
《约瑟夫问题的线性表模拟》

实验报告

小组成员

姓名：刘子彬

学号：2020212901

分工：顺序表 测试

姓名：赵昕鹏

学号：2020211935

分工：链式表 文档

姓名：严怡彬

学号：2020210459

分工：可视化 文档

2020 年 10 月 25 日

目录

《约瑟夫问题的线性表模拟》实验报告	1
1 问题描述	1
2 设计描述	1
3 测试设计	7
4 心得体会	12
5 模拟的可视化实现	14
6 提交材料说明	18

《约瑟夫问题的线性表模拟》实验报告

1 问题描述

1.1 任务描述

约瑟夫问题：设有 N 个数字有序排列（即 1 到 N ）构成一个头尾相接的环，从位置 1 开始，每次从这个圆圈里删除第 M 个位置的数字，删除后从下一个位置开始计数，继续删除从这个位置开始的第 M 个位置的数字，如此循环直至环中数字全部删除结束。

分别使用顺序存储的线性表和链式存储的线性表来模拟上述约瑟夫问题。

模拟约瑟夫的过程中，将环中被删除的数字依次保存在另外一个线性表中，模拟约瑟夫问题过程结束后，将全部被删除数据元素一次性全部输出。

1.2 输入描述

两个整数值，第一个大于 0 的整数为 N ，第二个大于 0 的整数为 M

1.3 输出描述

顺序存储的线性表，输出由<删除数字，移动次数>两个数字组成的删除序列；

链式存储的线性表，输出由<删除数字，指针移动次数>两个数字组成的删除序列。

1.4 程序功能

程序通过输入的 N ， M 分别给出约瑟夫环在顺序存储和链式存储的线性表模拟下的情况，并将删除与移动的结果使用文件保存。

2 设计描述

2.1 算法思路

根据实验要求，该项目分别完成了按顺序存储的线性表和按链式存储的线性表模拟约瑟夫问题。

顺序表存储的线性表实现模拟：如 2.2 数据结构定义图中所示，我们使用了结构体定义了用数组实现的线性表，它的元素有大小 **Size** 和数据指针 **Data**，并有线性表的基本函数包括初始化、销毁、插入、获取大小、是否为空以及删除元素。为了实现本题的要求，使用了函数 **MoveByMovement**，实现将列表指针指向待删除元素的功能。

（关于增加的函数功能与实现在 2.5 算法的特殊说明中有详细的解释。）当读取完输入后，线性表完成初始化，并根据 M 的步进开始删除元素，并将删除的元素写入到 **out** 文件中保存，直到删除完线性表中所有元素，此时约瑟夫问题模拟完成，进行线性表的销毁，达到了实验的目的。

顺序表存储的线性表实现模拟：如 2.2 数据结构定义图中所示，我们使用了结构体分别定义了链表节点和用链表实现的线性表。线性表有元素尾指针 Tail 有游标指针 Cur，用 ListLength 保存线性表长度，并有线性表的基本函数包括初始化、销毁、尾插入、获取大小。为了实现本题的要求，增加了函数 DeleteNextElement 和 DeleteNextMthElement，DeleteNextMthElement 用以完成将游标指针 Cur 移动至指向后第 M-1 个元素，并且在内部调用 DeleteNextElement 完成删除第 M 个元素的功能。（关于增加的函数功能与实现在 2.5 算法的特殊说明中有详细的解释。）当读取完输入后，线性表完成初始化，并根据 M 的步进开始删除元素，并将删除的元素写入到 out 文件中保存，直到删除完线性表中所有元素，此时约瑟夫问题模拟完成，进行线性表的销毁，达到了实验的目的。

2.2 数据结构定义

顺序存储的线性表数据结构定义

```
struct ListByArray
{
    int Size;
    int *Data;
    bool InitList(int InitSize);
    bool DestroyList();
    bool Insert(int x);
    bool PrintListForDebug();
    bool MoveByMovement(int* NowPos, int Movement);
    bool GetSize(int* NowSize);
    bool IsEmpty();
    bool DeleteElement(int NowPos, ResultPackage* NowPackage);
};
```

