数据结构：祖玛

姓名：鲁国锐　　学号：17020021031　　专业：电子信息科学与技术

2019 年 3 月 15 日

Contents

1 问题分析

1.1 题目描述

祖玛是一款曾经风靡全球的游戏，其玩法是：在一条轨道上初始排列着若干个彩色珠子，其中任意三个相邻的珠子不会完全同色。此后，你可以发射珠子到轨道上并加入原有序列中。一旦有三个或更多同色的珠子变成相邻，它们就会立即消失。这类消除现象可能会连锁式发生，其间你将暂时不能发射珠子。

开发商最近准备为玩家写一个游戏过程的回放工具。他们已经在游戏内完成了过程记录的功能，而回放功能的实现则委托你来完成。

游戏过程的记录中，首先是轨道上初始的珠子序列，然后是玩家接下来所做的一系列操作。你的任务是，在各次操作之后及时计算出新的珠子序列。

1.1.1 输入

第一行是一个由大写字母 A Z 组成的字符串，表示轨道上初始的珠子序列，不同的字母表示不同的颜色。

第二行是一个数字 n，表示整个回放过程共有 n 次操作。

接下来的 n 行依次对应于各次操作。每次操作由一个数字 k 和一个大写字母 $\sum$ 描述，以空格分隔。其中，$\sum$ 为新珠子的颜色。若插入前共有 m 颗珠子，则 $k \in [0, m]$ 表示新珠子嵌入之后（尚未发生消除之前）在轨道上的位序。

1.1.2 输出

输出共 n 行，依次给出各次操作（及可能随即发生的消除现象）之后轨道上的珠子序列。

如果轨道上已没有珠子，则以“−”表示。

1.1.3 样例

- 输入

  ```
  ACCBA
  5
  1 B
  0 A
  2 B
  4 C
  0 A
  ```

- 输出

  ```
  ABCCBA
  AABCCBA
  AABBCCBA
  -
  A
  ```

1.2 问题分析

根据题目，我们需要解决的问题有：

1. 如何在插入后找出字符串中三个以上连续字符的位置；
2. 如何在进行消除后找出字符串中三个以上连续字符的位置；

至于消除的问题可以直接用模板库的函数来解决，所以不在我们要考虑的范围之内。

2 解决方案

首先这道题需要注意的一点是，**给定的输入中可能有三个以上连续重复的字符**，但根据以往玩游戏的经验，它并不会自动消除，而是需要人为地往里面再添加一个相同字符。所以我们不能每次都对整个字符串进行遍历。事实上，我们也不需要对整个字符串遍历，因为每次可能会出现消除的地方只会是插入后的位置以及消除后的“接口”位置。再进一步考虑，由于消除后后面的元素会向前填补，所以插入位置和消除后的“接口”处可以用同一个下标来表示。所以我们只用反复地考查插入位置附近是否有三个及以上连续相同字符并在符合条件的情况下进行消除。

由此我们引出了解决本题时需要实现的两个函数：一个是 *find* 函数，用于考察插入位置附近是否有连续相同字符，返回该字符串的终止位置；一个是 *ablat*，用于在符合条件时对 find 函数找出的字符串进行消除，并返回找出的字符串是否符合条件以确定是否需要再次调用 *find* 函数进行考察。

首先我们来看一看 *find* 函数。为了实现其功能，我们需要对其传入参数插入位置的下标 *start* 以及目标字符串 *balls*。我们再定义两个变量 *left* 和 *right*，分别用来表示连续相同字符序列的起始下标和**终止下标的下一位**。我们先从 *start* 开始向左遍历，如果前一个元素等于当前元素，则令 *left* 减一；再从 *start* 开始向右遍历，如果下一个元素等于当前元素，则令 *right* 加一。注意不论是向左还是向右遍历，都不能超出字符串边界。另外还有一个小问题，我们调用 *find* 函数是为了找出满足条件的子序列的两头，我们可以直接返回 *right* 来输出末尾的下标，但起始的怎么办？如果是用一个数组来存放两个下标未免太不划算，所以这里我们在传参时直接传入 *start* 的引用，并在函数结束前把 *left* 的值赋给 *start* 即可。

接下来我们再来看看 *ablat* 函数。为了实现其功能，我们需要传入起始下标 *start*、终止下标 *end* 以及目标字符串 *balls*。我们在定义一个变量 *flag*，并令：

$$flag = ((end - start) \geq 3) \tag{1}$$

公式1的目的是用来判断 *find* 函数找出的字符串是否满足条件。如果是，则进行消除。最后再返回 *flag* 给主函数判断此次调用是否进行了消除操作，如果是，则还需调用 *find* 函数再次对 *start* 附近进行考查；反之则进入到下一次的插入操作当中去。

到此算法的主要部分就分析完毕了，其余内容相对简单，在此不加赘述。

另外在具体的实现上，我用了 *vector* 和 *list* 两种模板库进行实现（分别见 list1和 list2）。其中 *vector* 版我参照课本 [1] 实现了它的部分函数，并在清华大学的 *Open Judge* 上提交通过。而 *list* 版还未实现，所以还未经 *Open Judge* 的检验，在此只给出它的代码。

