山东大学 计算机科学与技术 学院

数据结构与算法 课程实验报告

学号：202300130183	姓名： 宋浩宇	班级：23 级人工智能班
实验题目：2024 数据结构--数据/智能 实验 7 队列		
实验学时：2	实验日期：2024/10/30	

实验目的：

1. 掌握队列结构的定义与实现；
2. 掌握队列结构的使用。

软件开发工具：

1. visual studio code 2022（使用 C/C++、C/C++ Extension Pack、C/C++ Themes 插件）
2. mingw64 工具包

1. 实验内容

完成 2024 数据结构--数据/智能 实验 7 队列 A 卡片游戏

实现一个队列 queue 类，并实现其中的 pop、push、top 方法，解决这个实验的算法问题。

2. 数据结构与算法描述 （整体思路描述，所需要的数据结构与算法）

本题使用的数据结构为队列，这个题相对比较简单，并没有什么特别需要优化的地方，如果按照常规思路来解决这个问题（此处只讨论常规思路即模拟），主要分歧点在于队列类是按照数组来实现还是按照链表来实现。在此两种方式都会进行介绍，且在分析与探讨板块讨论非常规解法。首先按照链表的方式来实现，是相当简单的，我们只需要将链表原来的代码进行修改，将原本链表的 pop_front 方法改写为 pop，将 push_back 方法改写为 push 即可，关于 top 方法，只需要对原本链表结构中的头指针进行一此解引用即可。按照上述思路，只需要对链表的代码进行相当少的改动即可获得一个队列类，其中链表节点的头指针是使用不到的，可以删除/注释掉这一部分代码来减少无意义的性能消耗。而关于数组实现则有些需要注意的地方，首先在一个线性容器上，我们进行 pop 的最快的方法是直接将队列开始位置（用 front 表示）的下标加一，我们进行 push 的最快的方法是将队列末尾（用 tile 表示）的下标加以后再对对应位置赋值。此处会牵扯到一个队列容量的问题，我们在 push 的时候，如果当前的末尾下标加一之后和容量的大小相等了，我们就扩大这个队列的内存空间，我们只需要创建一个容量翻倍之后的空间，再将这个队列中的 front 到 tile 之间的数据复制过去，并将 front 重置为 0，tile 重置为 tile-front（同时也是队列中元素的个数）即可，但这种方式其实是在不知道后续 push 的时候会有多少个元素才这样做的，对于本题来讲，原本的线性空间就够用了，我们只需要把原本空间的 0 到 tile - front 重新赋值即可将队列搬运过去。当我们实现了这个队列类之后，只需要使用模拟的方式就可以解决这个问题了，具体模拟过程为，在队列的卡牌的数量大于 1 的时候，执行一次 pop()，再执行一次 push(pop())即可。

3. 测试结果（测试输入，测试输出）

测试输入为：

100

测试输出为：

72

4. 分析与探讨（结果分析，若存在问题，探讨解决问题的途径）

就我们的测试结果来看，我们成功解决了这个问题，但实际上这个问题除了通过常规思路，也就是队列模拟这个思路之外，还可以找其中的规律来解决。是的，我们可以将这个题作为一个数学题来做。实际上我们可以观察到，我们将从这堆卡牌顶遍历到原来的卡牌底称为一轮，我们这对卡牌每操作一轮，其中下标为偶数的卡牌会被丢弃，如果卡牌原本的张数为偶

数，这会是一个非常简单的过程，而即使卡牌的张数是奇数，这个过程也并不复杂，实际上只是在每一轮多执行一次操作，即将一轮执行完之后的头放到尾。因此，我们可以简化这个计算过程，将原本 O（n）的问题转化为 O（logn）的问题，实际上，按照这种思路去做，这个题的用时会短很多很多，且仅使用数组就可以解决，但这样就不符合我们的实验要求了，实际还是书写一个队列类模板来解决问题。与栈面临的问题是一致的，就是我们的队列类的内存管理存在问题，缺少 throw-catch 结构来解决程序运行中的问题。解决思路也和之前是一致的，如果使用数组结构，那就在 pop 时手动调用一次对象的析构函数，如果使用链表结构，就及时的进行 delete 操作，且使用智能指针来存储索引。

5. 附录：实现源代码（本实验的全部源程序代码，程序风格清晰易理解，有充分的注释）

本附录分为两个部分，第一部分为数组描述的队列代码，第二部分为链表描述的队列代码

一、数组描述队列

```cpp
/*2024 级数据结构--数据智能 实验 7 队列 A 卡片游戏 数组描述.cpp*/
#include <iostream>

using namespace std;

template<class T>
class queue
{
private:
    T* data;
    int front; //当前的头
    int tile;  //当前的最后一个的下一个
    int capacity;
public:
    queue(int capacity) {
        this->capacity = capacity;
        data = new T[capacity];
        front = 0;
        tile = 0;
    }
    ~queue() {
        delete[] data;
    }
    void push(T&& a_data) {
        if (tile == capacity) {
            T* newData = new T[capacity * 2];

            for (int i = front; i < tile; i++) {
                newData[i - front] = this->data[i];
            }
            delete[] data;
            tile -= front;
            data = newData;
            front = 0;
            capacity *= 2;
```

```
            }
            data[tile++] = a_data;
        }
        void push(T& a_data)
        {
            if (tile == capacity) {
            T* newData = new T[capacity * 2];

            for (int i = front; i < tile; i++) {
                newData[i - front] = this->data[i];
            }
            delete[] data;
            tile -= front;
            data = newData;
            front = 0;
            capacity *= 2;
        }
            data[tile++] = a_data;
        }
        T pop() {
            return data[front++];
        }

        int size() {
            return tile - front;
        }
    };


    class Solution
    {
    public:
        void solve();
    };


    void Solution::solve()
    {
        queue<int> q(1000);
        int n;
        cin >> n;
        for (int i = 0; i < n; i++) {
            q.push(i + 1);
        }
        while (q.size() > 1) {
```

```
            q.pop();
            q.push(q.pop());
        }
        cout << q.pop();
    }


    int main()
    {
        Solution s;
        s.solve();
        return 0;
    }
```