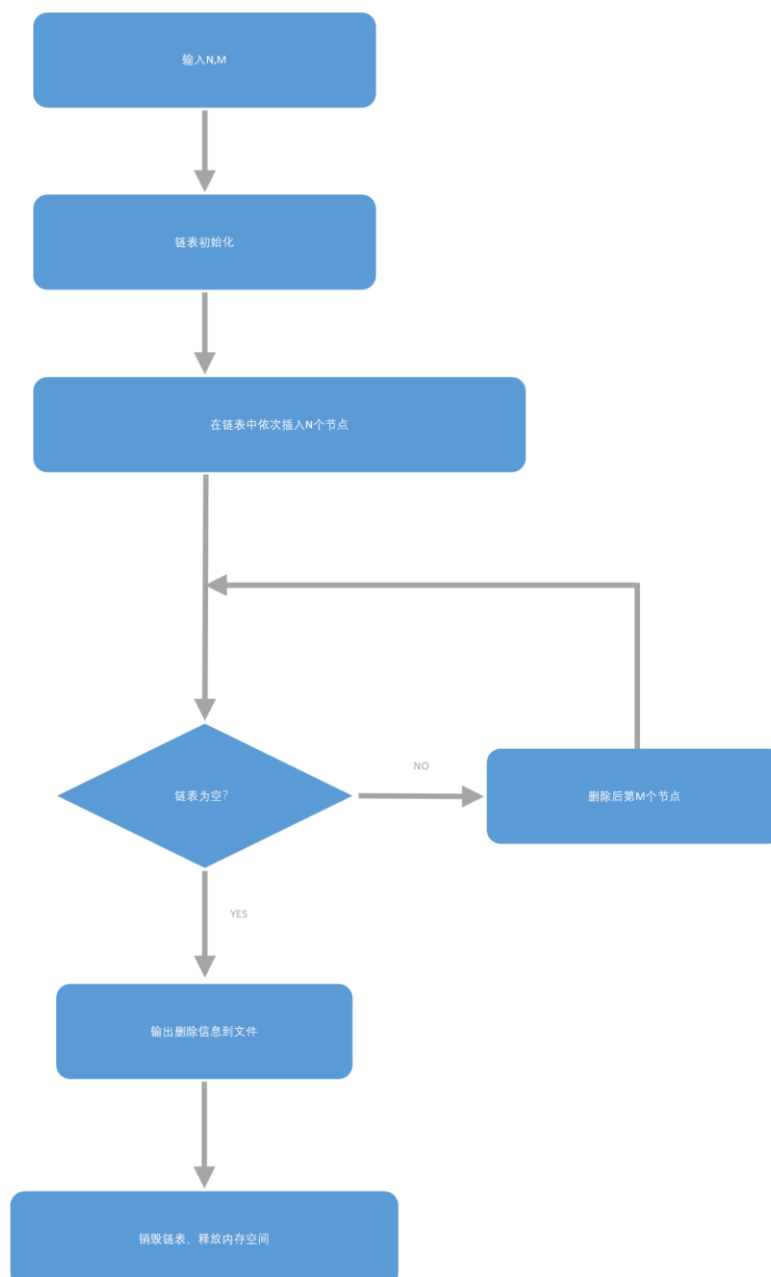
链式存储的线性表数据结构定义

```
struct ListNode
{
    int Data;
    ListNode *Next;
};

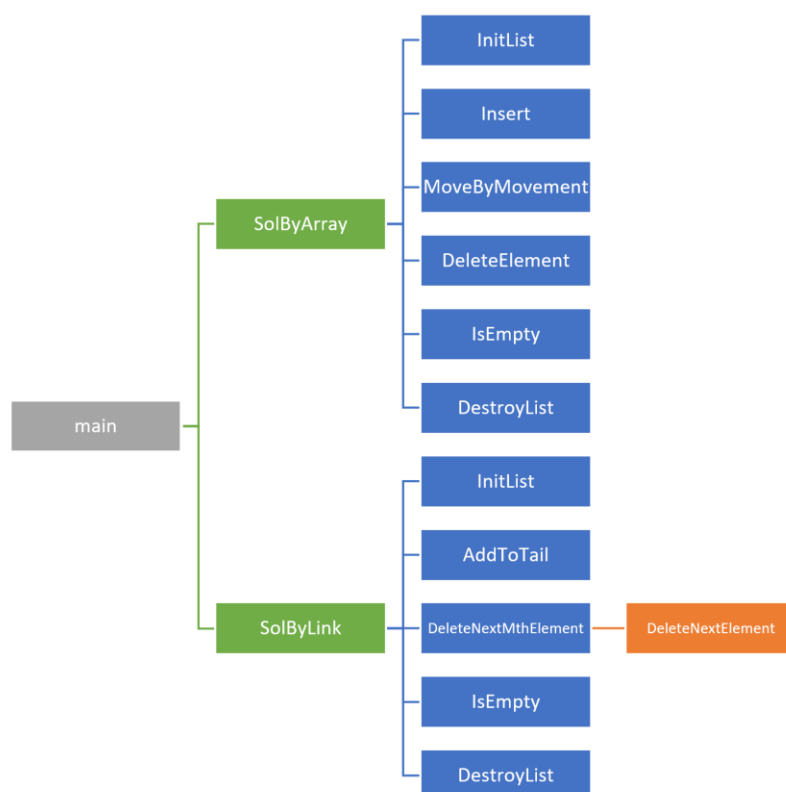
bool ListNodeInit(ListNode **x);
struct ListByLink
{
    ListNode *Tail, *Cur;
    int ListLength;
    bool InitList();
};
```

```
bool DestroyList();
bool AddToTail(int x);
bool DeleteNextElement(ResultPackage *Result);
bool DeleteNextMthElement(int m, ResultPackage *Result);
bool PrintListForDebug();
int GetSize();
bool IsEmpty();
};
```

2.3 核心程序的处理流程



2.4 各模块的调用关系



2.5 需要特别说明的算法描述

2.5.1 本项目自定义了一个数据类型 **ResultPackage**，内部有 **Pos** 和 **Cost** 两个 **int** 型元素，分别对应了输出要求的删除元素和移动元素（指针）个数，代码实现如下图。

```

struct ResultPackage
{
    int Pos, Cost;
};
struct ResultList
{
    int Size;
    ResultPackage *Data;
    bool InitList(int InitSize);
    bool DestroyList();
    bool Insert(ResultPackage x);
    bool PrintList(FILE* fout);
};
  