3 算法设计

见下页图1。

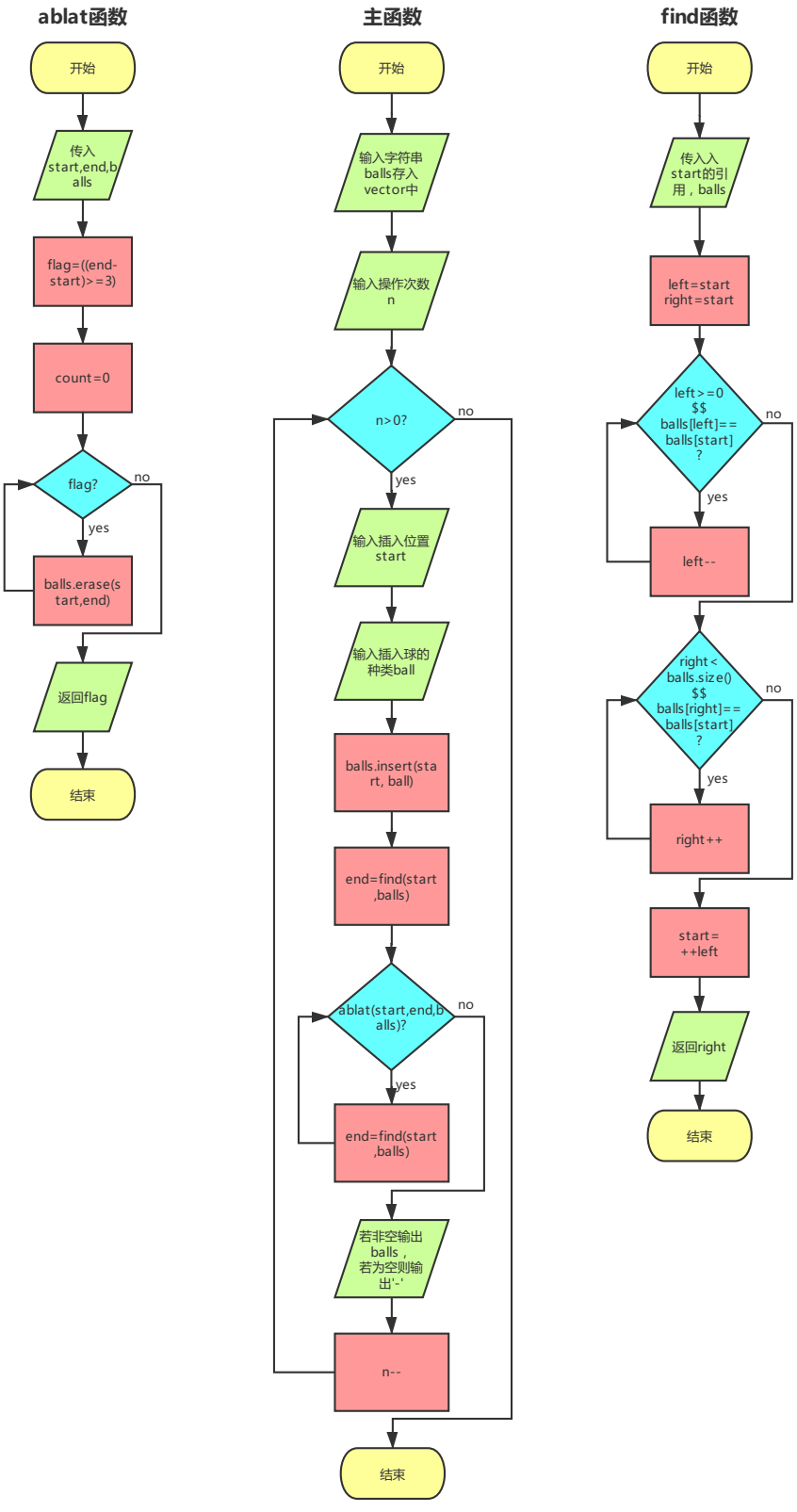

图 1: vector 版流程图

4 编程实现

4.1 *vector* 版

Listing 1: vector 版祖玛代码

```cpp
#include<iostream>
#include<stdio.h>

using namespace std;

// refer to 数据结构（c++语言版）
template <typename T> class vector
{
    int _size;
    int _capacity;
    T *_elem;
    protected:
        void copyFrom(T const *A, int lo, int hi)
        {
            _elem = new T[_capacity = 2 * (hi - lo)];
            _size = 0;
            while (lo < hi)
                _elem[_size++] = A[lo++];
        }
    public:
        vector(int c = 32, int s = 0, T v = 0)
        {
            _elem = new T[_capacity = c];
            for (_size = 0; _size < s; _elem[_size++] = v);
        }
        vector(T const *A, int n) {copyFrom(A, 0, n);}
        vector(T const *A, int lo, int hi) {copyFrom(A, lo, hi);}
        vector(vector<T> const &A) {copyFrom(A._elem, 0, A._size);}

        ~vector() {delete [] _elem;}

        T& operator[] (int r) const {return _elem[r];}


        bool empty() const {return !_size;}
        int size() const {return _size;}
        void expand()
        {
            if (_size < _capacity) return;
            if (_capacity < 32)
                _capacity = 32;
            T *oldElem = _elem;
            _elem = new T[_capacity <<= 1];
```

```
            for (int i = 0; i < _size; i++)
                _elem[i] = oldElem[i];
            delete [] oldElem;
        }
        int insert(int r, T const &e)
        {
            expand();
            for (int i = _size; i > r; i--)
                _elem[i] = _elem[i-1];
            _elem[r] = e;
            _size++;
            return r;
        }
        void push_back(T const &e)
        {
            insert(_size, e);
        }
        void erase(int lo, int hi)
        {
            if (lo == hi) return;
            while (hi < _size)
                _elem[lo++] = _elem[hi++];
            _size = lo;
            return;
        }



};


int find(int &start, vector<char> &balls)
{
    int left = start, right = start;
    while (left >= 0 && balls[left] == balls[start])
        left--;
    while(right < balls.size() && balls[right] == balls[start])
        right++;
    start = ++left;
    return right;
}

bool ablat(int &start, int &end, vector<char> &balls)
{
    bool flag = ((end - start) >= 3);
    int count = 0;
    if (flag)
        balls.erase(start, end);
```

```
92      return flag;
93  }
94
95
96  int main()
97  {
98      vector<char> balls;
99      char c;
100     while (1)
101     {
102         scanf("%c", &c);
103
104         if (c == '\n')
105             break;
106         balls.push_back(c);
107     }
108
109     int n;
110     scanf("%d", &n);
111     while (n > 0)
112     {
113         int start;
114         char ball;
