

数据结构-课程实习

数据结构-课程实习

- 一、说明
- 二、文件结构
- 三、测试
 - 1. 线性链表
 - 3. 循环队列
 - 3. 哈夫曼树
 - 4. 图

一、说明

本项目为遥感学院《数据结构》课程实习。本项目使用的是C++语言，使用了面向对象的模式进行代码的编写。

本次实习一共包含四部分内容：

- 线性链表及其相关操作的实现 [linklist.cpp](#)
- 循环队列及其相关操作的实现 [cirqueue](#)
- 哈夫曼树的生成 [hfmtree](#)
- 图的深度优先遍历 [graph](#)

二、文件结构

```
1  .
2  |— build 编译文件夹
3  |— img 图片文件夹
4  |— readme.md
5  |— src 源码文件夹
6      |— cirqueue 循环队列源码
7      |— graph 图源码
8      |— hfmtree 哈夫曼树源码
9      |— linklist 线性链表源码
```

三、测试

1. 线性链表

测试代码

```
1  int main(int argc, char const *argv[]) {
2      try {
3          int array[] = {1, 2, 3, 4};
4          // 构造一个含有四个元素的链表
5          ns_db::LinkedList ls(array, 4);
6
7          // 打印链表
8          ls.traverse(print);
9          std::cout << std::endl;
10
11         int elem;
12         // 将链表下标为"3"的元素删除(下标从"0"开始), 并打印链表
13         ls.erase(elem, 3).traverse(print);
14         std::cout << std::endl;
15
16         // 先后在链表的两个位置插入两个元素, 并打印链表
17         ls.insert(12, 2).insert(34, 0).traverse(print);
18         std::cout << std::endl;
19
20     } catch (const std::exception &e) {
21         std::cerr << e.what() << '\n';
22     }
23     return 0;
24 }
```

程序输出

```
1  // 输出结果, 其中'node-destructor'表示节点被删除, 内存被释放
2  1 2 3 4
3  'node-destructor'
4  1 2 3
5  34 1 2 12 3
6  'node-destructor'
7  'node-destructor'
8  'node-destructor'
9  'node-destructor'
10 'node-destructor'
11 'node-destructor'
```

3. 循环队列

测试代码

```
1  int main(int argc, char const *argv[]) {
2      try {
3          // 构造最大长度为4的队列
4          ns_db::CirQueue cq(4);
5          // 4个元素依次入队, 并遍历输出队列
6          cq.push(12).push(23).push(34).push(13).traverse(print);
7          std::cout << std::endl;
8          // 输出该队列的状态信息
```

```

9      std::cout << cq << std::endl
10          << std::endl;
11
12      int top;
13      // 元素出队
14      cq.pop(top).traverse(print);
15      std::cout << "the top elem is: " << top << std::endl;
16      // 输出该队列的状态信息
17      std::cout << cq << std::endl
18          << std::endl;
19
20      cq.push(43);
21      cq.traverse(print);
22      // 输出该队列的状态信息
23      std::cout << cq << std::endl;
24      std::cout << std::endl
25          << std::endl;
26
27      // 元素出队
28      cq.pop(top);
29      std::cout << "the top elem is: " << top << std::endl;
30      cq.traverse(print);
31      // 输出该队列的状态信息
32      std::cout << cq << std::endl;
33      std::cout << std::endl
34          << std::endl;
35
36      } catch (const std::exception &e) {
37          std::cerr << e.what() << '\n';
38      }
39      return 0;
40  }

```

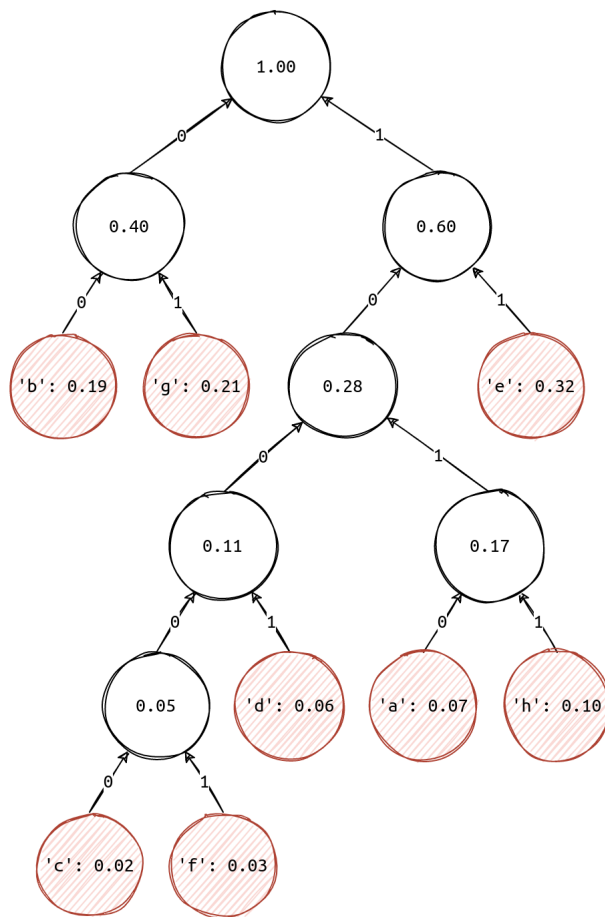
程序输出

```

1  12 23 34 13
2  {'maxSize': 5, 'enableSize': 4, 'front': 0, 'rear': 4}
3
4  23 34 13 the top elem is: 12
5  {'maxSize': 5, 'enableSize': 4, 'front': 1, 'rear': 4}
6
7  23 34 13 43 {'maxSize': 5, 'enableSize': 4, 'front': 1, 'rear': 0}
8
9
10 the top elem is: 23
11 34 13 43 {'maxSize': 5, 'enableSize': 4, 'front': 2, 'rear': 0}

