

数据结构：求有向无环图最长路径 实验报告

毛子恒 李臻 张梓靖

2020 年 12 月 4 日

小组成员

班级：2019211309

姓名：毛子恒

学号：2019211397

分工：代码 文档

班级：2019211310

姓名：李臻

学号：2019211458

分工：测试 文档

班级：2019211308

姓名：张梓靖

学号：2019211379

分工：可视化 文档

目录

1 需求分析	2
2 概要设计	3
3 详细设计	4
4 调试分析报告	6
5 用户使用说明	7
6 测试结果	7
7 可视化	8

1 需求分析

1.1 题目描述

给定一个有向无环图，找到图中距离最远的两个结点。

1.2 输入描述

程序从标准输入中读入数据。

输入的第一行包含两个整数 n, m ，用空格分隔，分别表示有向无环图的点数和边数。

接下来的 m 行，每行三个整数，用空格分隔，分别表示每个弧的弧尾、弧头和权值，顶点编号 x 的范围满足 $1 \leq x \leq n$ ，权值为正数。

1.3 输出描述

程序向标准输出中输出结果。

输出分为以下五种情况：

1. 输入合法，程序正常运行结束，此时输出一行三个整数，用空格分隔，分别表示最长的距离和最长的两个顶点的编号。
2. 输入的有向无环图没有边，此时程序输出一个字符串 “No solution.” 表示无解。
3. 输入的有向无环图结点数不足 2 个，此时程序输出一个字符串 “there should be at least two points!”。
4. 输入的弧头、弧尾或者权值信息范围有误，此时程序输出一个字符串 “Invalid input.”。
5. 程序发生运行时错误，比如内存分配失败等，此时程序没有输出。

1.4 样例输入输出

1.4.1 样例输入输出 1

【输入】

```
aaababcd
```

【输出】

1.4.2 样例输入输出 2

【输入】 (samples/sample2.in)

【输出】 (samples/sample2.out)

1.4.3 样例输入输出 3

【输入】(samples/sample3.in)

【输出】(samples/sample3.out)

1.4.4 样例输入输出 4

【输入】

【输出】

1.4.5 样例输入输出 5

【输入】

【输出】

1.5 程序功能

程序找到图中距离最大的两个点并且输出这两个点及其距离。

2 概要设计

2.1 问题解决的思路

程序建立采用邻接表存储的图，求出从源点（虚拟的）开始到每个顶点的距离及其路径，找到距离源点距离最大的点并且输出。

2.2 有向无环图的设计

```
1 // 数据对象
2 typedef struct node
3 {
4     char data;
5     int cnt;
6     short parent, lc, rc; // 哈夫曼树双亲、孩子结点的下标
7 } Node;
8
9 /*
10 * 操作：编码
11 * 前件：inputFile和outputFile分别指向输入、输出文件
12 * 后件：将编码结果输出到outputFile指向的文件中
13 */
14 void encode(FILE * inputFile, FILE * outputFile);
```

```
15
16  /*
17  * 操作：解码
18  * 前件：inputFile和outputFile分别指向输入、输出文件
19  * 后件：将解码结果输出到outputFile指向的文件中
20  */
21 void decode(FILE * inputFile, FILE * outputFile);
```

