东北大学秦皇岛分校

计算机与通信工程学院

数据结构课程设计

设计题目 在 20 亿个整数中查找出现次数最多的数

专业名称	计算机科学与技术
班级学号	1802 班 20188068
学生姓名	**孔天欣**
指导教师	程绍辉
设计时间	**2019** 年 **12** 月 **29** 日— **2020** 年 **1** 月 **5** 日

课程设计任务书

专业：计算机科学与技术 **学号：**20188068 **学生姓名（签名）：**

设计题目：在 20 亿个整数中查找出现次数最多的数

一、设计实验条件

Windows 10 系统的电脑，IDE 为 Visual Studio 2017.

二、设计任务及要求

1. 有一个包含 20 亿个全是 32 位整型的大文件，在其中找到出现次数最多的数。
2. 内存限制为 2GB。

三、设计报告的内容

1. 设计题目与设计任务（设计任务书）
2. 前言（绪论）(设计的目的、意义等)
3. 设计主体（各部分设计内容、分析、结论等）

3.1 需求分析

以无歧义的陈述说明程序设计的任务，强调的是程序要做什么？给出功能模块图和流程图。同时明确规定：输入的形式和输出值的范围；输出的形式；程序所能够达到的功能；测试数据：包括正确的输入及其输出结果和含有错误的输入及其输出结果。

3.2 系统设计

说明本程序中所有用到的数据及其数据结构的定义，包含基本操作及其伪码算法。画出函数之间的调用关系图；写出主程序及其主要模块的伪码流程。

3.3 系统实现

给出算法的实现；程序调试过程中遇到的问题是如何解决的；对设计与实现的回顾和分析；算法的时空分析和改进思想。

3.4 用户手册

说明任何使用你编写的程序，详细列出每一步的操作步骤。

3.5 测试

给出测试过程及结果。

4. 结束语（设计的成果，展望等）

5. 参考资料

6. 附录

带注释的源程序。

四、设计时间与安排

1、设计时间：1 周

2、设计时间安排：

熟悉实验设备、收集资料：1 天

设计图纸、实验、计算、程序编写调试：3 天

编写课程设计报告：2 天

答辩：1 天

课程设计报告

前言

本次课程设计的题目为在 20 亿个数字中寻找出现频次最多的数，且内存使用量少于 2GB。本次题目对学生的数据结构掌握能力提出了较高的要求，首先在数据的存储上，就要求学生掌握各种数据结构的基础，例如，这 20 亿的数字应当用什么方式存储，使之能够实现较快的查找。同时，内存少于 2GB 这一要求，需要学生掌握内存管理的相关知识和算法，例如，20 亿的数字不能一次性全部读入内存，需要用什么样的方式进行内存管理使之不发生溢出。

当代社会发展日新月异，数据不断迭代更新，互联网已然进入大数据时代，每天都需要有海量的数据进行处理，因此使用恰当的数据结构和算法对这些数据进行高效的分析和运算已经成了极其必要之需求，如今的互联网公司也专门开设了大数据部门来应对这种需求。本次课程设计的题目能够让学生体验到大数据处理的一些基本思想，例如“分而治之”等，还能提高对算法的时间复杂度和以及内存管理的能力，并能够在此基础上进行分析和改进算法，从而在总体上提高程序整体效率。学生通过课程设计题目的练习，能够加深对数据结构和算法的了解，将各项知识点融会贯通，从而提高数据结构和算法的技能水平。

设计主体

3.1 需求分析

设计的程序要求能够对包含 20 亿（或以下）个 32 位正整数的大文件进行分析，并从中找出出现次数最多的数字，且要求使用内存量小于 2GB.

本程序能够提供如下功能：

1、生成一个包含 20 亿个（或以下）个 32 位正整数的大文件。

2、加载一个包含 20 亿个（或以下）个 32 位正整数的大文件。

3、分析文件，找出其中出现频次最高的正整数。

4、查看当前各项参数。

本程序的功能模块图见图 1。

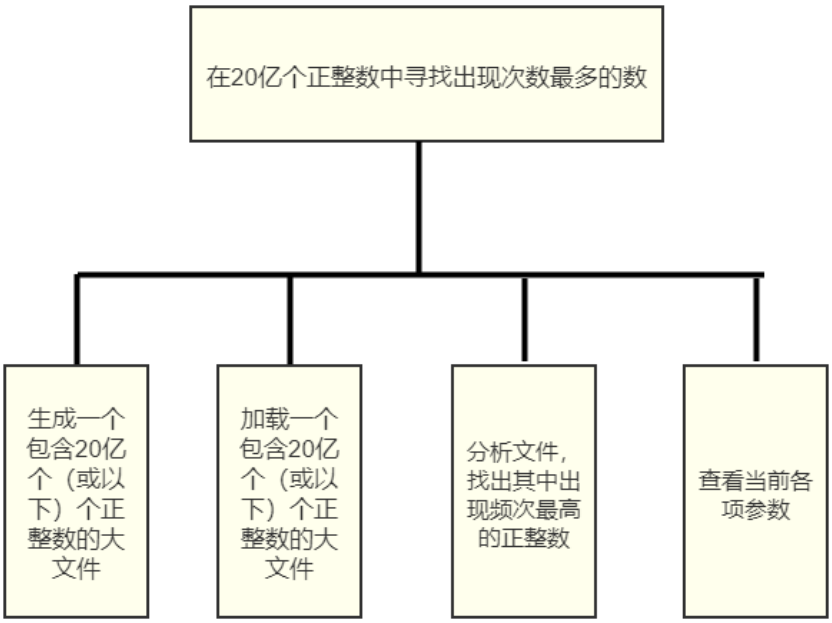

图 1 功能模块图

用户通过输入序号的方式进行交互和各项操作，序号和上文的选项号码一致。本程序的流程图见图 2。

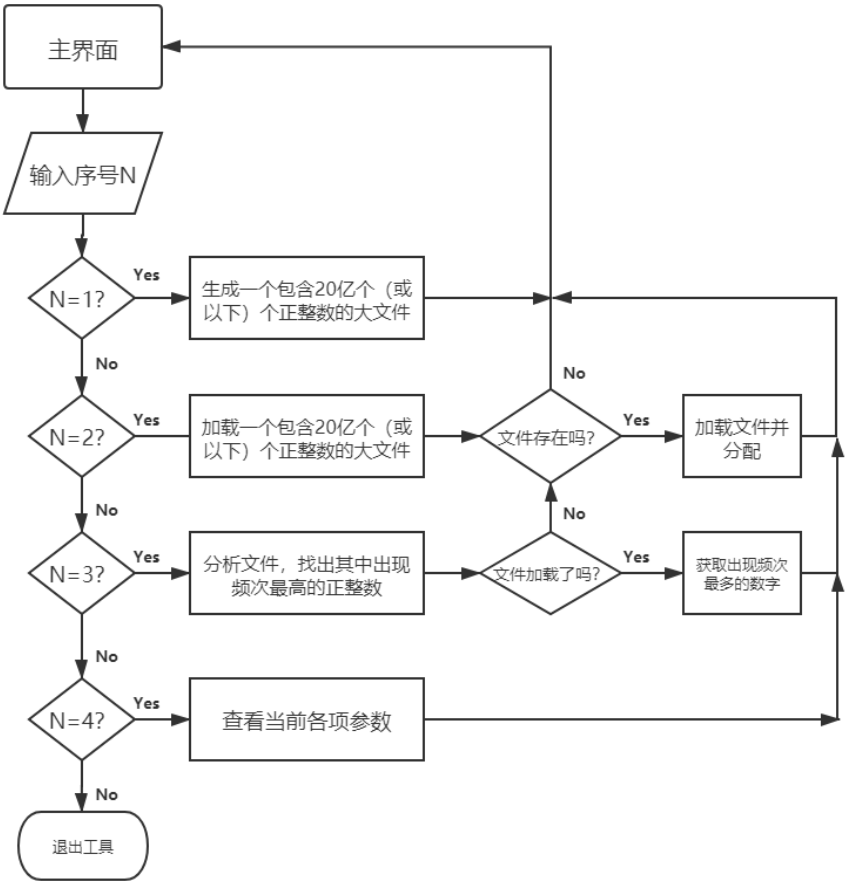

图 2 系统流程图

该程序对于输入和输出有以下要求：

1、在主界面中，输入是界面中要求的序号数字，其他形式的输入会被判定为无效输入并退出程序。

2、在输入要生成的正整数的个数时，输入的数字适宜范围是 20 亿以内的 32 位正整数，错误的输入会回到主界面。

3、在输入要哈希映射到的文件个数时，输入的数字要求是正整数且能够被生成数字的个数除尽，错误的输入会回到主界面。

4、本程序的输出是大文件中出现次数最多的数字以及它的出现频次。

鉴于大文件数字都为随机生成，测试时需要使用已存在的文件。为了避免程序所在文件夹占用硬盘空间太大，测试用例的生成数字量皆为 100 万以内，同时，为提高读取效率，测试用例采用二进制存储方式，本程序所在文件夹下含有“测试用例”文件夹，在使用前选择一文件并放到上级目录下，重命名为“numbers.txt”。在程序中运行结果如下：

第一个文件名称为“1 万个数字”，运行结果为：出现次数最多的数字是：1962，出现次数是 7 次。

第二个文件名称为“10 万个数字”，运行结果为：出现次数最多的数字是：16217，出现次数是 12 次。

第三个文件名称为“100 万个数字”，运行结果为：出现次数最多的数字是：521805，出现次数是 54 次。

3.2 系统设计

本程序用到的全局数据有：

1、maxNum：生成的数字个数

2、filesNum：通过哈希映射分配到的文件个数

3、addrNum：每个哈希表的表长（=生成数字个数/文件个数）

本程序采用哈希表作为数据结构，并采用链地址法存储，其定义如下：

```
1.  /* 链表的每个结点 */
2.  typedef struct HashNode {
```

```
    /* 关键字 */
    int key;
    /* 记录出现频次 */
    int value;
        /* 同地址的下一个关键字结点 */
    HashNode *next;
};

/* 数组链表 */
typedef struct
{
    HashNode* firstout;

}*HashList, HashElem;

/* 整个哈希表 */
typedef struct HashMap {
    HashList adr;
    /* 表长 */
    int num;
}HashMap;
```

该结构和图的邻接表结构类似，每一个关键字都是一个结点，该结点存储关键字、关键词出现的频次、下一个关键词结点，相同哈希地址的关键字结点链接在同一链表上，多个哈希地址的链表共同组成一个数组，该数组的每个元素都是一个链表，并附设一个数据域 num 记录表长。

对于此哈希表，存在的基本操作和算法如下：

1. void InitHashMap(HashMap *hashmap); 初始化哈希表

```
void InitHashMap(HashMap *hashmap)
{
    hashmap->adr = new HashElem[addrNum];
    for (int i = 0; i < addrNum; i++)
    {
        hashmap->adr[i].firstout = NULL;
    }
    hashmap->num = addrNum;
}
```