```

3. 哈夫曼树



测试代码

```

1  int main(int argc, char const *argv[]) {
2      try {
3          /**
4           * {a, b, c, d, e, f, g, h}
5           * {0.07, 0.19, 0.02, 0.06, 0.32, 0.03, 0.21, 0.10}
6           */
7          ns_db::Node nodes[] = {
8              ns_db::Node('a', 0.07f), ns_db::Node('b', 0.19f), ns_db::Node('c', 0.02f), ns_db::Node('d',
0.06f),
9              ns_db::Node('e', 0.32f), ns_db::Node('f', 0.03f), ns_db::Node('g', 0.21f), ns_db::Node('h',
0.10f)};
10         // 构建哈夫曼数
11         ns_db::HFMTTree hfmt(nodes, 8);
12         // 输出哈夫曼树的内部构造数据
13         std::cout << hfmt << std::endl;
14         // 打印最后的哈夫曼编码
15         hfmt.printHFMCCode();
16
17     } catch (const std::exception &e) {
18         std::cerr << e.what() << '\n';
19     }
20
21     return 0;
22 }

```

程序输出

```

1  0: {'pa': 10, 'lc': -1, 'rc': -1, 'w': 0.07}
2  1: {'pa': 12, 'lc': -1, 'rc': -1, 'w': 0.19}

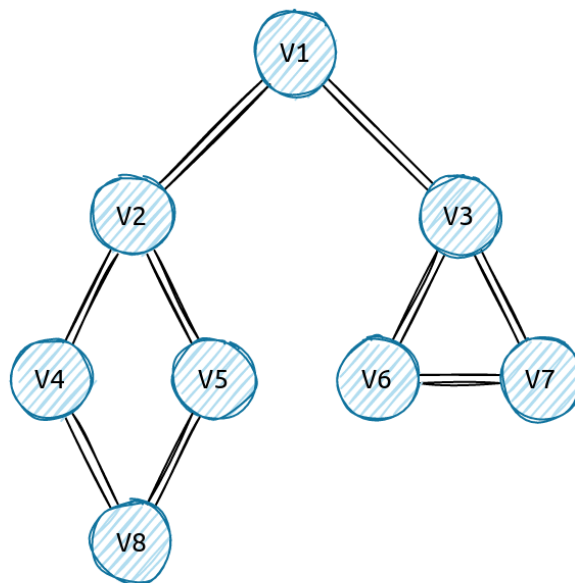
```

```

3  2: {'pa': 8, 'lc': -1, 'rc': -1, 'w': 0.02}
4  3: {'pa': 9, 'lc': -1, 'rc': -1, 'w': 0.06}
5  4: {'pa': 13, 'lc': -1, 'rc': -1, 'w': 0.32}
6  5: {'pa': 8, 'lc': -1, 'rc': -1, 'w': 0.03}
7  6: {'pa': 12, 'lc': -1, 'rc': -1, 'w': 0.21}
8  7: {'pa': 10, 'lc': -1, 'rc': -1, 'w': 0.1}
9  8: {'pa': 9, 'lc': 2, 'rc': 5, 'w': 0.05}
10 9: {'pa': 11, 'lc': 8, 'rc': 3, 'w': 0.11}
11 10: {'pa': 11, 'lc': 0, 'rc': 7, 'w': 0.17}
12 11: {'pa': 13, 'lc': 9, 'rc': 10, 'w': 0.28}
13 12: {'pa': 14, 'lc': 1, 'rc': 6, 'w': 0.4}
14 13: {'pa': 14, 'lc': 11, 'rc': 4, 'w': 0.6}
15 14: {'pa': -1, 'lc': 12, 'rc': 13, 'w': 1}
16
17 'a': 1010
18 'b': 00
19 'c': 10000
20 'd': 1001
21 'e': 11
22 'f': 10001
23 'g': 01
24 'h': 1011

```

4. 图



测试代码

```

1  int main(int argc, char const *argv[]) {
2      ns_db::UDGraphInfo info[] = {
3          ns_db::UDGraphInfo('1', '2'),
4          ns_db::UDGraphInfo('1', '3'),
5          ns_db::UDGraphInfo('2', '4'),
6          ns_db::UDGraphInfo('2', '5'),
7          ns_db::UDGraphInfo('4', '8'),
8          ns_db::UDGraphInfo('5', '8'),
9          ns_db::UDGraphInfo('3', '6'),
10         ns_db::UDGraphInfo('3', '7'),

```

```
11     ns_db::UDGraphInfo('6', '7'));
12
13     ns_db::UDGraph graph(info, 9);
14     std::cout << graph << std::endl;
15
16     graph.traverseDFS(print);
17     std::cout << std::endl;
18
19     return 0;
20 }
```

程序输出

```
1  {'pos': 0, 'symbol': 1, 'links': [(1)-(2)-]}
2  {'pos': 1, 'symbol': 2, 'links': [(0)-(3)-(4)-]}
3  {'pos': 2, 'symbol': 3, 'links': [(0)-(6)-(7)-]}
4  {'pos': 3, 'symbol': 4, 'links': [(1)-(5)-]}
5  {'pos': 4, 'symbol': 5, 'links': [(1)-(5)-]}
6  {'pos': 5, 'symbol': 8, 'links': [(3)-(4)-]}
7  {'pos': 6, 'symbol': 6, 'links': [(2)-(7)-]}
8  {'pos': 7, 'symbol': 7, 'links': [(2)-(6)-]}
9
10 1 2 4 8 5 3 6 7
```