西安電子科技大學
XIDIAN UNIVERSITY

数据结构第二次上机实验报告

队列与栈

陈德创

19030500217

计算机科学与技术专业

指导教师 王凯东

October 28, 2020

数据结构第二次上机实验报告

陈德创　　19030500217

西安电子科技大学

日期：2020 年 10 月 28 日

目录

1　小组名单 **2**

2　括号匹配 **2**

2.1　题目描述 2

2.2　题目分析 2

2.3　程序设计 2

2.4　程序分析 4

3　颜色置换 **4**

3.1　题目描述 4

3.2　题目分析 4

3.3　程序设计 4

3.4　程序分析 5

4　总结 **6**

附录 **7**

A　括号匹配代码 **7**

B　颜色置换代码 **9**

1 小组名单

学号	姓名	工作
19030500217	陈德创	完成程序、联系例程、调试、小组讨论

2 括号匹配

2.1 题目描述

给定一个字符串，字符串含有三种括号 (‘()’,‘[]’,‘{}’)，也含有其他字符。括号可以任意嵌套，问给定字符串中括号时候匹配。

2.2 题目分析

栈的经典应用。读入一个字符串 *s*。维护一个栈 *st*，按序遍历 *s*，假设当前遍历到的字符为 *c*。如果 *c* 不是括号，跳过该字符，因为非括号字符并不会对括号匹配造成任何影响。如果是括号，那么有两种情况：

1). *c 是左括号*

将 *c* 压入栈 *st* 中，继续遍历。

2). *c 是右括号*

弹出 *st* 栈顶元素，设为 *t*。显然，*t* 是一个左括号。那么如果 *t* 和 *c* 不可以匹配，那么字符串 *s* 中括号不匹配。否则继续遍历。

那么当我们遍历完整个字符串的之后，*st* 可能有两种状态。

1). *st 为空*

奈斯，这个字符串中的括号匹配。

2). *st 非空*

很遗憾，字符串中的括号不匹配，而是如下形式：..(..[...]..{..}..)..({...

那么为什么这样是正确的呢？

我们可以考虑到，匹配成功的括号和非括号字符不会对结果造成影响，所以每次匹配成功，就可以将括号在字符串里删去。那么如果我们将所有非括号字符和匹配成功的字符删去，显然，如果最后串不为空，那么串括号不匹配。

实际上，我们做的就是：

如果当前字符为左括号，那么它是新的待匹配字符；如果是右括号，那么将它和待匹配字符匹配，匹配成功继续迭代；如果不匹配，那这个字符串中的括号不匹配。每次匹配成功，将它们从串中删去。

2.3 程序设计

程序设计主要是栈的设计，本题中栈可以使用最基本的静态栈。

```
struct STACK{
  char x[MAXN]; // 存放数据
  int tot; // 栈顶指针（指向栈顶元素的下一个元素）

  // 构造函数，将数据和栈顶指针清零
  STACK(){
    memset(x, 0, sizeof(x));
    tot = 0;
  }

  // 判空
  inline bool isEmpty(){
    return tot == 0;
  }

  // 元素压栈
  inline void push(char t){
    x[tot++] = t;
  }

  // 返回栈顶元素
  inline int top(){
    return x[tot - 1];
  }

  // 弹栈，并返回弹栈元素
  inline int pop(){
    return x[--tot];
  }
};
```

其次是主逻辑，很简单，按照先前的思路进行模拟即可。

```
cin >> buf;
// 待测字符串是否匹配
bool flag = true;
// buf: 字符串
for (int i = 0; buf[i]; ++i){
  char c = buf[i];
  // 不是括号，跳过这个字符
  if (!isBracket(c)) continue;
  // 如果是左括号，压栈
  if (isLeft(c)) {st.push(c); continue;}
  // 如果是右括号，但是栈空了，那么右括号不匹配
  if (st.isEmpty()){
    flag = false;
    break;
  }
```

```
    // 弹出栈顶元素，看是否匹配
    char t = st.pop();
    if (!matching(c, t)) {
      flag = false;
      break;
    }
  }
  // 如果最后栈不为空，那么左括号不匹配
  flag &= st.isEmpty();
  cout << (flag ? "YES" : "NO") << endl;
```