二、链表描述队列

```
    /*2024级数据结构--数据智能 实验7 队列 A 卡片游戏.cpp*/
    #include <iostream>

    using namespace std;

    template<class T>
    class list_node
    {
    private:
        T data;

    public:
        list_node<T>* front;
        list_node<T>* next;
        bool operator==(const list_node<T>& list_node) const { return this->data == list_node.data; };
        T& getData() { return data; };
        list_node<T>* getFront() { return front; };
        list_node<T>* getNext() { return next; };
        list_node(){};
        list_node(const T& data, list_node<T>* front = nullptr, list_node<T>* next = nullptr);
        list_node(list_node<T>* const list_node) { this->data = list_node->data;this->front = list_node->front;this->next = list_node->next; };
    };

    template<class T>
    list_node<T>::list_node(const T& data,list_node<T>* front,list_node<T>* next)
    :data(data),front(front),next(next)
    {
```

```
28.
29.  }
30.
31.  template<class T>
32.  class queue
33.  {
34.  private:
35.      list_node<T>* head;
36.      list_node<T>* tail;
37.      int count;
38.  public:
39.      queue() { head = nullptr; tail = nullptr;count = 0; };
40.      bool empty() { return count == 0; };
41.      void push(T& data);
42.      void push(T&& data);
43.      void push(list_node<T>& data);
44.      void pop_free();
45.      T& pop();
46.      T& front();
47.      list_node<T>& pop_node();
48.      void print();
49.      int size() { return count; };
50.  };
51.
52.  template<class T>
53.  void queue<T>::push(T&& data)
54.  {
55.      list_node<T>* new_node = new list_node<T>(data);
56.      if (empty()) {
57.          head = new_node;
58.          tail = new_node;
59.      }
60.      else
61.      {
62.          tail->next = new_node;
63.          tail = new_node;
64.      }
65.      count++;
66.  }
67.
68.  template<class T>
69.  list_node<T>& queue<T>::pop_node()
70.  {
71.      if (empty())
72.      {
73.          return *(new list_node<T>());
```

```
74.         }
75.         else
76.         {
77.             count--;
78.             list_node<T>& data = *head;
79.             head = head->getNext();
80.             return data;
81.         }
82.     }
83.
84.     template<class T>
85.     void queue<T>::push(T& data)
86.     {
87.         list_node<T>* new_node = new list_node<T>(data);
88.         if (empty()) {
89.             head = new_node;
90.             tail = new_node;
91.         }
92.         else
93.         {
94.             tail->next = new_node;
95.             tail = new_node;
96.         }
97.         count++;
98.     }
99.
100.    template<class T>
101.    void queue<T>::push(list_node<T>& data)
102.    {
103.        if (empty()) {
104.            head = &data;
105.            tail = &data;
106.        }
107.        else
108.        {
109.            tail->next = &data;
110.            tail = &data;
111.        }
112.        count++;
113.    }
114.
115.    template<class T>
116.    T& queue<T>::pop()
117.    {
118.        if (empty())
119.        {
```

```
120.         return *(new T(0));
121.     }
122.     else
123.     {
124.         count--;
125.         T& data = head->getData();
126.         head = head->getNext();
127.         return data;
128.     }
129. }
130. 
131. 
132. template<class T>
133. void queue<T>::print()
134. {
135.     list_node<T>* temp = head;
136.     while (temp!= nullptr)
137.     {
138.         cout << temp->getData() << " ";
139.         temp = temp->getNext();
140.     }
141.     cout << endl;
142. }
143. template<class T>
144. void queue<T>::pop_free()
145. {
146.     if (empty())
147.     {
148.         return;
149.     }
150.     else
151.     {
152.         count--;
153.         list_node<T>* temp = head;
154.         head = head->getNext();
155.         // temp.~list_node<T>();
156.         delete temp;
157.     }
158. }
159. 
160. template<class T>
161. T& queue<T>::front()
162. {
163.     if (empty())
164.     {
165.         return *(new T(0));
```

```
166.
167.     }
168.     else
169.     {
170.         return head->getData();
171.     }
172. }
173.
174. class Solution
175. {
176. public:
177.     void solve();
178. };
179.
180. void Solution::solve()
181. {
182.     queue<int> q;
183.     int n;
184.     cin >> n;
185.     for (int i = 0; i < n; i++)
186.     {
187.         q.push(i + 1);
188.     }
189.     while (q.size() > 1)
190.     {
191.         q.pop_free();
192.         // int temp = q.pop();
193.         // q.push(temp);
194.         q.push(q.pop_node());
195.     }
196.     cout << q.front();
197. }
198.
199.
200. int main()
201. {
202.     Solution solution;
203.     solution.solve();
204.     return 0;
205. }
```