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

November 11, 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 11 月 11 日

目录

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 魔王语言解释

2.1 需求分析

2.1.1 题目描述

有一个魔王总是使用自己的一种非常精炼而抽象的语言讲话，没有人能听得懂，但他的语言是可以逐步解释成人能听懂得语言，因为他的语言是由以下两种形式的规则由人的语言逐步抽象上去的：

(1) $\alpha \to \beta_1\beta_2 \ldots \beta_m$

(2) $(\theta\delta_1\delta_2 \ldots \delta_n) \to \theta\delta_n\theta\delta_{n-1} \ldots \theta\delta_1\theta$

在这两种形式中，从左到右均表示解释。试写一个魔王语言的解释系统，把他的话解释成人能听懂的话。

用下述两条具体规则和上述规则形式 (2) 实现。设大写字母表示魔王语言的词汇；小写字母表示人的语言词汇；希腊字母表示可以用大写字母或小写字母代换的变量，魔王语言可能含人的词汇。

(1) $B \to tAdA$

(2) $A \to sae$

2.1.2 题目分析

题目要求对魔王语言进行解释，我们发现魔王语言有两种解释规则。我们可以逐一实现每种解释。

- 规则一

  规则一实际上就是一个简单的替换过程。如果在进行一次替换之后语句中仍含有魔王的语言，那么仍需要再次进行替换，直至语句中全部只有人类语言（即小写字母）。这一过程我们可以借助队列实现。

  对于题目中的情况较为简单，因为替换规则是一定的。而如果替换规则是需要用户输入的，那么我们可以使用映射表（*map*）来储存这种映射关系。

  容易想到，如果魔王语言的解释规则中存在环，即我们根据替换规则不断替换魔王语言最终又能得到最初的语言时（即形如 $A \to \ldots B \ldots, B \to \ldots C \ldots, C \to \ldots A \ldots$ 的情况），此时魔王语言的解释不存在。

- 规则二

规则而实际上是将一段字符串反转的过程（当然其中新增了一些字符），我们可以借助栈实现。即，遍历字符串，当遍历到左括号时，将以后遍历到的除第一个字符外的全部字符全部放入一个栈中（第一个字符需要额外储存），直至遍历到右括号。

当遍历到右括号时，将栈中所有字符弹栈并压入答案队列。在每个字符压入队列之前，将先前储存的第一个字符压入队列，以完成规则二的解释效果。

综上所述，我们需要两个数据结构来完成程序。一个栈，一个队列。栈用于反转序列，而答案用于储存答案序列。在进行程序设计时，我们需要考虑规则一和规则二得优先级问题，因为在特殊情况下不同的优先级将会产生不同的结果（比如：$(aBaB)$）。

在题目中这是一个未定义行为，所以我们定义规则二的优先级大于规则一，即当所有得规则二解释完成后，再统一进行规则的解释。

2.2 程序设计

程序设计秉持着先前的思路，先后进行规则二的解释和规则一的解释。

对于规则二的解释 我们利用栈来完成对序列的反转，代码如下：

```
int len = strlen(s);
char c = 0;
for (int i = 0; i < len; ++i){
  // 翻转括号内的序列
  if (s[i] == '(') {
    q.push(s[++i]);
    // 储存括号后第一个字符，用于反转时插入
    c = s[i];
    continue;
  }
  if (s[i] == ')') {
    // 依次弹栈，达到反转字符串的效果
    while (!st.empty()) {
      // 按照规则，应当插入括号后第一个字符
      q.push(st.top());
      q.push(c);
      st.pop();
    }
    c = 0;
    continue;
  }
  // 如果存在括号后第一个字符，那么这个字符一定是括号内的字符
  if (c) st.push(s[i]);
  else q.push(s[i]);
}
```

对于规则一的解释 我们最后统一解决替换的问题，这一部分可以合并到输出。即我们在实现规则二的时候会将所有字符按照顺序存入一个队列中，那么我们的输出其实就是将队列中所有

元素输出。输出时注意下替换规则就好了。

```
while (!q.empty()){
  c = q.front(); q.pop();
  if (c == 'A') cout << "sae";
  else if (c == 'B') cout << "tsaedsae";
  else cout << c;
}
```

2.3 程序分析

时间复杂度 毫无疑问是 $O(n)$ 的，因为每个字符最多被操作两次。

空间复杂度 也是 $O(n)$。

3 迷宫

3.1 需求分析

3.1.1 题目描述

以一个 $m \times n$ 的长方阵表示迷宫，0 和 1 分别表示迷宫中的通路和障碍。设计一个程序，对任意设定的迷宫，求出一条从入口到出口的通路，或得出没有通路的结论。

3.1.2 题目分析

经典的搜索题，采用 *BFS* 或者 *DFS* 均能通过此题。这里我们采用 *DFS*。在递归中采用栈记录路径，当最终走到出口时，从**栈底**开始输出路线即可。

在程序之前我们先行解决下列问题：

1). 地图的储存利用二维矩阵模拟地图，为了遍历方便，我们将地图周围设置为 1（即有障碍物的情况）。这样我们在搜索时只用关心有无搜索过和是否时边界，而不需要担心是否会走出地图。