```

2.5.2 本项目实现顺序表模拟采用的函数 `MoveByMovement`，该函数内将当前位置加上步进大小 `M` 对 `Size` 取模，实现了将顺序表视作环形线性表的功能，代码实现如下图。

```
bool ListByArray::MoveByMovement(int* NowPos, int Movement)
{
    (*NowPos) = ((*NowPos) + Movement) % Size;
    return true;
}
```

2.5.3 本项目实现链表模拟采用的函数 `ListNodeInit`，该函数的传入参数为一个指向 `ListNode` 类型指针的指针，目的是给指向 `ListNode` 类型的指针申请空间，已完成 `ListNode` 的初始化，代码实现如下图。

```
bool ListNodeInit(ListNode **x)
{
    *x = (ListNode*) malloc(sizeof(ListNode));
    if(x == NULL)
        return false;
    (*x) -> Next = NULL;
    return true;
}
```

2.5.4 本项目实现链表模拟采用的函数 `DeleteNextElement`，该函数的传入参数为 2.5.1 中描述的 `ResultPackage` 指针，如果当前位置或当前位置的 `next` 为 `NULL` 则返回 `false`，否则执行线性表删除下一位元素的工作，因为线性表初始化时将尾元素的 `next` 指向为尾元素本身，因此若删除的元素为最后一位，即 `tail` 指向的元素时，新的 `Tail` 即为 `Cur` 游标指针当前指向的元素，并将本次移动的 `ResultPackage` 写入参数指向的地址中，代码实现如下图。

```
bool ListByLink::DeleteNextElement(ResultPackage *Result)
{
    if(Cur == NULL)
        return false;
    if(Cur -> Next == NULL)
        return false;
    ListLength--;
    ListNode* Tmp = Cur -> Next;
```

```
if(Tmp == Tail)
{
    Tail = Cur;
    if(Cur == Tmp)
        Tail = NULL;
}
Result -> Pos = Tmp -> Data;
Tmp = Tmp -> Next;
free(Cur -> Next);
Cur -> Next = Tmp;
return true;
}
```

2.5.5 本项目实现链表模拟采用的函数 `DeleteNextMthElement`，该函数的传入参数为 2.5.1 中描述的 `ResultPackage` 指针，在函数内实现了将游标指针移动到当前位置后第 $M-1$ 的位置，并调用 2.5.4 中描述的函数 `DeleteNextElement`，删除下一位即第 M 位元素，最后将本次删除的元素位置和指针移动次数传入 `ResultPackage` 所指向的地址中，代码实现如下图。

```
bool ListByLink::DeleteNextMthElement(int m, ResultPackage* Result)
{
    if(Cur == NULL || ListLength == 0)
        return false;
    Result -> Cost = (m - 1) % ListLength;
    for(int i = 1; i <= Result -> Cost; i++)
    {
        if(Cur -> Next == NULL)
            return false;
        Cur = Cur -> Next;
        TailAdd(Demo, Cur->Data);
    }
    if(!DeleteNextElement(Result))
        return false;
    TailAdd(Demo, Result->Pos);
    return true;
}
```


3 测试设计

3.1 测试用例输出说明

在测试用例设计中，由于本算法在 m 固定时，删除元素具有很强的前置性，故输出只分为两种：

- (1) **ERROR!** 表示输入数据为非法数据或程序运行失败。
- (2) 一个正整数。表示最后一个被删除的数字。

3.1.1 测试用例 1

输入：-1 -1

输出：ERROR!

用例设计意图：测试程序能否对非法输入做出判断，提高程序鲁棒性。

3.1.2 测试用例 2

输入：10 1

输出：10

用例设计意图：测试程序在 $m=1$ 的极端环境下，（即不论是数组还是链表均不会发生 move 操作就直接进行删除操作）能否正常输出结果。

3.1.3 测试用例 3:

输入：10 10

输出：8

用例设计意图：当第一次删除时，程序会删除最后一个元素，测试程序在删除元素为极端远的时候，程序能否正确输出结果。且当第二轮删除时， $n < m$ ，也能测试程序在遍历完所有元素后能否继续进行删除操作。

3.1.4 测试用例 4:

输入：2000 1692

输出：86

用例设计意图：由于本程序单次删除复杂度均为 $O(\text{Size})$ 级别（Size 表示剩余元素个数），故正常规模数据下，本程序的设计复杂度为 $O(n^2)$ 。设计本测试样例可以测试程序在极端数据下会不会出现内存溢出以及运行时间过程，爆栈等情况。

3.2 测试用例概述：上述 4 个测试用例从输入合法性，程序内存复杂度，时空复杂度符合设计思路， $m=1, m=n$ 的边界条件的极端环境下是否出现数组越界，指针出错等情况测试了程序，在测试用例较少的情况下，能有效覆盖所有测试场景。