该哈希表分配了长度为 addrNum 的表长，同时每个链表都设置为无结点（NULL），并初始化该哈希表表长。

2. int Hash(int key); 哈希函数

```
1.   int Hash(int key)
2.  {
3.      return key % addrNum;
4.  }
```

本哈希表采用链地址法存储，由于冲突的关键词都在同一链表中存储。因此设计哈希函数时不需要另设处理冲突的公式。

3. void add(HashMap *hashmap, int key) 添加一个关键词到哈希表

```
1.  /* 向哈希表中插入关键字 */
2.  void add(HashMap *hashmap, int key)
3.  {
4.      int address = Hash(key);
5.      HashNode *p = hashmap->adr[address].firstout;
6.      HashNode *temp = p;
7.      /* flag=0 不存在相同关键字 flag=1 存在相同关键字 */
8.      int flag = 0;
9.      while (p)
10.     {
11.         /* 如果有冲突，则添加该关键词的词频 */
12.         if (p->key == key)
13.         {
14.             p->value++;
15.             flag = 1;
16.             break;
17.         }
18.         else
19.         {
20.             temp = p;
21.             p = p->next;
22.         }
23.     }
24.     /* 如果没有冲突，则添加关键词至表尾 */
25.     if (!flag)
26.     {
27.         if (temp == p)
28.         {
29.             p = new HashNode;
30.             p->next = hashmap->adr[address].firstout;
31.             hashmap->adr[address].firstout = p;
32.         }
```

```
33.         else
34.         {
35.             p = new HashNode;
36.             p->next = temp->next;
37.             temp->next = p;
38.         }
39.
40.         p->value = 1;
41.         p->key = key;
42.     }
43. }
```

为了记录关键字的词频，如果出现了冲突，首先看关键字是否已经存在，如果是，该关键字所在结点的出现频次（value）自增，如果扫描完这条链表，发现关键词不存在，则把关键字添加至表尾，初始化它的词频为 1。由于大多数时间都会发生冲突，因此此后的查找算法时间复杂度不再是 O(1)。

4. void findMaxOccurrence(HashMap *hashmap, int &maxkey, int &maxFrequency) 获取最大词频数字及其词频

```
1.  /* 返回这个哈希表的最大词频以及最大词频对应的关键词 */
2.  void findMaxOccurrence(HashMap *hashmap, int &maxkey, int &maxFrequency)
3.  {
4.      maxkey == -1;
5.      maxFrequency = -1;
6.      for (int i = 0; i < hashmap->num; i++)
7.      {
8.          HashNode *p = hashmap->adr[i].firstout;
9.          while (p)
10.         {
11.             if (p->value > maxFrequency)
12.             {
13.                 maxkey = p->key;
14.                 maxFrequency = p->value;
15.             }
16.             p = p->next;
17.         }
18.     }
19. }
```

其原理是扫描整个哈希表的所有关键字结点，并获取其中词频最高的关键字，返回的是关键字及其词频。

5. void destory(HashMap *hashmap) 销毁哈希表

```
1.  void destory(HashMap *hashmap)
2.  {
3.      for (int i = 0; i < hashmap->num; i++)
4.      {
5.          HashNode *p = hashmap->adr[i].firstout;
6.          HashNode *q = p;
7.          while (p)
8.          {
9.              q = p;
10.             p = p->next;
11.             delete q;
12.         }
13.     }
14.     delete[] hashmap->adr;
15.     delete hashmap;
16. }
```

扫描所有结点，释放所有结点所占的空间，同时释放整个结构体数组所占的空间来销毁整个哈希表。

对于整个主程序，其对应的每个选项调用关系见图 3：

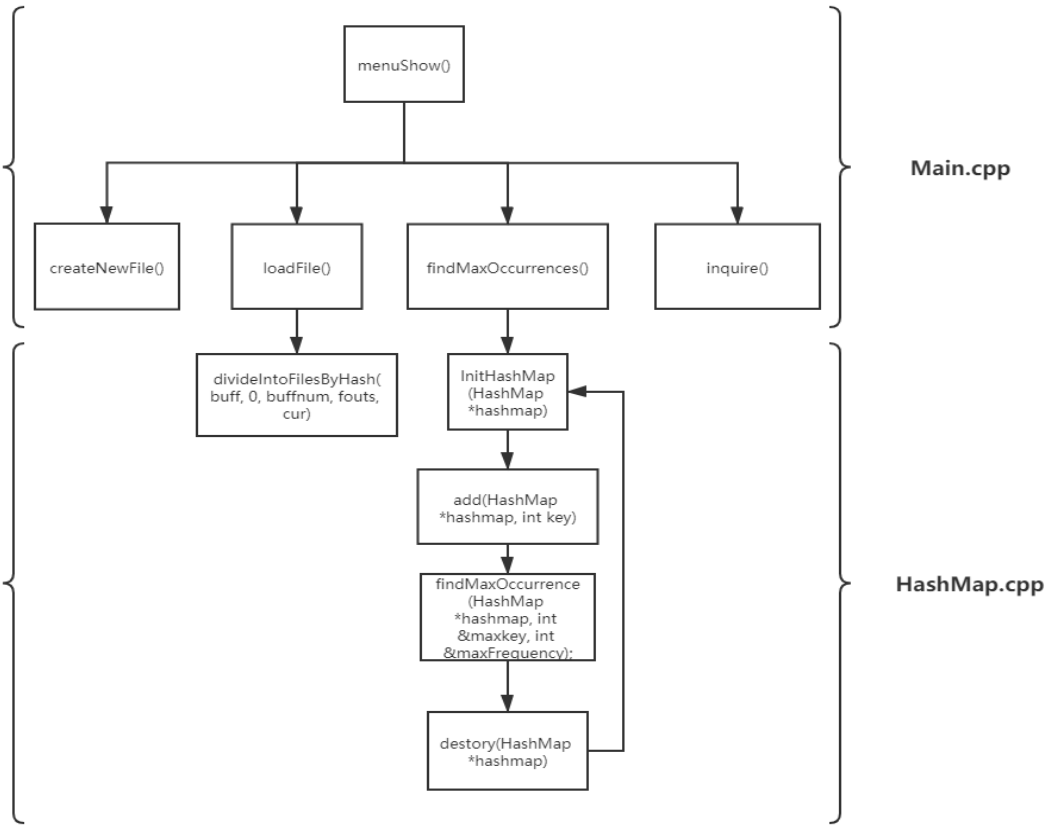

图 3 函数调用关系图

主界面的调用代码如下：

```
int main()
{
    while (1)
    {
        int opt = menuShow();
        switch (opt)
        {
        case 1:createNewFile(); break;
        case 2:loadFile(); break;
        case 3:findMaxOccurrences(); break;
        case 4: inquire(); break;
        default:exit(0); break;

        }
    }

}
```

主界面核心功能的函数的原理如下：

1. void createNewFile() 生成一个包含 20 亿个（或以下）个 32 位正整数的大文件

```
void createNewFile()
{
    /* 输入 n，异常判断 */

    /* 更新全局变量 */
    maxNum = n;

    /* 随机生成 maxNum 个数 */
    for (int i = 1; i <= maxNum; i++)
    {
        int temp = /* 随机数 */
        fwrite(&temp, sizeof(int), 1, fout);
    }
}
```

生成 maxNum 个随机数，其中随机数是整数，它是通过随机数种子来生成的，区间为[0,maxNum]。

2. void loadFile() 加载一个包含 20 亿个（或以下）个 32 位正整数的大文件

```
1.  /* 加载大文件，同时通过哈希函数映射到若干文件上 */
2.  void loadFile()
3.  {
4.      /* 将关键词分配的文件个数 */
5.      cin >> files;
6.
7.      /* 异常处理 */
8.
9.      /* 修正数字个数，文件个数和小哈希表表长 */
10.     maxNum = len;
11.     filesNum = files;
12.     addrNum = maxNum / filesNum;
13.        /* 异常处理 */
14.
15.     /* 预载所有文件指针，避免频繁开关输出流 */
16.
17.        /*哈希映射*/
18.     while (ftell(fin) != end)
19.     {
20.         fread(buff, sizeof(int) * buffnum, 1, fin);
21.         /* 进行哈希映射并分到各个小文件 */
22.         divideIntoFilesByHash(buff, 0, buffnum, fouts, cur);
23.     }
24.
25.     /* 关闭所有文件指针 */
26. }
```

将大文件中的每一个关键字都进行哈希映射，然后将所有关键字分开存入不同的小文件当中，以此避免一次性读大文件导致的内存空间不足。每个小文件存储一定的区间范围的哈希地址对应的关键词，“分而治之”。由于本程序哈希函数的性质，重复的元素哈希地址一定是相同的，因此必定在同一个文件中。此后的哈希表便可以大幅节省内存开销。

3. void findMaxOccurrences() 分析文件，找出其中出现频次最高的正整数

```
1.  /* 在所有哈希表中寻找出现频次最多的关键词 */
2.  void findMaxOccurrences()
3.  {
4.      /* 异常处理 */
```

```
5.
6.      /* 每个文件出现次数最多的关键词数组以及它们对应的出现频次 */
7.      int *maxkeys = new int[filesNum];
8.      int *maxfrequency = new int[filesNum];
9.
10.     /* 开始对每个小文件进行哈希表存储和关键词筛选 */
11.     for (int i = 0; i < filesNum; i++)
12.     {
13.         HashMap *hashmap = new HashMap;
14.         InitHashMap(hashmap);
15.         int key;
16.         sprintf(filePath, "Collection\\%d.txt", i);
17.         FILE *fin = fopen(filePath, "rb");
18.
19.         while (ftell(fin) != end)
20.         {
21.             fread(&key, sizeof(int), 1, fin);
22.             add(hashmap, key);
23.         }
24.         fclose(fin);
25.
26.         maxfrequency[i] = -1;
27.         /* 获取这个文件的频次最高关键词及其频次 */
28.         findMaxOccurrence(hashmap, maxkeys[i], maxfrequency[i]);
29.         remove(filePath);
30.         destory(hashmap);
31.     }
32.         /* 获得这批数组中出现次数最多的数字的下标 */
33.     int maxcur = getMax(maxfrequency, maxkeys);
34. }
```

加载文件完毕后，一个大文件通过哈希映射分割成了不同数量的小文件，此时的哈希表的表长可以取每个文件平均分配的关键字个数。使用一张哈希表对一个文件的所有关键字进行存储和词频分析完毕后，就销毁该表，以保证内存空间充足。同时，设立辅助数组 maxkeys[]和 maxFrequency[]，分别存储每个文件中出现次数最多的数字以及它的出现频次，最后将数组中的所有数字的频次进行大小比较，其中的最大值就是整个大文件出现频次最高的数字。