2.4 程序分析

时空复杂度非常显然，都是 $O(n)$。

3 颜色置换

3.1 题目描述

假设以二维数组 $g(1\ldots m, 1\ldots n)$ 表示一个图像区域，$g[i,j]$ 表示该区域中点 (i,j) 所具有的颜色，其值为从 0 到 k 的整数。编写算法置换点 (i_0, j_0) 所在区域的颜色。约定和 (i_0, j_0) 同色的上、下、左、右的邻接点为同色区域的点。

3.2 题目分析

题目给定了一个颜色矩阵，每次修改相当于修改所有同色的连通块。我们的主要任务就是找到 (i_0, j_0) 点所在的联通块。这可以通过搜索得到。

搜索的基本实现，就是对于当前遍历到的点，暴力枚举它能到达的所有点，我们称之为目标点。然后对于每个目标点，如果其满足题目要求，则继续遍历。搜索根据搜索顺序的不同分为 DFS（$depth-first\ search$ 深度优先搜索）和 (BFS)（$breadth-first\ search$ 广度优先搜索）。二者各有优劣，在本题中我们选用 BFS。

算法简述

算法实现非常简单，用一个队列 q 来储存所有将要遍历的点。当 q 非空的时候，每次从中取出队首元素，依次遍历队首元素的临界点并判断其是否满足题目要求（即是否和 (i_0, j_0) 点颜色相同），如果相同则改变其颜色并将其放入队列中。

考虑到可能有多次修改，我们可以将算法封装成一个函数，每次调用即可。

3.3 程序设计

结构体设计 在题目中我们将需要用到点，所以需要一个结构体储存点。定义结构体 $Point$，具体如下：

```
struct Point{
  // 点的两个坐标
  int x, y;
  // 无参构造和有参构造
  Point():x(0), y(0){}
  Point(int x, int y):x(x), y(y){}
};
```

替换算法的实现 算法思路已经在前面讲过，在此不再复述。队列使用 $c++$ STL 中的 $queue$。

```
// 将(x,y)点所在的区域置换为k颜色
inline void replace(int x, int y, int k){
  // 储存x,y所在区域的颜色
  int c = g[x][y];
  // BFS
  queue<Point> q;
  q.push(Point(x, y));
  while (!q.empty()){
    // 取出队首元素并改变其颜色
    Point p = q.front();
    q.pop();
    g[p.x][p.y] = k;
    // 遍历队首元素的邻接点
    for (int i = 0; i < 4; ++i){
      // 判断是否是同一区域的点
      int tx = dx[i] + p.x, ty = dy[i] + p.y;
      if (tx > 0 && tx <= n && ty > 0 && ty <= n && g[tx][ty] == c)
        q.push(Point(tx, ty));
    }
  }
}
```

注意我们在遍历队首元素邻接点的时候并没有讲四个方向都写在代码中，而是采用了 $dx[], dy[]$ 数组表示方向，$dx[], dy[]$ 数组如下：

```
int dx[] = {0, 0, 1, -1};
int dy[] = {1, -1, 0, 0};
```