3.3 附加测试用例说明：除在 source\TestData 下存放的 4 组测试数据之外，程序包还提供了多组随机测试数据生成的功能（程序内默认是生成 6 组随机数据），以真实模拟程序实际运行环境，看程序能否在更加普遍的输入下输出正确。其使用方法和说明将在下方开发调试环境说明里详细阐述。

3.4 开发调试环境说明：在程序包的 source 文件目录下存放着所有源代码，其中 GeneratorAndChecker.cpp 文件用于测试及调试，测试者只需要双击 GeneratorAndChecker.exe 即可开始测试。本程序一旦运行，会在\source 目录下的

debug.in 文件写入一个 1，此后 main.cpp 均会在 test.in 内读入输入，若要将其转为命令行输入模式，请在 debug.in 文件内写入一个整数 0。


其源代码如下：

```
#include<cstdio>
#include<ctime>
#include<cstdlib>
#include<windows.h>
#include<string>
using namespace std;
int GetLive(int Cur, int InitM)
{
    if(Cur == 1)return 1;
    return (GetLive(Cur - 1, InitM) + InitM - 1) % Cur + 1;
}
int main()
{
    srand(time(NULL));
    FILE *fdebug =fopen("debug.in", "w");
    fprintf(fdebug,"1\n");
    fclose(fdebug);
    string prefix1, prefix2, suffix1, suffix2, now;
    prefix1 = "TestData/Input",prefix2 = "test", suffix1 =
".in",suffix2 = ".out";
    for(int TestRound = 1; TestRound <= 4; TestRound++)
    {
        //此循环用来自动将我们设计好的 4 个测试用例(TestData/Input.in)载入
        test.in 以供 main.cpp 读取
        //并将运行结果与 TestData 文件夹下内置的 CorrectAnswer.out 调用 cmd 中
        fc 指令进行比较
        //若比较无误则会输出 Test xx Passed! 字样
        char tmp[20];
        int InitN, InitM;
        itoa(TestRound, tmp, 10);
        now = tmp;
        FILE *fin = fopen((prefix1 + now + suffix1).c_str(),
        "r");
        FILE *testin = fopen((prefix2 + suffix1).c_str(), "w");
        fscanf(fin, "%d%d", &InitN, &InitM);
        fprintf(testin, "%d %d\n", InitN, InitM);
        fclose(fin);
        fclose(testin);
    }
}
```

```
        system("g++ ListByLink.cpp ListByArray.cpp
ResultPackage.cpp main.cpp -o test");
        system("test");
        string prefix3 = "fc AnsForChecker.out
TestData/CorrectAnswer";
        if(system((prefix3 + now + suffix2).c_str()))
        {
            puts("ERROR!");
            return 0;
        } else
        {
            fprintf(stderr, "Test %d Passed!\n", TestRound);
        }
    }
    fprintf(stderr, "Fixed TestData(Test 1~4) Passed!\n");
    for(int TestRound = 5; TestRound <= 10 ; TestRound++)
    {
        //此循环将自动生成 6 组随机数据载入 test.in 以供 main.cpp 读取
        //并将运行结果与本程序中使用 GetLive 程序得到的答案 调用 cmd 中 fc 指令进行
        比较
        //若比较无误则会输出 Test xx Passed! 字样
        int InitN = rand() % 2000 +1, InitM = rand() % 5000 + 1;
        FILE *testin = fopen((prefix2 + suffix1).c_str(), "w");
        fprintf(testin, "%d %d\n", InitN, InitM);
        fclose(testin);
        FILE *testout = fopen("CorrectAnswer.out", "w");
        if(InitN > 0 && InitM > 0) fprintf(testout, "%d\n",
GetLive(InitN,InitM));
        else fprintf(testout, "ERROR!\n");
        fclose(testout);
        system("g++ ListByLink.cpp ListByArray.cpp
ResultPackage.cpp main.cpp -o test");
        system("test");
        if(system("fc AnsForChecker.out CorrectAnswer.out"))
        {
            puts("ERROR!");
            return 0;
        } else
        {
            fprintf(stderr, "Test %d Passed!\n", TestRound);
        }
    }
    fprintf(stderr, "Random TestData(Test 5~10) Passed!\n");
```

```
}
```

下附上程序成功运行截图。

 F:\lab_1\Source\GeneratorAndChecker.exe

```
正在比较文件 AnsForChecker.out 和 TESTDATA/CORRECTANSWER1.OUT
FC: 找不到差异

Test 1 Passed!
正在比较文件 AnsForChecker.out 和 TESTDATA/CORRECTANSWER2.OUT
FC: 找不到差异

Test 2 Passed!
正在比较文件 AnsForChecker.out 和 TESTDATA/CORRECTANSWER3.OUT
FC: 找不到差异

Test 3 Passed!
正在比较文件 AnsForChecker.out 和 TESTDATA/CORRECTANSWER4.OUT
FC: 找不到差异

Test 4 Passed!
Fixed TestData(Test 1~4) Passed!
正在比较文件 AnsForChecker.out 和 CORRECTANSWER.OUT
FC: 找不到差异

Test 5 Passed!
正在比较文件 AnsForChecker.out 和 CORRECTANSWER.OUT
FC: 找不到差异

Test 6 Passed!
正在比较文件 AnsForChecker.out 和 CORRECTANSWER.OUT
FC: 找不到差异

Test 7 Passed!
正在比较文件 AnsForChecker.out 和 CORRECTANSWER.OUT
FC: 找不到差异

Test 8 Passed!
正在比较文件 AnsForChecker.out 和 CORRECTANSWER.OUT
FC: 找不到差异

Test 9 Passed!
正在比较文件 AnsForChecker.out 和 CORRECTANSWER.OUT
FC: 找不到差异

Test 10 Passed!
Random TestData(Test 5~10) Passed!

-----
Process exited after 15.04 seconds with return value 0
请按任意键继续. . .
```