4. void manualSetting() 查看当前各项参数

```
1.  void inquire()
2.  {
```

```
    cout << "当前的各项参数如下：" << endl;
    cout << "生成的数字个数：";
    if (!maxNum) { cout << "未初始化" << endl; }
    else { cout << maxNum << endl; }
    cout << "分配的文件个数：";
    if (!filesNum) { cout << "未初始化" << endl; }
    else { cout << filesNum << endl; }
    cout << "每个小哈希表的表长：";
    if (!addrNum) { cout << "未初始化" << endl; }
    else { cout << addrNum << endl; }
    system("pause");
}
```

可以查看的参数有 maxNum，filesNum 和 addrNum。程序启动时，各项数值均初始化为 0。

3.3 系统实现

1. 读取大文件，并将关键字通过哈希映射分配到 N 个文件中

```
/* 加载大文件，同时通过哈希映射到若干文件上 */
void loadFile()
{
    FILE *fin;
    fin = fopen("numbers.txt", "rb");
    if (!fin) { Exception(5); return; }

    cout << "将关键词分配到多少个文件中？" << endl;
    int files;
    cin >> files;

    /* 异常处理 */
    if (files <= 0) { Exception(1); return; }
    if (maxNum%files != 0) { Exception(3); return; }

    /* 获得文件字节大小（end)以及数字个数（len） */
    fpos_t end;
    fpos_t now = -1;
    fseek(fin, 0, SEEK_END);
    fgetpos(fin, &end); //获取文件大小
    long int len = end / sizeof(int);
    rewind(fin);

    /* 修正数字个数，文件个数和小哈希表表长 */
```

```
25.     maxNum = len;
26.     if (!maxNum) { Exception(2); return; }
27.     filesNum = files;
28.     addrNum = maxNum / filesNum;
29.
30.     cout << "开始加载大文件，需要较长时间，请稍等..." << endl;
31.     double start = clock();
32.
33.     /* 规定从缓冲区读取的字节个数 */
34.     const int buffnum = 1024;
35.     int buff[buffnum];
36.
37.     char filePath[20];
38.
39.     FILE** fouts = new FILE*[filesNum];
40.
41.     /* 预载所有文件指针，避免频繁开关输出流 */
42.     for (int order = 0; order < filesNum; order++)
43.     {
44.         sprintf(filePath, "Collection\\%d.txt", order);
45.         remove(filePath);
46.         fouts[order] = fopen(filePath, "ab+");
47.     }
48.
49.     int cur = 0;
50.     while (now!=end)
51.     {
52.         fread(buff, sizeof(int) * buffnum, 1, fin);
53.         fgetpos(fin, &now);
54.         /* 进行哈希映射并分到各个小文件 */
55.         divideIntoFilesByHash(buff, 0, buffnum, fouts, cur);
56.     }
57.
58.     cout << "成功通过哈希函数映射值将关键字分配到" << filesNum << "个文件中,,总共耗
    时 " << (clock() - start) / CLOCKS_PER_SEC << "秒！" << endl;
59.
60.     /* 关闭所有文件指针 */
61.     for (int order = 0; order < filesNum; order++)
62.     {
63.         fclose(fouts[order]);
64.     }
65.     fclose(fin);
66.
67.     system("pause");
68. }
```

在实现读取大文件的算法时，遇到了许多问题，例如文件读取速度相当慢，异常处理问题较多等。在文件读取相当慢方面，一开始本人采用 read 函数进行读取，但速度不尽人意，读取 100 万个数字需要 172 秒左右。经过资料查阅，发现 read 函数并不会设立缓冲区，它每次读取字节都是直接从文件中读，因此时间开销较大。后来改用 fread 函数，该函数封装好了缓冲区，每次都会先从文件中取一部分数据放到缓冲区中，此后的读取便直接从内存中读取，减少了文件访问的次数，经测试，读取 100 万个数字的时间提高了 40 秒左右。但由于读取时间还是太长，本人进行了扩大缓冲区处理，让输入流读取时能够一次读取更多数据放到内存，然而效果不明显，读取时间仅仅减少了 4、5 秒。后来本人经过代码分析，改进了代码流程。原流程是读取一个关键词—>分析关键词—>计算出关键词该放在哪个文件—>打开该文件输出流—>放入关键词—>关闭输出流，如此计算，读取 100 万个数字需要开关 100 万次输出流，如此频繁的开关输出流造成的时间开销是相当大的。因此本人改进流程，在读取大文件前，预先加载所有输出流，等到所有关键字都放进文件后，统一关闭所有输出流，这样开关输出流的次数是 2N（N 为分配的文件个数）次，大大减少了时间开销，经测试，读取 100 万个数字时间为 0.6 秒，比原先快了两百多倍，读取时间有了质的改善。

后来尝试开多线程进行分批同时处理数据，可能是个人水平有限，对多线程原理了解不够深刻，时间开销反而变大，因此改回单线程处理。

2.用哈希映射分配到文件中

```
1.  void divideIntoFilesByHash(int key[], int start, int end, FILE* fouts[], int &
    cur)
2.  {
3.      while (start < end)
4.      {
5.          if (key[start] < 0)
6.          {
7.              break;
8.          }
9.          else
10.         {
11.             int period = addrNum;
12.
```

```
            /* !不能使用 hash 函数，因为这个是对所有数据的哈希映射，而不是对小文件数据的哈希映射 */
            int value = key[start] % maxNum;

            int order = value / period;

            /* 将哈希值进行划分，每个文件存储一定区间的哈希值对应的关键字 */
            fwrite(&key[start], sizeof(int), 1, fouts[order]);
            processAnimation(cur++, maxNum);
            start++;
        }

    }

}
```

本算法的时间复杂度为 O(n)。代码内部本来是通过 for 循环来确定关键字该放入的文件，时间复杂度为 O(n)，导致整段代码的时间复杂度变成 $O(n^2)$。因此加以改进，把这段代码通过一个计算公式改成了 O(1)。start 和 end 是文件的缓冲区开头和结尾下标。事实上，可以直接传入缓冲区地址作为参数，但之前尝试了多线程未果，遗留下来这些参数，但也能充当缓冲区数组的下标上下限使用，所以未加修改。

3. 生成一个包含 20 亿个（或以下）个 32 位正整数的大文件

```
void createNewFile()
{
    int n;
    cout << "要生成多少个整数？（20 亿个以内）" << endl;
    cin >> n;
    if (n <= 0) { Exception(1); return; }

    /* 更新全局变量 */
    maxNum = n;

    /* 计时器 */
    double start = clock();

    /* 伪随机数随机种子 */
    srand((unsigned)time(NULL));
    cout << "正在生成" << maxNum << "个数字到一个大文件，请稍等，可能需要较长时间..." << endl;
```

```
17.
18.     FILE *fout = fopen("numbers.txt", "wb");
19.     if (!fout) { Exception(5); return; }
20.
21.     /* 随机生成 maxNum 个数 */
22.     for (int i = 1; i <= maxNum; i++)
23.     {
24.
25.         int temp = (int)((double)rand() / RAND_MAX * maxNum);
26.         fwrite(&temp, sizeof(int), 1, fout);
27.
28.         /* 模拟进度条，方便查看进度 */
29.         processAnimation(i, maxNum);
30.     }
31.     fclose(fout);
32.
33.     cout << endl << "生成完毕，总共耗
   时 " << (clock() - start) / CLOCKS_PER_SEC << "秒！" << endl;
34.     system("pause");
35.
36. }
```

本算法的时间复杂度为 O(n)，主要开销在于就是随机生成 maxNum 个数字，使用 for 循环依次生成。

4. 分析文件，找出其中出现频次最高的正整数。

```
1.  void findMaxOccurrences()
2.  {
3.      /* 如果没有加载过文件 */
4.      if (_access("Collection\\0.txt", 0) == -1) { Exception(4); return; }
5.      if (!addrNum || !filesNum) { Exception(2); return; }
6.
7.      /* 每个文件出现次数最多的关键词数组以及它们对应的出现频次 */
8.      int *maxkeys = new int[filesNum];
9.      int *maxfrequency = new int[filesNum];
10.
11.     cout << "开始对每个小文件进行哈希表存储和关键词筛选，请稍等..." << endl;
12.     for (int i = 0; i < filesNum; i++)
13.     {
14.         HashMap *hashmap = new HashMap;
15.         InitHashMap(hashmap);
16.
17.         char filePath[20];
```

```
18.         sprintf(filePath, "Collection\\%d.txt", i);
19.
20.         FILE *fin = fopen(filePath, "rb");
21.
22.         int key;
23.
24.         fseek(fin, 0, SEEK_END);
25.         int end = ftell(fin);
26.         rewind(fin);
27.
28.         while (ftell(fin) != end)
29.         {
30.
31.             fread(&key, sizeof(int), 1, fin);
32.             add(hashmap, key);
33.         }
34.         fclose(fin);
35.
36.         maxfrequency[i] = -1;
37.         /* 获取这个文件的频次最高关键词及其频次 */
38.         findMaxOccurrence(hashmap, maxkeys[i], maxfrequency[i]);
39.         if (maxfrequency[i] != -1)
40.         {
41.             cout << "第" << i + 1 << "个文件的出现次数最多的数字是:
    " << maxkeys[i] << ", 出现了 " << maxfrequency[i] << " 次。" << endl;
42.         }
43.         else
44.         {
45.             cout << "第" << i + 1 << "个文件没有数字存储。" << endl;
46.         }
47.
48.         remove(filePath);
49.         destory(hashmap);
50.     }
51.     int maxcur = getMax(maxfrequency, maxkeys);
52.     cout << endl << "-----------------------------------------------------------
    -------↓" << endl;
53.     cout << "综上，出现次数最多的数字是: " << maxkeys[maxcur] << ", 出现次数是
    " << maxfrequency[maxcur] << "次" << endl;
54.     system("pause");
55. }
```

上述算法的时间复杂度为 O(n)，通过哈希表对每一个文件的所以关键字进行存储和词频分析，然后把每一个文件的出现频次最高的数字存入辅助数组中。

最后在辅助数组中找到总体出现频次最高的数字。刚开始本人先是一次性直接生成 filesNum 个哈希表，然而当数字量为亿级时，就会出现 bad alloc 的错误，也就是内存空间申请不足。这是因为每个哈希表都至少占 8B * addrNum 个空间的大小，一次性生成所有哈希表至少需要申请 8B * addrNum * filesNum 个空间，如果是 20 亿个数字，则至多需要申请 16GB 的内存空间，这是远远不够的。因此调整策略，改成先申请一张哈希表，然后对一个文件的关键字进行存储，分析完毕后再销毁这张哈希表，最后申请新的哈希表，以此往复。如果是分配到 16 个文件，则一张哈希表的内存申请仅需要 1GB 左右，使用完了就销毁，没有占用，是满足需求设计中的不多于 2GB 内存的要求的。