115         cin >> start >> ball;
116         balls.insert(start, ball);
117
118         int end = find(start, balls);
119
120         while (ablat(start, end, balls))
121         {
122             end = find(start, balls);
123         }
124
125         if (balls.empty())
126             printf("-");
127         else
128             for (int i = 0; i < balls.size(); i++)
129                 printf("%c", balls[i]);
130         printf("\n");
131         n--;
132     }
133
134     return 0;
135 }
```

4.2 *list* 版

Listing 2: list 版祖玛代码

```cpp
#include<iostream>
#include<list>

using namespace std;

list<char>::iterator find(list<char>::iterator &start, list<char> &balls)
{
    list<char>::iterator left = start, right = start;
    while (left != balls.begin() && *left == *start)
        left--;
    while (right != balls.end() && *right == *start)
        right++;

    if (left == balls.begin() && *left == *start)
        start = balls.begin();
    else
        start = ++left;
    return right;

}


bool ablat(list<char>::iterator &start, list<char>::iterator &end, list<char> &balls)
{
    bool flag = false;
    int count = 0;
    for (list<char>::iterator i = start; i != end; i++)
        if (++count >= 3)
        {
            flag = true;
            break;
        }
    if (flag)
        while (start != end)
            start = balls.erase(start);

    //if (flag && start != balls.begin())
    if (start == balls.end())
        start--;

    //cout << *start << "   ";
    return flag;
}



int main()
```

```
{
    list<char> balls;
    char c;
    while (1)
    {
        cin.get(c);

        if (c == '\n')
            break;
        balls.push_back(c);
    }

    int n;
    cin >> n;
    while (n > 0)
    {
        int index;
        char ball;
        cin >> index >> ball;
        list<char>::iterator start = balls.begin();
        for (int i = 0; i < index; i++)
            start++;

        balls.insert(start, ball);
        start--;

        list<char>::iterator end = find(start, balls);
        while(ablat(start, end, balls))
        {
            end = find(start, balls);
        }


        if (!balls.size())
            cout << "-";
        else
            for (list<char>::iterator i = balls.begin(); i != balls.end(); i++)
                cout << *i;


        cout << endl;
        n--;
    }

    return 0;
}
```

5 结果分析

- 输入

  ACCBA

  5

  1 B

  0 A

  2 B

  4 C

  0 A

- 输出

  ABCCBA

  AABCCBA

  AABBCCBA

  -

  A

更换多组数据测试均正确，其中 *vector* 版本以 100 分的成绩通过清华大学 *Open Judge* 测试。

6 总结体会

第一次做这道题时我是用 *list* 做的，因为一开始觉得 *list* 删除元素只需常数时间。但在实现过程中发现 *list* 在无法像 *vector* 那样在常数时间内访问任意一个元素，只能通过迭代器一个一个往后找，所以到最后两种实现方式的时间复杂度应该相差不多。但用 *list* 实现由于大量操作都是通过迭代器来实现的且迭代器之间不能进行算数操作，这使得它实现起来要更为复杂一些。

参考文献

[1] 邓俊辉. 数据结构（c++ 语言版）. 清华大学出版社. 3