4 心得体会

(1) 对题目理解的统一是如何做到的？（形式：开会、IM 讨论、主要贡献人；如题目内容很明确，大家无歧义，可跳过此内容）

答：题目内容明确，大家无歧义。

(2) 对程序的算法和数据结构设计是如何做的？（形式：开会、IM 讨论等；主要负责人）

答：开会讨论主要设计，赵昕鹏与刘子彬同学负责主要算法设计，赵昕鹏同学负责链式存储线性表的设计，刘子彬同学负责顺序存储线性表设计。实际设计中保持线上交流，确保算法设计的兼容性与统一性，便于整合，同时在一台服务器上编写代码，便于调试。

(3) 对程序的输入输出和测试用例如何确定的？（形式：开会、IM 讨论等；主要负责人）

答：由刘子彬同学主要负责测试用例的设计，设计过程中线上与其他同学保持交流。针对问题的特点，有针对性地设计测试用例，保证覆盖所有情况的前提下用最少的测试数据完成测试，同时构造了一系列涉及边界处理、可能容易出错的特殊测试用例保证程序的可靠性。

(4) 调试过程中遇到的问题是如何解决；

答：调试过程中，对多文件编译出现问题，查找相关资料后解决。对顺序存储的线性表调试时未遇到问题。链式存储线性表调试过程中，出现运行时错误。检查代码定位问题，为 DestroyList 时 free 尾指针后并未将野指针赋为 NULL，导致出现问题。

(5) 算法的时空分析和改进设想；

答：对于顺序存储的线性表，空间复杂度 $O(n)$ ，取出当前位置后第 m 个元素的时间复杂度为 $O(1)$ ，每删除一个数据元素需要进行 $O(n)$ 次的移动，故总时间复杂度为 $O(n^2)$

对于链式存储的线性表，空间复杂度 $O(n)$ ，取出当前位置后第 m 个元素的时间复杂度为 $O(m)$ ，删除一个数据元素时间复杂度为 $O(1)$ ，故总时间复杂度为 $O(nm)$

改进设想：对于约瑟夫问题，可以使用线段树、平衡树等高级数据结构，单次寻找删除的负责度为 $O(\log n)$ ，总时间复杂度 $O(n \log n)$

(6) 小组成员自评：对自己在实验中长短处的评价（建议和小组组员讨论形成共识）；从实验中取得的经验等。

刘子彬：在本次实验中较好的发挥了自己擅长的代码编写能力和问题分析能力，出色完成了分配到的主程序实现任务，且在测试用例设计方面综合地考虑了程序性能的各方面因素，高效完成了测试用例的构造和后续随机化数据生成及测试模块的编写，且在实验中积极与队员交流，与组员达成了良好的团队协作，保证了代码风格的一致性以便阅读。虽在多文件链接，文件输入输出流代码实现的掌握还不够熟练，但经组内讨论顺利解决了问题。本次实验主要对工程文件的接口撰写原则，测试模块的搭建，顺序存储与链式存储的线性表的各项功能，以及团队分工意识的必要性有了更加充分的认识。

赵昕鹏：本次实验中，充分发挥了自己对 C++ 语法较熟悉、对代码编写较为熟练的优点，负责了链式存储结构线性表的设计与编写，完美完成了程序开发的任务，并进行了代码的调试工作，解决了实验中遇到的文件输入输出流等问题。同时我借助之前的工程代码开发经验，设计并确定了统一的代码风格与接口形式，在实验中与组员保持了较好的交流。在本次实验中，我进一步熟悉了线性表的实现、多文件编译与指针的使用，对于团队协作开发的流程以及团队协作的必要性有了更进一步的认识。

严怡彬：

在本次实验中，我通过与小组成员的交流协作，顺利地完成了本次实验。在本次实验的过程中，我认为我们协作分工比较明晰，由刘子彬负责顺序存储线性表的模拟实验和测试数据，赵昕鹏负责链式存储线性表的模拟实

验和心得整理，由我负责可视化实现和实验报告的说明与整理，较好地锻炼了我们的团队协作能力。此外，该实验还较好地调动了我们对两种不同的线性表的运用，有效地提升了我们对该知识点的掌握。我在本次实验中还使用了 `python` 中的 `pygame` 库，成功实现了程序的可视化。在这项工作中，非常好地锻炼了我使用 `python` 调用数据、整理数据、展示数据的能力。