3.4 用户手册

本程序提供 4 个选项供操作，分别可以实现不同的功能。用户在启动主程序后，通过输入每个选项对应的数字进行操作。主界面如下：

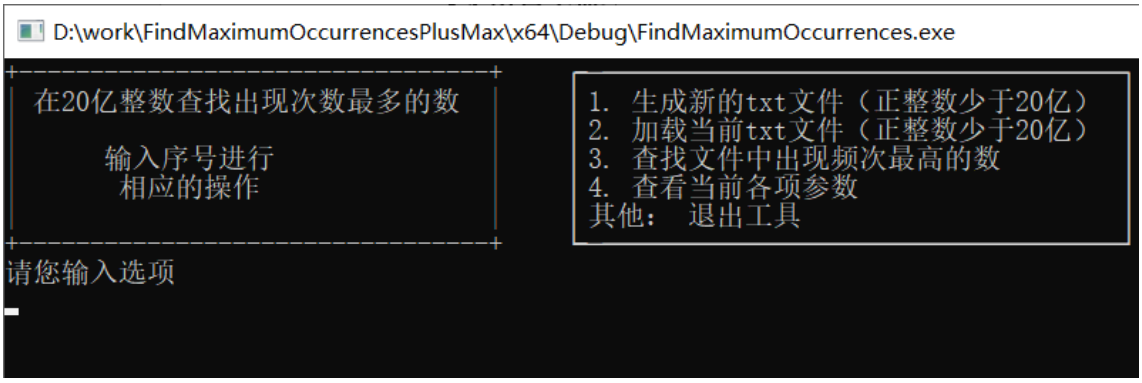

图 4 主界面演示

选项 1：用户输入 1 后，在文字引导下，可以输入一个 20 亿以内（推荐）的正整数，随后程序会自动生成一个文件(numbers.txt)，这个文件用二进制形式存储，存放在程序所在目录下，内含均为随机数，且数字的区间范围为[0, 用户输入的数字]，生成的数字个数就是用户输入的数字。见图 5。

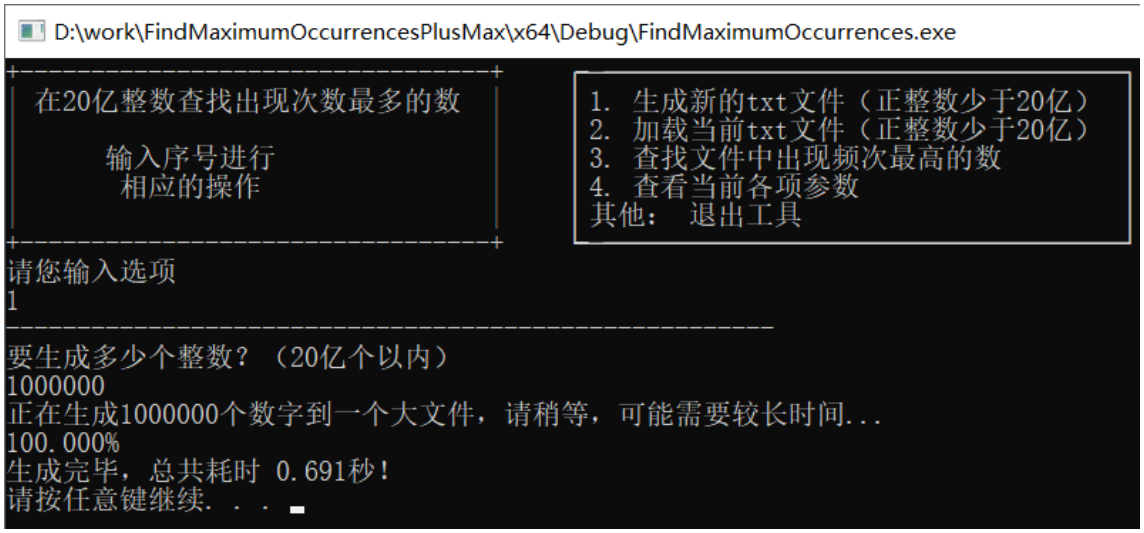

图 5 生成 100 万个随机正整数

选项 2：用户键入 2 后，系统首先询问用户要通过哈希函数映射到文件的个数（用户需要根据自身电脑内存情况进行合理判断），用户输入映射到的文件个数后，系统会加载当前的 txt 文件。如果用户输入的文件个数不能被生成的数字个数整除，会弹出错误提示。如果用户已经自带一个用二进制形式存储的 txt 文件，命名为“numbers.txt”且放在程序目录下，系统会自动读取该文件。如果没有，会发出错误警示。见图 6、7。

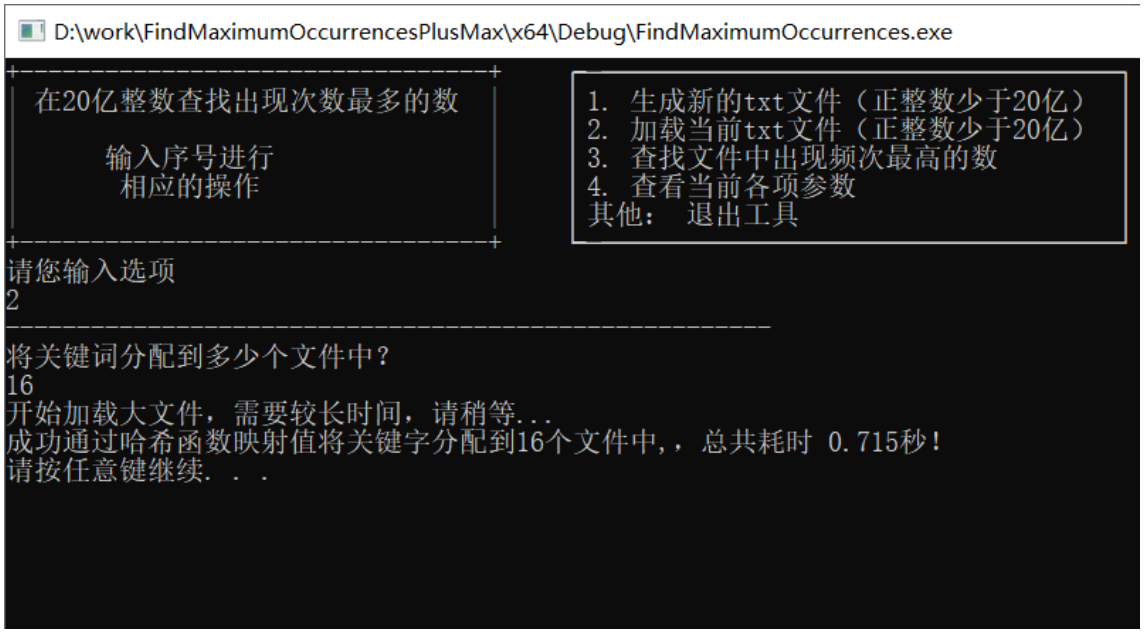

图 6 加载成功

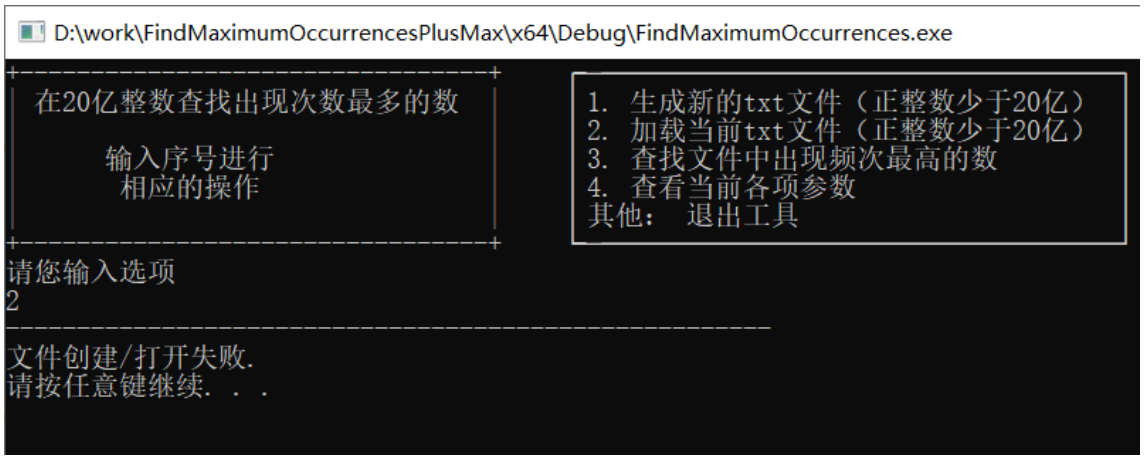

图 7 加载失败

选项 3：键入 3，如果用户先前已经键入 2 成功加载了文件，程序会自动进行分析，并给出返回结果。如果文件没有加载过，会弹出错误警示，见图 8、9.