2). 答案储存手写栈储存答案，当我们遍历一个结点的下一个结点时，栈中压入路径；当回溯时将当前栈顶元素弹出。这样可以保证在输出是输出的为从起点到终点的路径。

3). 栈中元素在写栈时，我们需要的元素类型要有三个域，x, y, d 分别表示当前坐标和来到此坐标的方向。

3.2 程序设计

3.2.1 整体概览

```
#include <iostream>
using namespace std;
const int MAXN = 1e3 + 5;
```

```
// 路径储存元素
struct Poi{}st[MAXN * MAXN]; // 静态开栈

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

// 输出答案
inline void print();

// dfs, 搜索, 程序主干
// 当存在路径时返回true, 否则false
bool dfs(int x, int y);

// 程序入口
int main()
```

3.2.2 具体分析

路径元素 已经说过，元素有三个域。

```
struct Poi{
  int x, y, d;
  Poi(int x = 0, int y = 0, int d = 0):x(x),y(y),d(d){}
}st[MAXN * MAXN];
```

搜索 *dfs* 函数含有两个参数，表示当前搜索到的点。注意 *dfs* 不需要“反悔”功能，即在遍历完结点后不需要将其 *vis* 置 *false*。这两者的差别在哪?

置 *false* 代表我要找到到这个点的所有的排列组合，即路径。而不置 *false* 代表我只关心有没有路径，或者我只需要一条路径。

```
bool dfs(int x, int y) {
  // dfs边界
  if (x == dsx && y == dsy){
    print();
    return true;
  }
  vis[x][y] = true;
  // 遍历四个方向
  for (int i = 1; i <= 4; ++i){
    int tx = x + dx[i], ty = y + dy[i];
    if (!vis[tx][ty]) {
      st[top++] = {x, y, i};
      if (dfs(tx, ty)) return true;
      --top;
    }
  }
```

```
17    // 不需要回溯，否则会TLE
18  //     vis[x][y] = false;
19    return false;
20  }
```

3.3 程序复杂度

时间复杂度 因为每个点最多可能被遍历一次，所以时间复杂度是 $O(nm)$，即棋盘大小。

空间复杂度 我们只需要储存真个棋盘就可以了，所以空间复杂度是 $O(mn)$，即棋盘大小。

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 魔王语言的解释源码

```cpp
#include <iostream>
#include <cstring>
#include <stack>
#include <queue>
using namespace std;
const int MAXN = 1e5 + 5;
char s[MAXN];
int main(){
  cin >> s;
  queue<char> q;
  stack<char> st;
  int len = strlen(s);
  char c = 0;
  for (int i = 0; i < len; ++i){
    // 翻转括号内的序列
    if (s[i] == '(') {
      q.push(s[++i]);
      // 储存括号后第一个字符，用于反转时插入
      c = s[i];
      continue;
    }
    if (s[i] == ')') {
      // 依次弹栈，达到反转字符串的效果
      while (!st.empty()) {
        // 按照规则，应当插入括号后第一个字符
        q.push(st.top());
        q.push(c); st.pop();
      }
      c = 0; continue;
    }
    // 如果存在括号后第一个字符，那么这个字符一定是括号内的字符
    if (c) st.push(s[i]);
    else q.push(s[i]);
  }
  while (!q.empty()){
    c = q.front(); q.pop();
    if (c == 'A') cout << "sae";
    else if (c == 'B') cout << "tsaedsae";
    else cout << c;
  }
  return 0;
}
```

B 迷宫源码

```
#include <iostream>
using namespace std;
const int MAXN = 1e3 + 5;
struct Poi{
  int x, y, d;
  Poi(int x = 0, int y = 0, int d = 0):x(x),y(y),d(d){}
}st[MAXN * MAXN];
int dx[] = {0, 0, 1, 0, -1};
int dy[] = {0, 1, 0, -1, 0};
int top;
int n, m, sx, sy, dsx, dsy;
bool vis[MAXN][MAXN];

inline void print(){
  for (int i = 0; i < top; ++i){
    printf("(%d,%d,%d),", st[i].x, st[i].y, st[i].d);
  }
  printf("(%d,%d,%d)", dsx, dsy, 1);
}
bool dfs(int x, int y) {
  if (x == dsx && y == dsy){ print(); return true; }
  vis[x][y] = true;
  for (int i = 1; i <= 4; ++i){
    int tx = x + dx[i], ty = y + dy[i];
    if (!vis[tx][ty]) {
      st[top++] = {x, y, i};
      if (dfs(tx, ty)) return true;
      --top;
    }
  }
  // 不需要回溯，否则会TLE
//    vis[x][y] = false;
  return false;
}
int main(){
  cin >> n >> m >> sx >> sy >> dsx >> dsy;
  for (int i = 1; i <= n; ++i)
    for (int j = 1; j <= m; ++j)
      cin >> vis[i][j];
  for (int i = 0; i <= n + 1; ++i) vis[i][0] = vis[i][m + 1] = true;
  for (int i = 0; i <= m + 1; ++i) vis[0][i] = vis[n + 1][i] = true;
  if (!dfs(sx, sy)) cout << "no";
  return 0;
}
```