不足：在实验后期的工作进度与团队的交流可以再优化一些，如更频繁地更新目前的工作进度、使用 `Notion` 等在线文档进行协调与分配任务等。

5 模拟的可视化实现

5.1 可视化的基础

5.1.1 交互文件定义

本项目可视化的实现使用 `python` 中的 `pygame` 库，在 `main.cpp` 中完成 `demo.csv` 的输出，其中 `demo.csv` 数据格式如下：

```
1. n,m
2. odder[0],cost[0]...odder[n-1],cost[n-1]
3. pointer[0],pointer[1],pointer[2]...
```

其中 `odder` 数组代表删除的节点顺序，`cost` 数组为删除当前节点的指针移动次数，`pointer` 数组记录指针指向的元素变化

5.1.2 核心代码程序段

```
while True:
    for event in pygame.event.get():
        if event.type == QUIT:
            exit()
        elif i == 0:
            waitforclick()
            break
    # 设置 FPS
    if i > 0 :
        DrawArc(screen, 300 + 150 * math.sin(rad * (Point[i-1] - 1)), 300 + 150 * math.cos(rad * (Point[i-1]-1)),
            (255, 255, 255), 6)
        clock.tick(2)
    try:
        pygame.draw.rect(screen, (255, 255, 255), (900, 400, 200, 50))
```



```

        DrawArc(screen, 300 + 150 * math.sin(rad *
(Point[i]-1)), 300 + 150 * math.cos(rad *(Point[i]-1)),
            (0, 0, 0), 5)
        text_point_value = GAME_FONT.render(str(Point[i]),
True, (0, 0, 0))
        screen.blit(text_point_value,(900,400))

        if Point[i] == Odder[i1] :
            disappear(screen,rad * (Odder[i1] - 1 ))
            pygame.draw.rect(screen, (255, 255, 255), (850,
450, 250, 50))
            text_cost_value =
GAME_FONT.render(str(Cost[i1]), True, (0, 0, 0))
            screen.blit(text_cost_value,(900,450))
            i1 = i1 + 1
            i = i + 1
        except IndexError:
            DrawArc(screen, 300 + 200 * math.sin(rad *
(Odder[i1] - 1)), 300 + 200 * math.cos(rad * (Odder[i1] -
1)),
                (255, 255, 255), 31)
            pygame.display.update()
            waitforclick()
            exit()
        pygame.display.update()

```

每过一个时钟周期，就更新指针的指向，并把变化的新指针使用 `pygame.draw()` 方法打印在屏幕上，若指针指向的元素与 `odder` 数组中删除的元素契合，则将这个元素在屏幕上删除，`odder` 数组的指针向后移动一位，如此循环往复，直到完成整个约瑟夫环问题的链表模拟可视化实现。

通过 2π 均分获取角度差，通过画布的长宽确定中心点。通过 `math.sin(rad * (Odder[i1] - 1))`, `300 + 200 * math.cos(rad * (Odder[i1] - 1))` 分别计算各个圆的位置。

5.1.3 删除元素的渐变实现函数：disappear(screen, rad)

```

def kill(screen,rad,color):
    DrawArc(screen, 300 + 200 * math.sin(rad), 300 + 200 *
math.cos(rad),
        color, 31)
    pygame.display.update()
    pygame.time.delay(1)

def disappear(screen,rad):
    color = 0

```

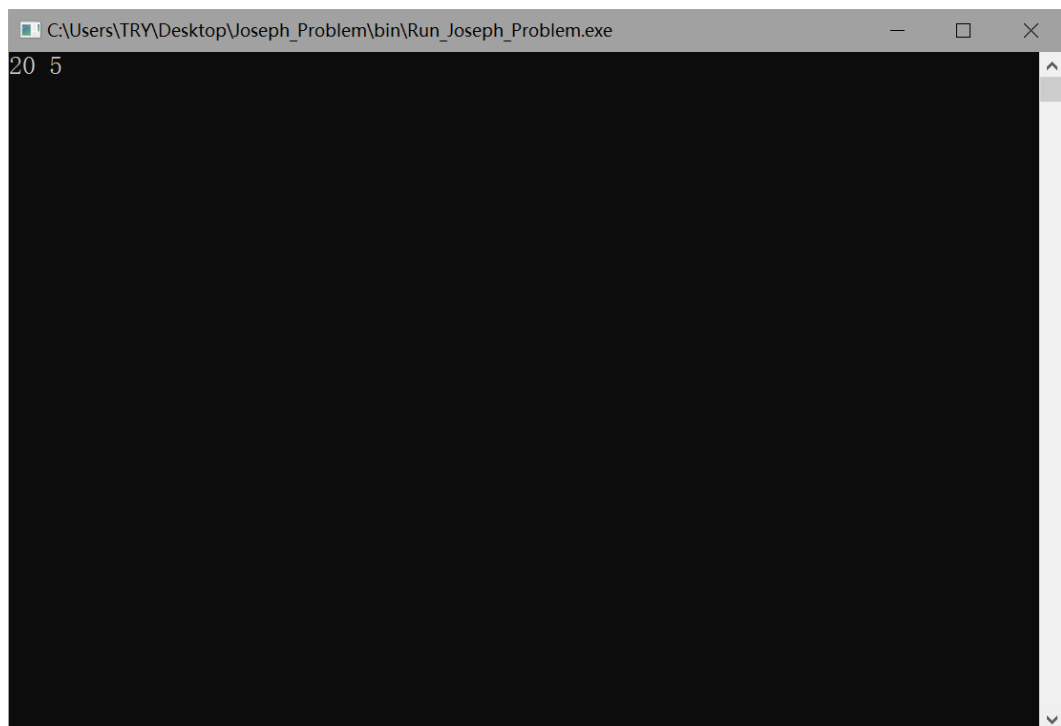
```
while(color != 256):  
    kill(screen,rad,(color,color,color))  
    color += 1
```