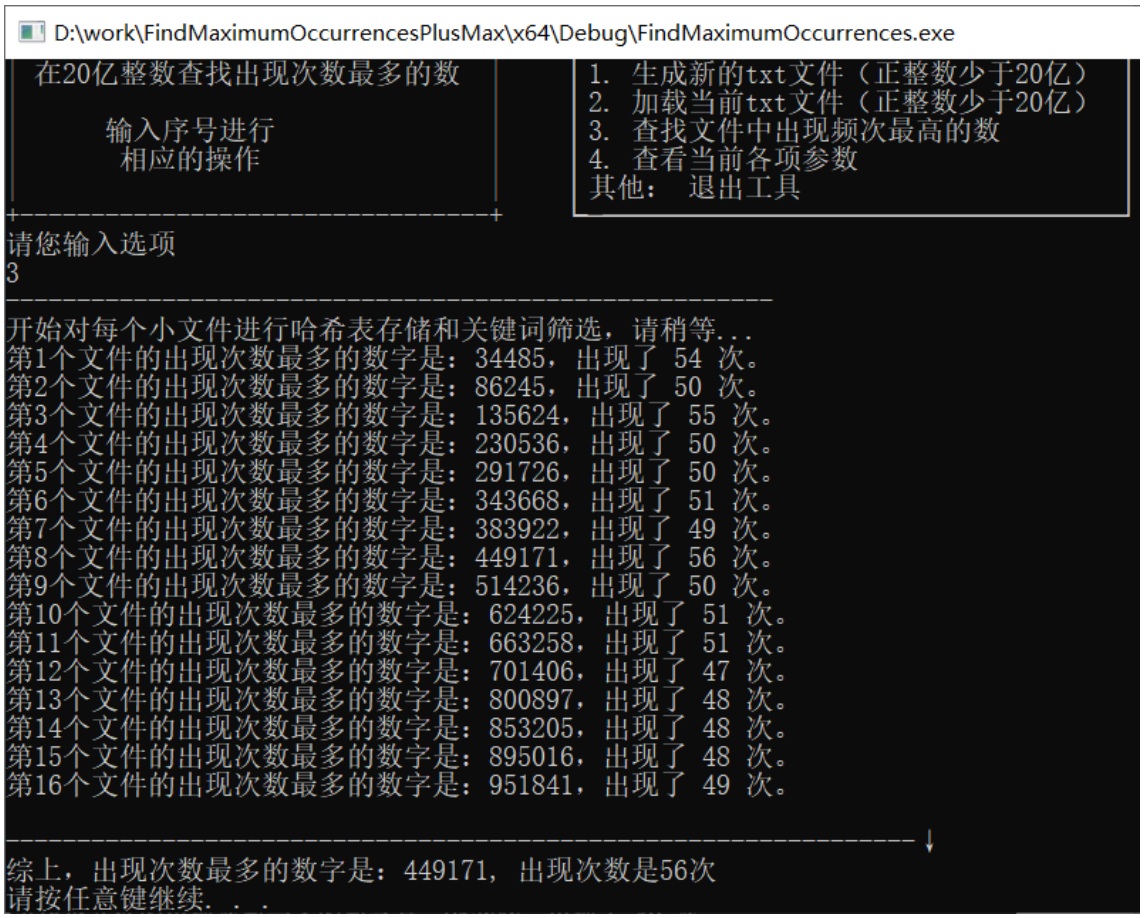

图 8 查找成功

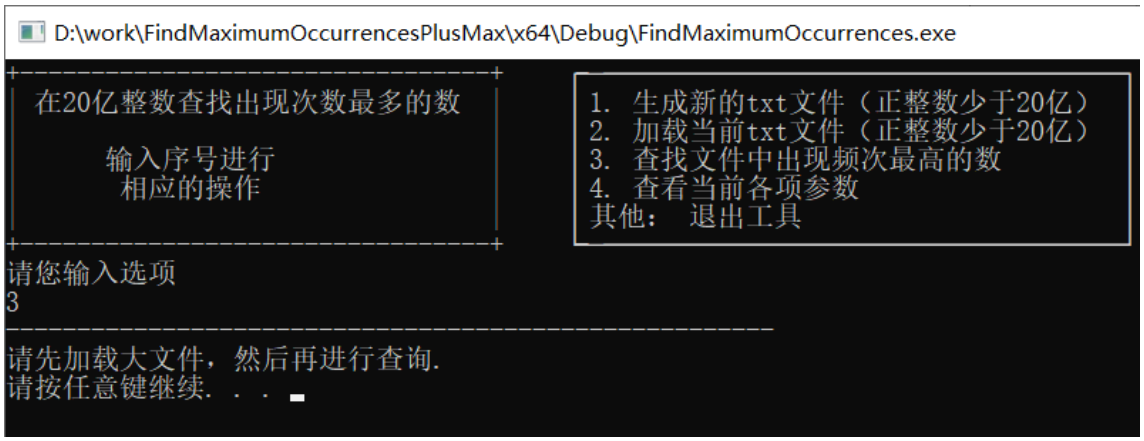

图 9 查找失败

由此下来，该程序的核心功能已经演示完毕。如果用户想查看当前各项参数，可以键入 4。如果刚启动程序，那么显示都是尚未初始化，需要用户自行键入 1、2、3 来进行初始化操作。在输入正确的前提下，用户键入 1 后输入的数字会初始化“生成的数字个数“，键入 2 后输入的数字会初始化”分配的文件个数”和“每个小哈希表的表长”。见图 10、11。

