.16 | 3.09 | 4.19 | 5.52 | 7.14 | 8.65 |

地图的截图

描述已自动生成

耗时与m = 500时接近，得到了相同的结果：时间复杂度为O(n2)。

对链式存储结构：

在寻找下一个出列对象时，需要执行p = p->next的次数与m有关。可以猜想，当m远小于n时，时间复杂度应随m而增大。

取n为40000，取不同的m进行实验

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| m | 100 | 300 | 500 | 800 | 1000 | 1500 | 2000 | 3000 |
| f | 0.17 | 0.29 | 0.40 | 0.58 | 0.68 | 1.01 | 1.23 | 1.85 |



外层循环次数为n，毫无疑问此时的时间复杂度与n呈正相关，所以此时的时间复杂度f(m, n) = O(mn)。

而当m远大于n时，如n取40000，当m取100000，耗时为17.88s，当m取1000000，耗时为17.67s，二者比较接近。这是因为p = p->next执行次数为m % n，与m本身的数值关系不大，因此十万级的m和百万级的m对结果并无太大影响。

因此，为探究时间复杂度与n的关系，可使m远大于n。令m取106，再取不同的n进行实验。

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| n(\*104) | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| f(n) | 0.14 | 0.55 | 1.24 | 2.31 | 3.58 | 5.31 | 7.51 | 9.99 |

地图的截图

描述已自动生成

可见，m远大于n时，链式存储结构方案的f(n) = O(n2)。

1. 根据以上结果，当报数上限m远大于n时，两种方案的时间复杂度均为O(n2)，且顺序存储略快于链式存储。但是当报数上限m远小于n时，使用链表更加节省时间。

五、用户手册

1. 需要用户输入三个参数，m 作为报数上限, n 为人数, type 为0或1分别对应顺序存储和链式存储
2. 在 Terminal 中定位到 a.out 所在的文件夹
3. 可直接运行 ./a.out
4. 执行 ./a.out -h 可查看帮助
5. 执行 ./a.out m n type 来运行。示例:手机屏幕截图

   描述已自动生成

(若Terminal窗口太小可能导致提示文字显示不正确)

1. 若n大于100，结果将仅输出于同目录下的sequence.data文件。

六、测试结果

1. m = 20, n = 7

输出：

The codes are: 4 9 1 18 20 16 5

The sequence is: 6 3 4 7 2 5 1

duration: 0.00s

1. m = 100, n = 20

The codes are: 28 57 89 65 63 98 20 22 83 3 78 36 35 24 9 16 47 90 55 98

The sequence is: 20 3 1 13 16 17 5 19 10 14 4 7 12 15 6 11 2 18 9 8

duration: 0.00s

1. m = 100000 n = 10000

输出见example.data

七、附录

清单：

/JosephProblem.xcodeproj 项目文件

/JosephProblem/main.cpp 主程序

/JosephProblem/Header.h 头文件

/JosephProblem/a.out 可执行文件

/JosephProblem/example.data 测试输出