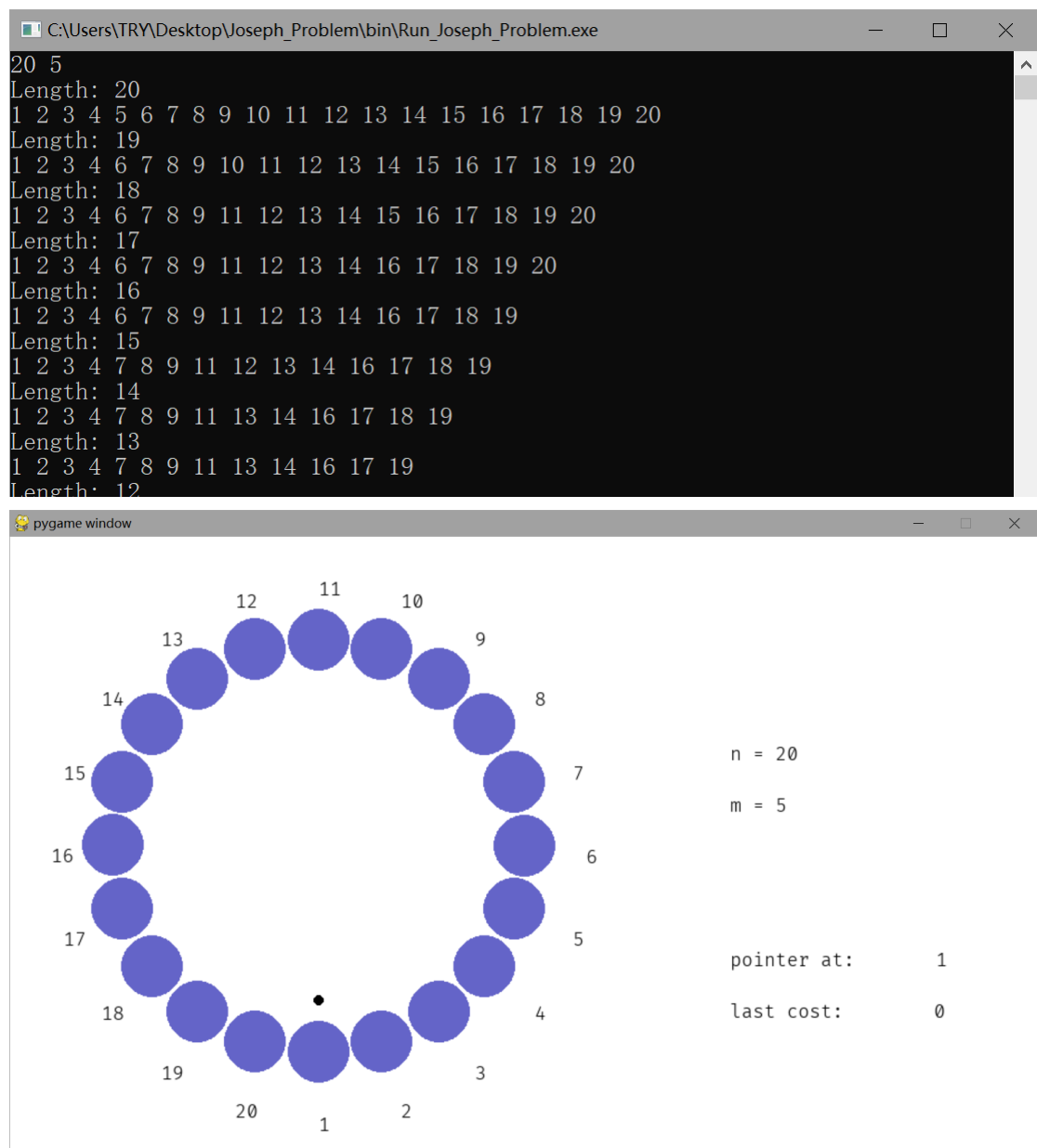
实现原理：在很短的时间内，不断打印从（0,0,0）到（255,255,255）颜色的圆，将原先的元素覆盖为背景色