图 10 查看各项参数

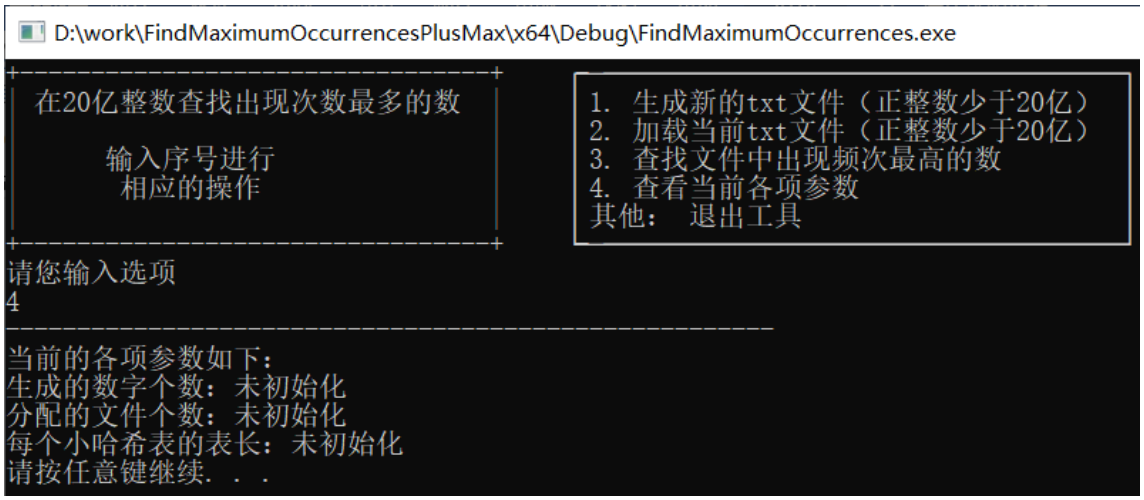

图 11 未初始化

程序的退出：用户输入其他数字即可直接退出程序，或者点击界面右上角的×。

3.5 测试

1. 1000 万个数字测试

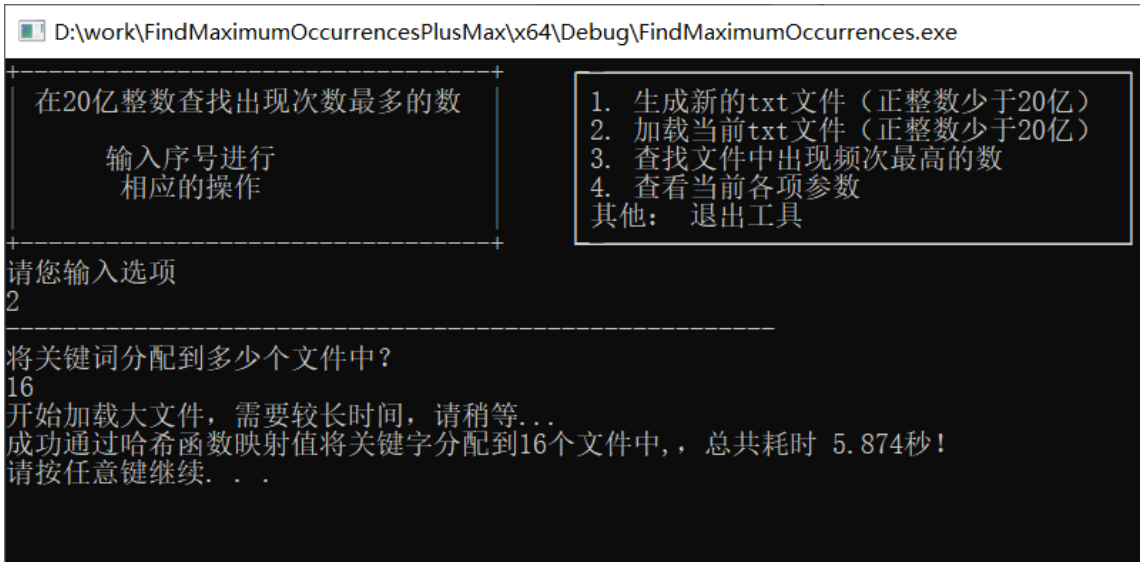

图 12 加载结果一

测试结果和内存占用情况见图 13、14。

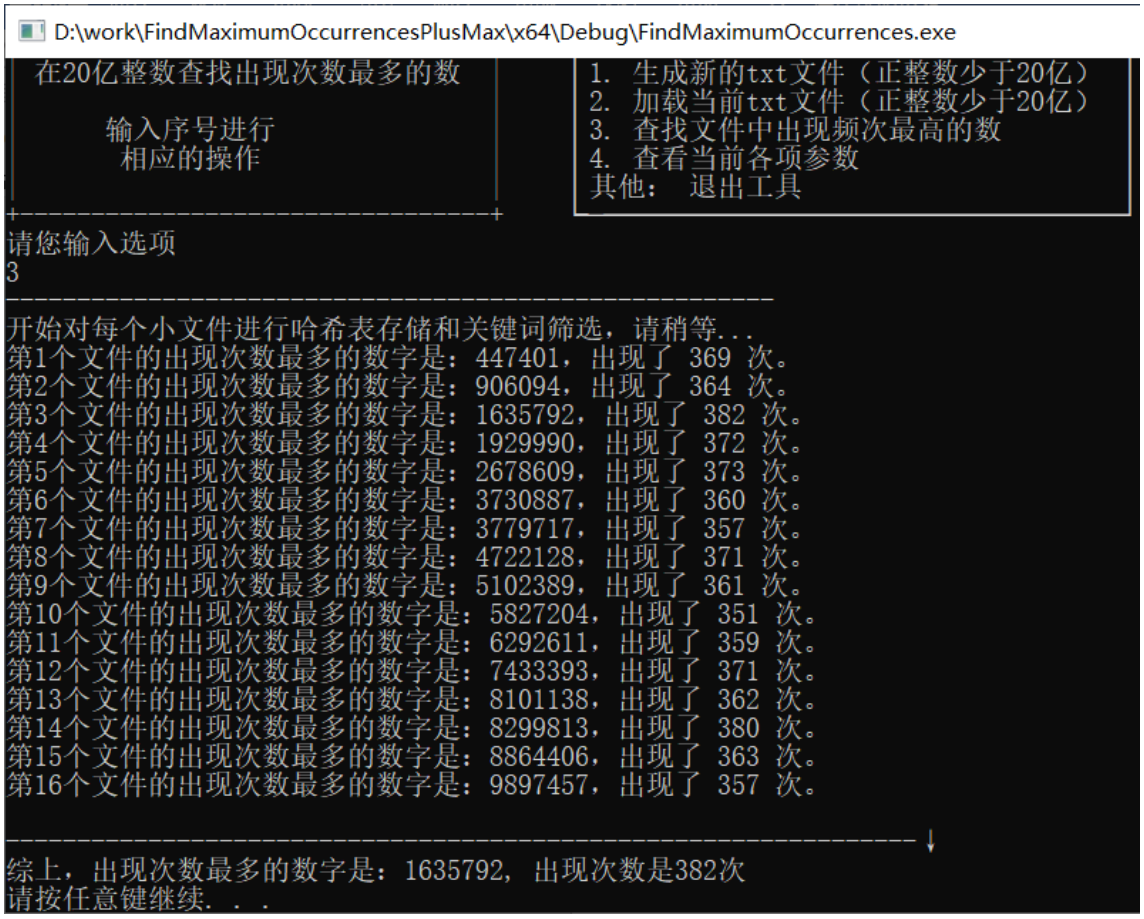

图 13 测试结果一

图 14 内存占用情况一

2. 2 亿个数字测试

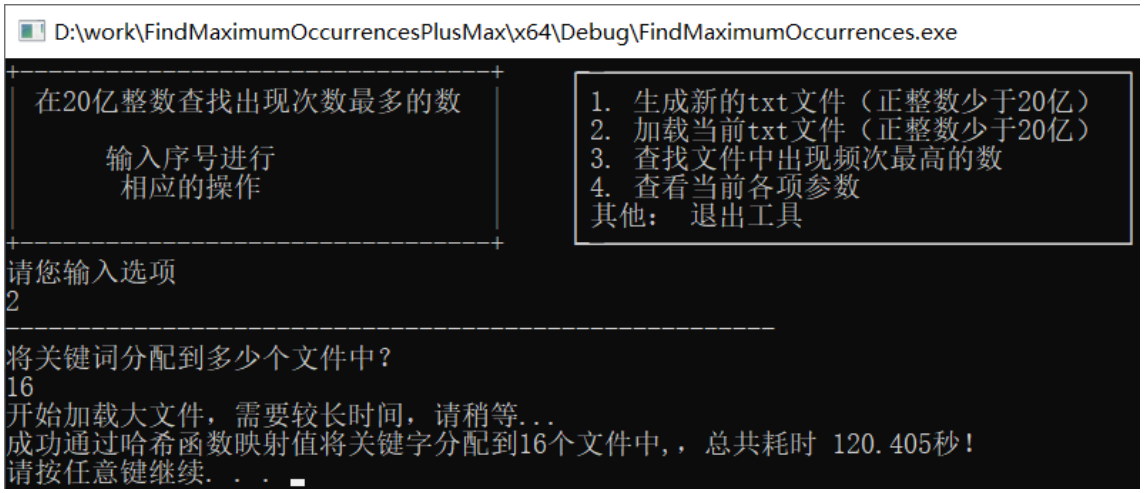

图 15 加载结果二

测试结果和内存占用情况见图 16、17。

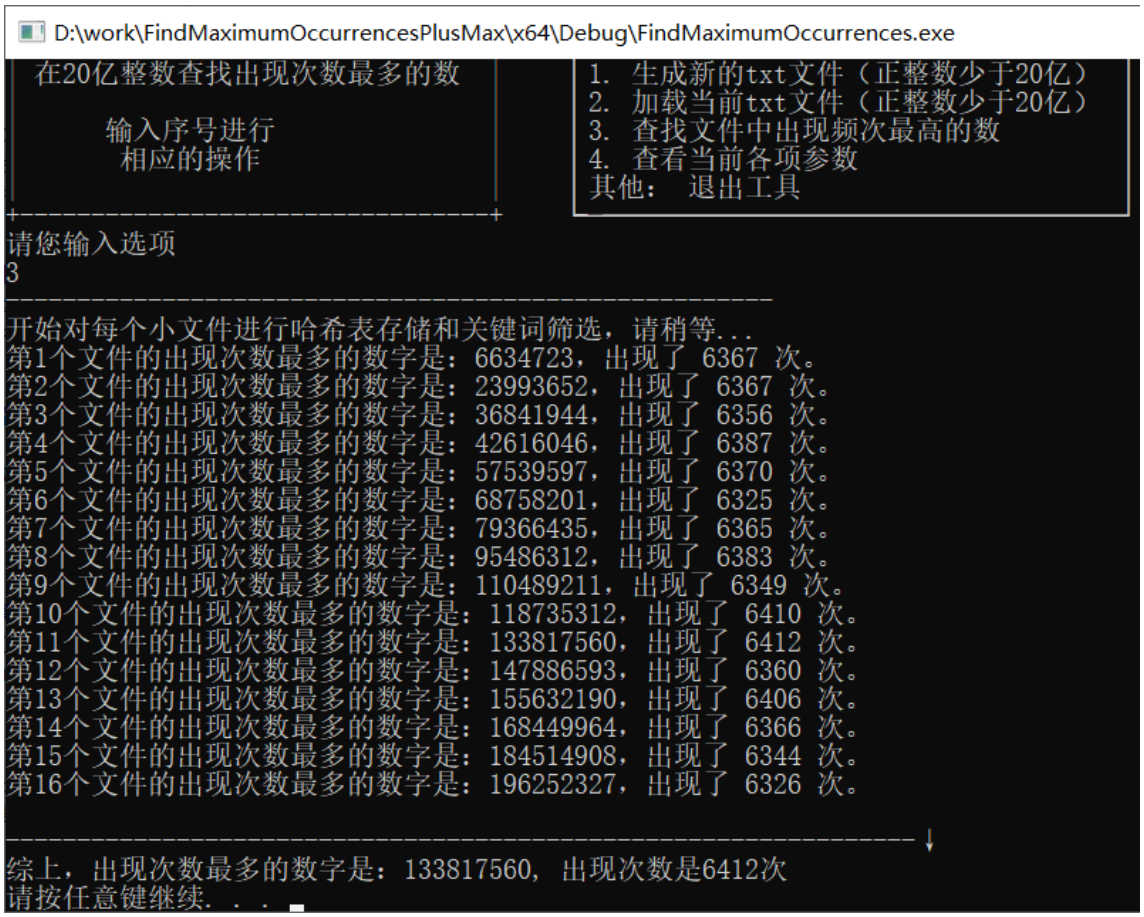

图 16 测试结果二

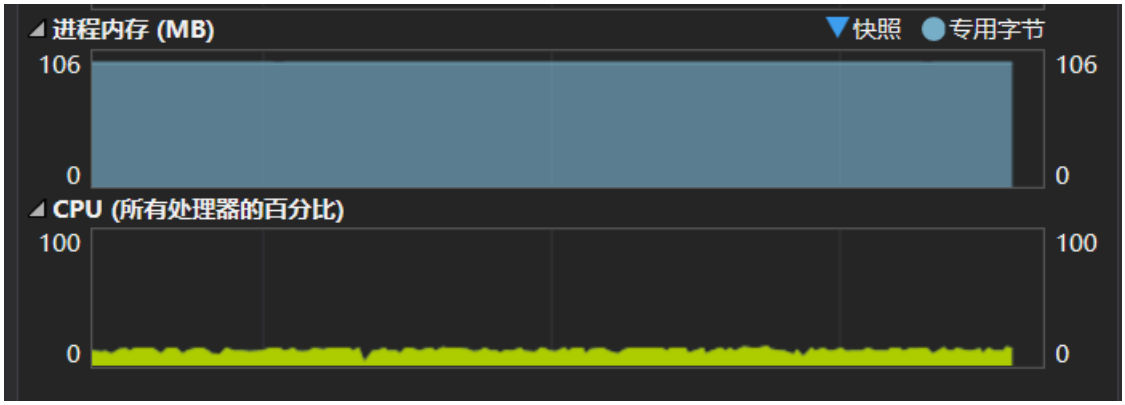

图 17 内存占用情况二

3. 20 亿个数字测试

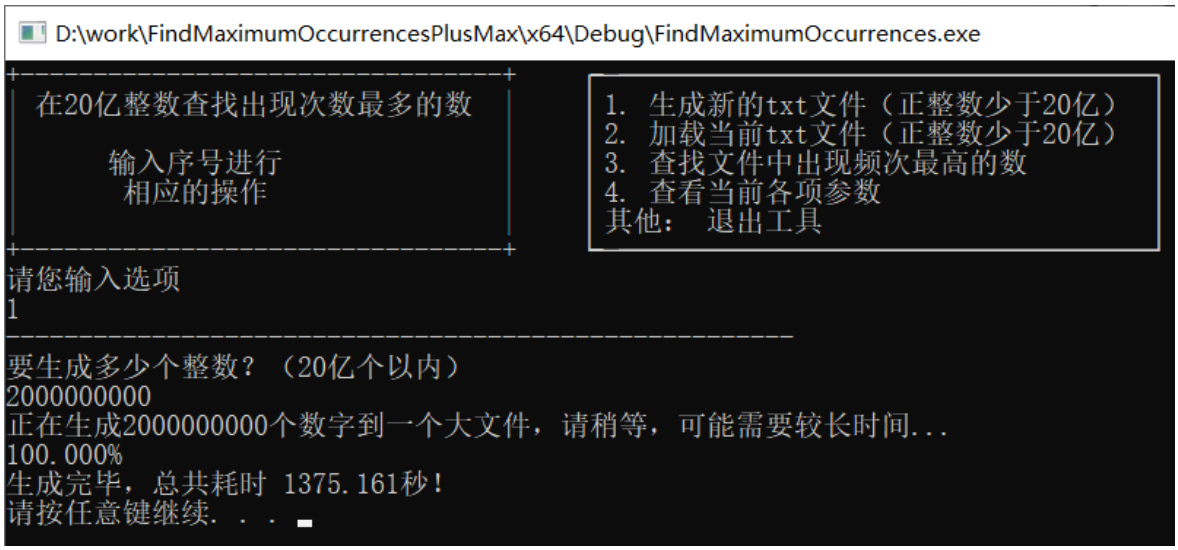

图 16 加载结果三

测试结果和内存占用情况见图 18、19。

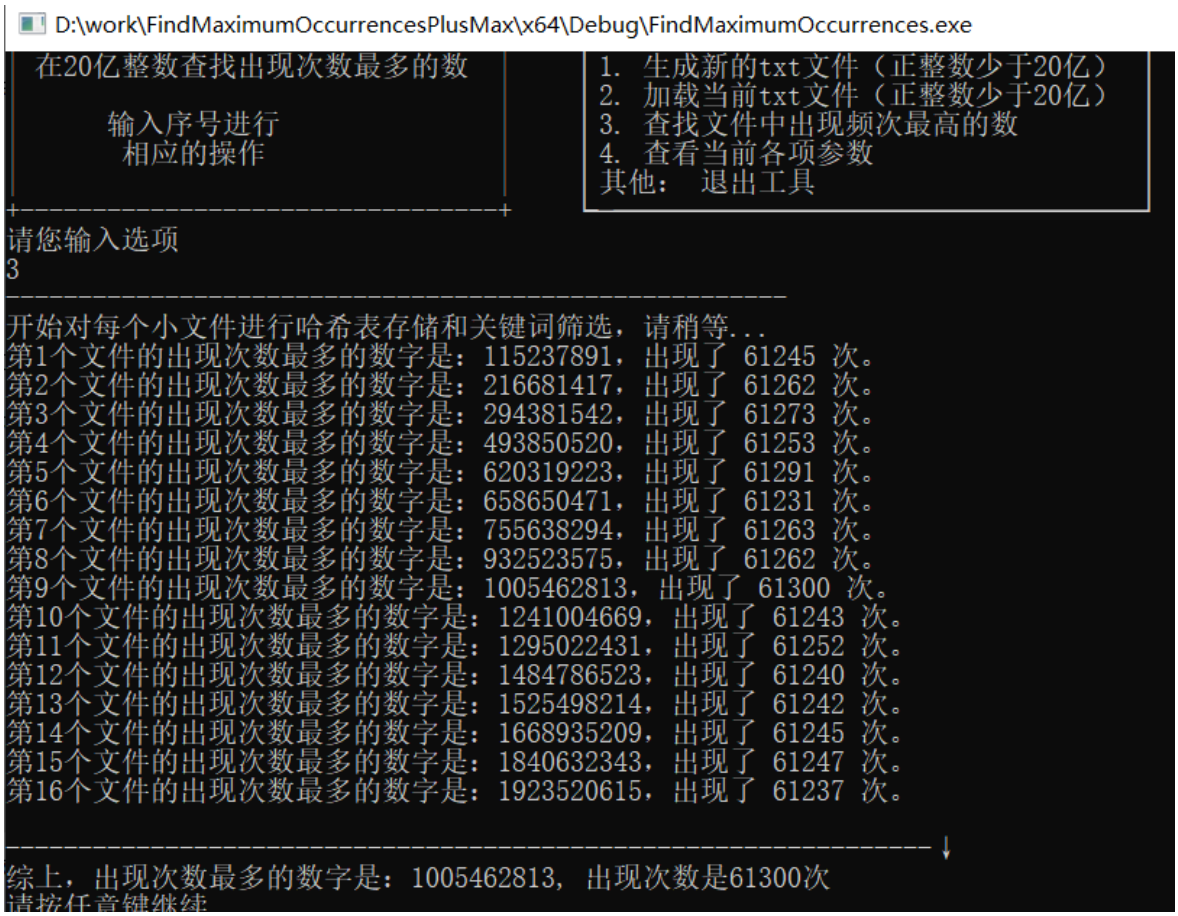

图 18 测试结果三

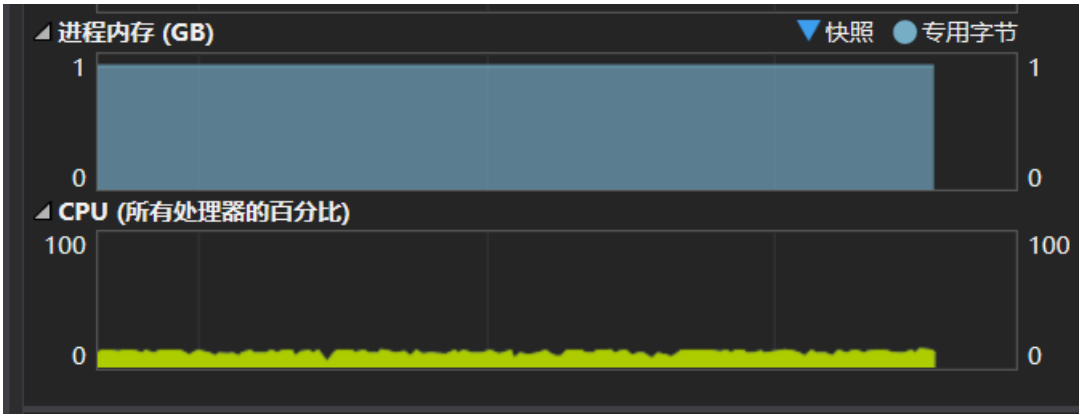

图 19 内存占用情况三

结束语

本次数据结构课程设计耗时一周，尽管途中遇到了许多问题，但在指导老师、网络资料和同学们的热情帮助下，这些问题都得以成功的解决，并且学习了许多新的知识点和巩固了旧的基础。在几天的写代码和测试后，本人制作出了完成度较高的成品。该制作成品考虑了许多异常处理和用户体验，并且代码配有大量注释，又分了模块，简洁易懂，分类明确，能够达到课程设计题目的基本要求。但美中不足在于读取亿级数字的时候，读取的时间还是有些偏长，也就是在性能方面存在着一些缺陷。如今现代社会的快节奏必然要求高效率的读取和分析，以尽快的给出结果，因此在这一方面还是亟待改进的。在改进读取和分析效率方面，本人曾通过改进算法，提高了 100 倍左右的读取效率，但在面对上亿级别的数据时，读取速度还是偏慢，分析 20 亿个数据居然要有接近 1 小时的时间开销。本人曾考虑多线程并行处理，但由于掌握水平不足，导致在实践中没能真正改善效率，这也暴露出了我在这块知识点的掌握程度不足，需要在这方面加深学习。而对此方面的深入了解，还需要认真学习此后的计算机专业课程例如计算机组成原理和操作系统等。总的来说，这次课程设计将上课时学到的各种数据结构知识点融会贯通，很好的加深了本人对数据结构和算法的认识，提高了本人对数据结构和算法的掌握能力和技能水平，而且对于内存管理机制的了解也比以往更加深刻，本人也意识到，在应对各种实际问题时，选用巧妙的数据结构和算法不仅能够大幅提高程序的效率，也能使程序的代码可读性提高，变得更易维护、有高健壮性。

参考资料

[1] 严蔚敏，吴伟民．数据结构（C 语言版）．北京：清华大学出版社，1997.

附录

本程序分为头文件和源文件两部分，头文件包含 animation.h（用于系统美化和异常处理）、HashMap.h（哈希表的数据结构及函数声明）。源文件包含 Animation.cpp、main.cpp（主程序）和 HashMap.cpp。

在 Windows 10 系统，Visual Studio 2017 编译通过。

头文件：

HashMap.h

```
#pragma once
#include<iostream>


/* 数字的个数 */
extern int maxNum;

/* 哈希映射的值分块放到的文件的个数 */
extern int filesNum;

/* 每个小哈希表的地址个数 */
extern int addrNum;

using namespace std;

/* 链表的每个结点 */
typedef struct HashNode {
    /* 关键字 */
    int key;
    /* 记录出现频次 */
    int value;
    /* 同地址的下一个关键字结点 */
    HashNode *next;
};

/* 数组链表 */
typedef struct
{
    HashNode* firstout;

}*HashList, HashElem;

/* 整个哈希表 */
```

```
34. typedef struct HashMap {
35.     HashList adr;
36.     /* 表长 */
37.     int num;
38. }HashMap;
39.
40.
41. /* 初始化哈希表 */
42. void InitHashMap(HashMap *hashmap);
43.
44. /* 哈希函数 */
45. int Hash(int key);
46.
47. /* 将大文件的所有数字根据哈希值分配到不同文件中去 */
48. void divideIntoFilesByHash(int key[], int start, int end, FILE* fouts[], int &
    cur);
49.
50. /* 向哈希表中插入关键字 */
51. void add(HashMap *hashmap, int key);
52.
53. /* 返回这个哈希表的最大词频以及最大词频对应的关键词 */
54. void findMaxOccurrence(HashMap *hashmap, int &maxkey, int &maxFrequency);
55.
56. /* 销毁哈希表 */
57. void destory(HashMap *hashmap);
```

animation.h

```
1.  #pragma once
2.
3.  #include<iomanip>
4.  #include<iostream>
5.  #include <stdio.h>
6.
7.  using namespace std;
8.
9.  /* 装饰：进度条动画 */
10. void processAnimation(int process, int end);
11.
12. /* 装饰：菜单界面 */
13. int menuShow();
14.
15. /* 报错处理 */
16. void Exception(int num);
```

源文件：

HashMap.cpp

```
#include"pch.h"

#define _CRT_SECURE_NO_WARNINGS

#include"animation.h"