在对于矩阵的四方向（或者八方向）遍历时，这样可以极大地简化代码，使程序结构简洁明朗，减少 $debug$ 难度。

3.4 程序分析

时间复杂度 考虑每次修改，最差情况下可能会遍历整个矩阵，所以每次修改的复杂度为 $O(mn)$，假设一共有 q 次修改，所以算法总体复杂度为 $O(qmn)$。

空间复杂度 没什么好说的，$O(mn)$，

4 总结

由于链表的基础（栈和队列实际上相当与对链表的封装），加之采用静态存储而不是动态分配，这次作业完成轻松得多。

这次作业让我对栈和队列性质和使用的理解大大加深。希望再接再厉。

附录

A 括号匹配代码

```cpp
// Bracket_matching.cpp
// Created by RainCurtain on 2020/10/25.
//

#include <iostream>
#include <cstring>
using namespace std;
const int MAXN = 1e5 + 5;

struct STACK{
    char x[MAXN]; // 存放数据
    int tot; // 栈顶指针（指向栈顶元素的下一个元素）

    // 构造函数，将数据和栈顶指针清零
    STACK(){
        memset(x, 0, sizeof(x));
        tot = 0;
    }

    // 判空
    inline bool isEmpty(){
        return tot == 0;
    }

    // 元素压栈
    inline void push(char t){
        x[tot++] = t;
    }

    // 返回栈顶元素
    inline int top(){
        return x[tot - 1];
    }

    // 弹栈，并返回弹栈元素
    inline int pop(){
        return x[--tot];
    }
}st;

inline bool isLeft(char c){
    return c == '{' || c == '(' || c == '[';
```

```
}

inline bool matching(char s, char t){
  if (s == '}' || s == ']' || s == ')') swap(s, t);
    if (s == '{' && t == '}') return true;
    if (s == '[' && t == ']') return true;
    if (s == '(' && t == ')') return true;
    return false;
}

inline bool isBracket(char c){
    return c == '[' || c == ']' || c == '{' || c == '}' || c == '(' || c == ')'
        ;
}

char buf[MAXN];
int main(){
    ios::sync_with_stdio(false);
    cin >> buf;
    // 待测字符串是否匹配
    bool flag = true;
    // buf: 字符串
    for (int i = 0; buf[i]; ++i){
        char c = buf[i];
        // 不是括号，跳过这个字符
        if (!isBracket(c)) continue;
        // 如果是左括号，压栈
        if (isLeft(c)) {st.push(c); continue;}
        // 如果是右括号，但是栈空了，那么右括号不匹配
        if (st.isEmpty()){
            flag = false;
            break;
        }
        // 弹出栈顶元素，看是否匹配
        char t = st.pop();
        if (!matching(c, t)) {
            flag = false;
            break;
        }
    }
    // 如果最后栈不为空，那么左括号不匹配
    flag &= st.isEmpty();
    cout << (flag ? "YES" : "NO") << endl;
    return 0;
}
```

B 颜色置换代码

```cpp
// colors.cpp
// Created by RainCurtain on 2020/10/28.
//

#include <iostream>
#include <queue>
using namespace std;
const int MAXN = 1e3 + 5;
int g[MAXN][MAXN];
int dx[] = {0, 0, 1, -1};
int dy[] = {1, -1, 0, 0};
int n, m;

struct Point{
    // 点的两个坐标
    int x, y;
    // 无参构造和有参构造
    Point():x(0), y(0){}
    Point(int x, int y):x(x), y(y){}
};

// 将(x,y)点所在的区域置换为k颜色
inline void replace(int x, int y, int k){
    // 储存x,y所在区域的颜色
    int c = g[x][y];
    // BFS
    queue<Point> q;
    q.push(Point(x, y));
    while (!q.empty()){
        // 取出队首元素并改变其颜色
        Point p = q.front();
        q.pop();
        g[p.x][p.y] = k;
        // 遍历队首元素的邻接点
        for (int i = 0; i < 4; ++i){
            // 判断是否是同一区域的点
            int tx = dx[i] + p.x, ty = dy[i] + p.y;
            if (tx > 0 && tx <= n && ty > 0 && ty <= n && g[tx][ty] == c)
                q.push(Point(tx, ty));
        }
    }
}

int main(){
    ios::sync_with_stdio(false);
```

```
46      cin >> n >> m;
47      for (int i = 1; i <= n; ++i)
48          for (int j = 1; j <= m; ++j)
49              cin >> g[i][j];
50      int q, x, y, k;
51      cin >> q;
52      while (q--) {
53          cin >> x >> y >> k;
54          replace(x, y, k);
55      }
56      for (int i = 1; i <= n; ++i){
57          for (int j = 1; j <= m; ++j)
58              cout << g[i][j] <<  " ";
59          cout << endl;
60      }
61      return 0;
62  }
```