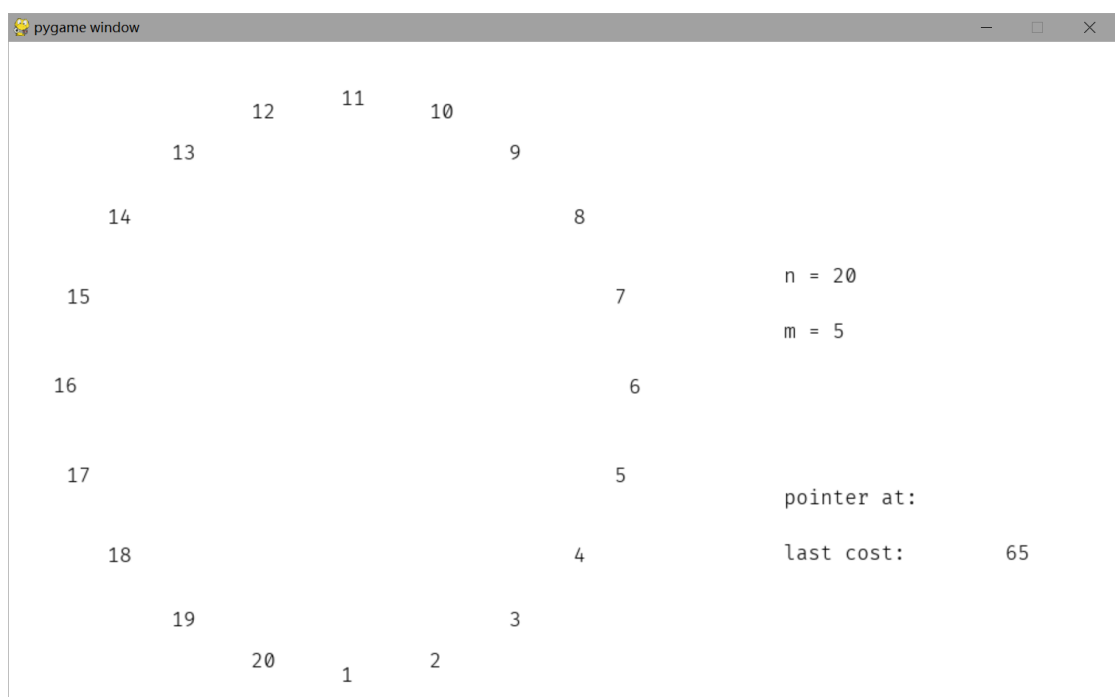
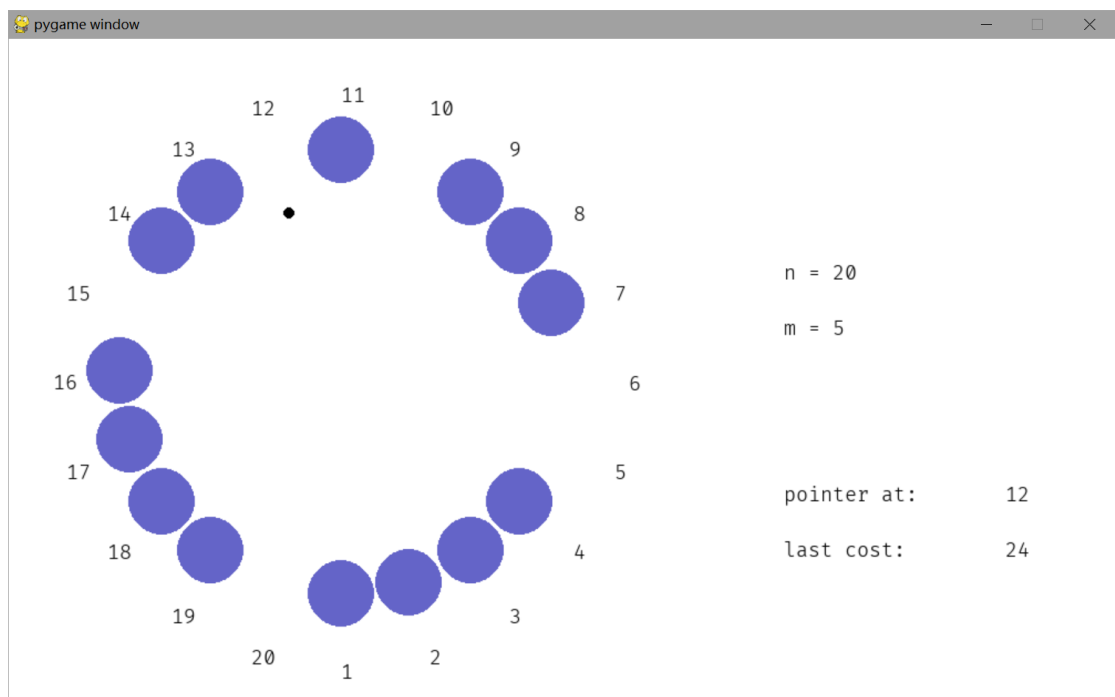
5.2 用户使用说明

运行 bin 文件夹中的 Run_Joseph_Problem.exe,输入 n m 后实现约瑟夫环的可视化

5.3 示例







6 提交材料说明

提交材料	是否提交	文件名称
1、实验报告	是	实验报告_lab1.pdf
2、可执行程序	是	Source/lab_1.exe
3、源程序，如果是多个文件要压缩到一个文件	是	Replacement.rar
4、可视化模型可执行文件	是	Visual/bin/Joseph_Problem.exe

上述内容可压缩到一个提交文件内，在文件名称处都填写一个文件名称。