#include"HashMap.h"


void InitHashMap(HashMap *hashmap)
{
    hashmap->adr = new HashElem[addrNum];
    for (int i = 0; i < addrNum; i++)
    {
        hashmap->adr[i].firstout = NULL;
    }
    hashmap->num = addrNum;
}

/* 哈希函数，除数是表长 */
int Hash(int key)
{
    return key % addrNum;
}



/* 向哈希表中插入关键字 */
void add(HashMap *hashmap, int key)
{
    int address = Hash(key);
    HashNode *p = hashmap->adr[address].firstout;
    HashNode *temp = p;
    /* flag=0 不存在相同关键字 flag=1 存在相同关键字 */
    int flag = 0;
    while (p)
    {
        /* 如果有冲突，则添加该关键词的词频 */
        if (p->key == key)
        {
            p->value++;
            flag = 1;
```

```
42.             break;
43.         }
44.         else
45.         {
46.             temp = p;
47.             p = p->next;
48.         }
49.     }
50.     /* 如果没有冲突，则添加关键词至表尾 */
51.     if (!flag)
52.     {
53.         if (temp == p)
54.         {
55.             p = new HashNode;
56.             p->next = hashmap->adr[address].firstout;
57.             hashmap->adr[address].firstout = p;
58.         }
59.         else
60.         {
61.             p = new HashNode;
62.             p->next = temp->next;
63.             temp->next = p;
64.         }
65.         p->value = 1;
66.         p->key = key;
67.     }
68. }
69.
70.
71.
72. void divideIntoFilesByHash(int key[], int start, int end, FILE* fouts[], int &
    cur)
73. {
74.     while (start < end)
75.     {
76.         if (key[start] < 0)
77.         {
78.             break;
79.         }
80.         else
81.         {
82.             int period = addrNum;
83.
84.             /* !不能使用 hash 函数，因为这个是对所有数据的哈希映射，而不是对小文件数据
    的哈希映射 */
```

```
85.             int value = key[start] % maxNum;
86.
87.             int order = value / period;
88.
89.             /* 将哈希值进行划分，每个文件存储一定区间的哈希值对应的关键字 */
90.             fwrite(&key[start], sizeof(int), 1, fouts[order]);
91.             processAnimation(cur++, maxNum);
92.             start++;
93.         }
94.
95.     }
96.
97. }
98.
99. /* 返回这个哈希表的最大词频以及最大词频对应的关键词 */
100. void findMaxOccurrence(HashMap *hashmap, int &maxkey, int &maxFrequency)
101. {
102.     maxkey = -1;
103.     maxFrequency = -1;
104.     for (int i = 0; i < hashmap->num; i++)
105.     {
106.         HashNode *p = hashmap->adr[i].firstout;
107.         while (p)
108.         {
109.             if (p->value > maxFrequency)
110.             {
111.                 maxkey = p->key;
112.                 maxFrequency = p->value;
113.             }
114.             p = p->next;
115.         }
116.     }
117. }
118.
119. void destory(HashMap *hashmap)
120. {
121.     for (int i = 0; i < hashmap->num; i++)
122.     {
123.         HashNode *p = hashmap->adr[i].firstout;
124.         HashNode *q = p;
125.         while (p)
126.         {
127.             q = p;
128.             p = p->next;
129.             delete q;
```

```
        }
    }
    delete[] hashmap->adr;
    delete hashmap;
}
```

main.cpp

```
#include "pch.h"

/* 忽略 VS 的安全警告 */
#define _CRT_SECURE_NO_WARNINGS

#include "HashMap.h"
#include"animation.h"
#include <stdio.h>
#include <io.h>
#include <fstream>
#include <iostream>
#include <direct.h>
#include <random>
#include <string>
#include<Windows.h>


using namespace std;

/* 数字的个数 */
int maxNum = 0;

/* 哈希映射的值分块放到的文件的个数 */
int filesNum = 0;

/* 每个小哈希表的地址个数 */
int addrNum = 0;


void createNewFile()
{
    int n;
    cout << "要生成多少个整数？（20 亿个以内）" << endl;
    cin >> n;
    if (n <= 0) { Exception(1); return; }
```

```
37.
38.     /* 更新全局变量 */
39.     maxNum = n;
40.
41.     /* 计时器 */
42.     double start = clock();
43.
44.     /* 伪随机数随机种子 */
45.     srand((unsigned)time(NULL));
46.     cout << "正在生成" << maxNum << "个数字到一个大文件，请稍等，可能需要较长时
    间..." << endl;
47.
48.     FILE *fout = fopen("numbers.txt", "wb");
49.     if (!fout) { Exception(5); return; }
50.
51.     /* 随机生成 maxNum 个数 */
52.     for (int i = 1; i <= maxNum; i++)
53.     {
54.
55.         int temp = (int)((double)rand() / RAND_MAX * maxNum);
56.         fwrite(&temp, sizeof(int), 1, fout);
57.
58.         /* 模拟进度条，方便查看进度 */
59.         processAnimation(i, maxNum);
60.     }
61.     fclose(fout);
62.
63.     cout << endl << "生成完毕，总共耗
    时 " << (clock() - start) / CLOCKS_PER_SEC << "秒！" << endl;
64.     system("pause");
65.
66. }
67.
68. /* 加载大文件，同时通过哈希映射到若干文件上 */
69. void loadFile()
70. {
71.     FILE *fin;
72.     fin = fopen("numbers.txt", "rb");
73.     if (!fin) { Exception(5); return; }
74.
75.     cout << "将关键词分配到多少个文件中？" << endl;
76.     int files;
77.     cin >> files;
78.
79.     /* 异常处理 */
```

```
80.     if (files <= 0) { Exception(1); return; }
81.     if (maxNum%files != 0) { Exception(3); return; }
82.
83.     /* 获得文件字节大小（end)以及数字个数（len） */
84.     fpos_t end;
85.     fpos_t now = -1;
86.     fseek(fin, 0, SEEK_END);
87.     fgetpos(fin, &end); //获取文件大小
88.     long int len = end / sizeof(int);
89.     rewind(fin);
90.
91.     /* 修正数字个数，文件个数和小哈希表表长 */
92.     maxNum = len;
93.     if (!maxNum) { Exception(2); return; }
94.     filesNum = files;
95.     addrNum = maxNum / filesNum;
96.
97.     cout << "开始加载大文件，需要较长时间，请稍等..." << endl;
98.     double start = clock();
99.
100.     /* 规定从缓冲区读取的字节个数 */
101.     const int buffnum = 1024;
102.     int buff[buffnum];
103.
104.     char filePath[20];
105.
106.     FILE** fouts = new FILE*[filesNum];
107.
108.     /* 预载所有文件指针，避免频繁开关输出流 */
109.     for (int order = 0; order < filesNum; order++)
110.     {
111.         sprintf(filePath, "Collection\\%d.txt", order);
112.         remove(filePath);
113.         fouts[order] = fopen(filePath, "ab+");
114.     }
115.
116.     int cur = 0;
117.     while (now!=end)
118.     {
119.         fread(buff, sizeof(int) * buffnum, 1, fin);
120.         fgetpos(fin, &now);
121.         /* 进行哈希映射并分到各个小文件 */
122.         divideIntoFilesByHash(buff, 0, buffnum, fouts, cur);
123.     }
124.
```

```
125.     cout << "成功通过哈希函数映射值将关键字分配到" << filesNum << "个文件中,, 总共耗时 " << (clock() - start) / CLOCKS_PER_SEC << "秒! " << endl;
126.
127.     /* 关闭所有文件指针 */
128.     for (int order = 0; order < filesNum; order++)
129.     {
130.         fclose(fouts[order]);
131.     }
132.     fclose(fin);
133.
134.     system("pause");
135. }
136.
137. /* 返回出现次数最多的 数字数组 和 频次数组 的下标 */
138. int getMax(int *maxfrequency, int *maxkeys)
139. {
140.     int max = -1;
141.     int cur = -1;
142.     for (int i = 0; i < filesNum; i++)
143.     {
144.         if (max < maxfrequency[i])
145.         {
146.             cur = i;
147.             max = maxfrequency[i];
148.         }
149.     }
150.     return cur;
151.
152. }
153.
154.
155. /* 在所有哈希表中寻找出现频次最多的关键词 */
156. void findMaxOccurrences()
157. {
158.     /* 如果没有加载过文件 */
159.     if (_access("Collection\\0.txt", 0) == -1) { Exception(4); return; }
160.     if (!addrNum || !filesNum) { Exception(2); return; }
161.
162.     /* 每个文件出现次数最多的关键词数组以及它们对应的出现频次 */
163.     int *maxkeys = new int[filesNum];
164.     int *maxfrequency = new int[filesNum];
165.
166.     cout << "开始对每个小文件进行哈希表存储和关键词筛选，请稍等..." << endl;
167.     for (int i = 0; i < filesNum; i++)
168.     {
```

```
169.         HashMap *hashmap = new HashMap;
170.         InitHashMap(hashmap);
171.
172.         char filePath[20];
173.         sprintf(filePath, "Collection\\%d.txt", i);
174.
175.         FILE *fin = fopen(filePath, "rb");
176.
177.         int key;
178.
179.         fseek(fin, 0, SEEK_END);
180.         int end = ftell(fin);
181.         rewind(fin);
182.
183.         int n = 0;
184.         while (ftell(fin) != end)
185.         {
186.             processAnimation(n++, addrNum);
187.             fread(&key, sizeof(int), 1, fin);
188.             add(hashmap, key);
189.         }
190.         fclose(fin);
191.
192.         maxfrequency[i] = -1;
193.         /* 获取这个文件的频次最高关键词及其频次 */
194.         findMaxOccurrence(hashmap, maxkeys[i], maxfrequency[i]);
195.         if (maxfrequency[i] != -1)
196.         {
197.             cout << "第" << i + 1 << "个文件的出现次数最多的数字是:
    " << maxkeys[i] << ", 出现了 " << maxfrequency[i] << " 次。" << endl;
198.         }
199.         else
200.         {
201.             cout << "第" << i + 1 << "个文件没有数字存储。" << endl;
202.         }
203.         destory(hashmap);
204.     }
205.
206.     /* 移除文件 */
207.     for (int i = 0; i < filesNum; i++)
208.     {
209.         char filePath[20];
210.         sprintf(filePath, "Collection\\%d.txt", i);
211.         remove(filePath);
212.     }
```

```
213.
214.     int maxcur = getMax(maxfrequency, maxkeys);
215.     cout << endl << "-----------------------------------------------------
    --------↓" << endl;
216.     cout << "综上，出现次数最多的数字是：" << maxkeys[maxcur] << "，出现次数是
   " << maxfrequency[maxcur] << "次" << endl;
217.     system("pause");
218. }
219.
220. void inquire()
221. {
222.     cout << "当前的各项参数如下：" << endl;
223.     cout << "生成的数字个数：";
224.     if (!maxNum) { cout << "未初始化" << endl; }
225.     else { cout << maxNum << endl; }
226.     cout << "分配的文件个数：";
227.     if (!filesNum) { cout << "未初始化" << endl; }
228.     else { cout << filesNum << endl; }
229.     cout << "每个小哈希表的表长：";
230.     if (!addrNum) { cout << "未初始化" << endl; }
231.     else { cout << addrNum << endl; }
232.     system("pause");
233. }
234.
235. void manualSetting()
236. {
237.     cout << "请输入已存在的大文件的数字个数" << endl;
238.     int num;
239.     cin >> num;
240.     if (num <= 0) { Exception(1); return; }
241.     maxNum = num;
242.     cout << "设置成功！" << endl;
243.     system("pause");
244. }
245.
246.
247. //case 5:inquire(); break;
248. int main()
249. {
250.     while (1)
251.     {
252.         int opt = menuShow();
253.         switch (opt)
254.         {
255.         case 1:createNewFile(); break;
```

```
256.         case 2:loadFile(); break;
257.         case 3:findMaxOccurrences(); break;
258.         case 4:inquire(); break;
259.         default:exit(0); break;
260.
261.         }
262.     }
263.
264. }
```

Animation.cpp

```
1.  #include "pch.h"
2.  #define _CRT_SECURE_NO_WARNINGS
3.  #include"animation.h"
4.
5.
6.
7.
8.  void processAnimation(int process, int end) {
9.      if (process % 1000 == 0)
10.     {
11.         cout << setiosflags(ios::fixed) << setprecision(3) << (double(process)
    / end) * 100 << "%";
12.         cout << "\r";
13.     }
14. }
15.
16. int menuShow()
17. {
18.     system("cls");
19.
20.     cout << "+---------------------------------+    ┌-———————————————————————
    ————————————┐\n";
21.     cout << "| 在 20 亿整数查找出现次数最多的数  |    | 1. 生成新的 txt 文件（正整数少
    于 20 亿） |\n";
22.     cout << "|                                 |    | 2. 加载当前 txt 文件（正整数
    少于 20 亿） |\n";
23.     cout << "|      输入序号进行               |    | 3. 查找文件中出现频次最高的
    数         |\n";
24.     cout << "|       相应的操作                |    | 4. 查看当前各项参
    数                  |\n";
25.     cout << "|                                 |    | 其他： 退出工
    具                      |\n";
```

```
26.     cout << "+--------------------------------+      └──────────────────────
    ──────────────┘\n";
27.     cout << "请您输入选项" << endl;
28.     int num;
29.     cin >> num;
30.     cout << "------------------------------------------------------" << endl;

31.     return num;
32. }
33.
34.
35. void Exception(int num) {
36.     if (num == 1)
37.     {
38.         cout << "输入的数字必须是正整数." << endl;
39.     }
40.     else if (num == 2)
41.     {
42.         cout << "分配的文件个数或生成的数字个数还没有初始化." << endl;
43.     }
44.     else if (num == 3)
45.     {
46.         cout << "由于工具限制，该文件数不能使关键词全部分配到这些文件中去，请重新输
    入." << endl;
47.         cout << "提示：能被生成的数字个数整除即可." << endl;
48.         cout << "返回开始菜单." << endl;
49.     }
50.     else if (num == 4)
51.     {
52.         cout << "请先加载大文件，然后再进行查询." << endl;
53.     }
54.     else if (num == 5)
55.     {
56.         cout << "文件创建/打开失败." << endl;
57.     }
58.     system("pause");
59. }
```