2.3 主程序的流程

1. 获取输入，判断是否合法
2. 如果合法，则调用压缩/解压函数

2.4 各程序模块之间的层次关系

程序模块层次关系图如图 1。

图 1: 程序模块层次关系

3 详细设计

3.1 哈夫曼编码的实现

哈夫曼编码的设计中基本操作的伪代码算法如下：

```
1 void encode(FILE * inputFile, FILE * outputFile)
2 {
3     while (true)
4     {
5         从inputFile读入长度最长为1024字符串
6         如果读到文件结束，退出循环
7         cnt数组用来记录各个字符出现的次数，将其初始化
8         统计各个字符出现次数（权值）
9         按照权值从大到小对cnt数组排序
10        从前往后，找到字符串的字符集的大小，设为tot
11        创建tot个哈夫曼树叶结点（哈夫曼树采用顺序存储，权值小的字符在前）
12        创建tot-1个非叶节点：
13        t1 <- 1 目前没有双亲节点的权值最小的叶节点的下标
14        t2 <- tot + 1 目前没有双亲节点的权值最小的非叶节点的下标
15        for (int i = tot + 1 to 2 * tot + 1)
16        {
17            在t1、t1+1、t2、t2+2四个结点中选取权值最小的两个结点
18            创建新的结点，左孩子和右孩子为刚刚选取的结点
19            更新两个结点的双亲以及相应的指针
20        }
21        将tot写入文件
22        将哈夫曼树叶节点存储的字符依次写入outputFile
```

```
23  将哈夫曼树非叶节点的左右孩子节点的下标依次写入outputFile
24  将原字符串的长度写入outputFile
25  for (int i = 1 to tot)
26  {
27      从叶结点开始，逆向求每个字符的哈夫曼编码，作为'0'/'1'字符串写入tempString
28      在huffmanCode数组（编码表）中分配位置，拷贝编码
29  }
30  将原字符串按照编码表编码，将编码结果写入tempString字符串中
31  将tempString字符串的长度补全到8的倍数
32  将tempString字符串每8个字符压缩到一个字节，写入到outputString中
33  将编码后的字符串长度写入outputFile
34  将outputString写入outputFile
35  释放huffmanCode的空间
36  }
37  }
38  void decode(FILE * inputFile, FILE * outputFile)
39  {
40      while (true)
41      {
42          从inputFile读入tot
43          如果读到文件结束，退出循环
44          读入tot个叶节点的信息，复原哈夫曼树
45          读入tot-1个非叶节点的信息，复原哈夫曼树
46          读入原字符串的长度和编码后字符串的长度
47          读入编码后的字符串inputString
48          cur <- 2 * tot - 1 指针，表示当前在哈夫曼树中的结点下标
49          for (int i = 0 to n-1)
50          {
51              for (int j = 0 to 7)
52              {
53                  获取当前位的值temp
54                  if (temp)
55                      cur指向当前结点的右孩子
56                  else
57                      cur指向当前结点的左孩子
58                  if (找到了叶节点)
59                  {
60                      将解码出的字符写入到outputFile
61                      if (解码完了所有字符)
62                          break;
63                      cur恢复指向根结点
64                  }
65              }
66          }
67      }
68  }
```

3.2 函数的调用关系图

函数调用关系图如图 2。

图 2: 函数调用关系图

4 调试分析报告

4.1 调试过程中遇到的问题和思考

由于对二进制文件的输入、输出不熟悉，曾尝试使用 `printf` 和 `scanf` 函数进行输入输出，了解相关知识后改为调用 `fread` 和 `fwrite` 函数。

在解码的过程中 `feof` 函数经常出现探测文件末尾不准确，故改为判断 `fread` 函数是否为 0 来结束循环。

在构造哈夫曼树的过程中由于判断条件过于复杂，出现错误；改为了更简洁的判断条件。

在写入到压缩文件之前，压缩码需要以 '0'/'1' 字符串的形式存储，否则调用各种字符串函数会出现错误。

编码以 8 位压缩到一个字节当中，写入和读取的时候注意高低位的顺序。

4.2 设计实现的回顾讨论

编码是每次从二进制文件中读入 1024 个字节，创建编码表并且压缩。

由于哈夫曼树结点数较少，所以存储下标时采用 `short` 类型，写入压缩文件时可以节省空间。

对于没有出现过的字符显然不需要编码。调用库函数 `qsort` 对权值进行排序。

哈夫曼树采取顺序存储结构，前 n 个结点为叶节点，之后 $n - 1$ 个结点为非叶节点，每次构造新的非叶节点时只需要比较叶节点中权值最小的两个结点和非叶节点中权值最小的两个结点，从中挑出权值最小的两个结点即可，使得查找结点的时间复杂度降低到 $O(1)$

压缩后的串为一个 01 串，将其每 8 位压缩到一个字节并写入压缩文件。

对于只有一种字符的情况，由于无法建立哈夫曼树，所以压缩文件中只存储 `tot`、该字符、原字符串的长度。编码和解码是均对这种情况特殊处理。

4.3 算法复杂度分析

`encode, decode` 函数的复杂度为 $O(len)$ ，其中 len 为字符串长度。

主函数的时间复杂度为 $O(1)$ ，整体时间复杂度为 $O(len)$ 。

4.4 改进设想的经验和体会

4.4.1 改进 1

可以通过进一步压缩位（比如去掉一定为 0 的某些字节）来使得压缩率更高，但是编程复杂度会显著提高。

4.4.2 改进 2

可以判断如果压缩率大于 1，则直接写入原文件。

5 用户使用说明

使用 gcc 编译生成可执行文件。

```
gcc -o main -std=c11 main.c huffman.c
```

执行可执行文件：

```
./main
```

在 Windows cmd 下：

```
main
```

之后将数据写入到指定文件中，可以通过标准输入选择选项，也可以通过执行可执行文件时传递参数来选择选项：

1. 如果执行可执行文件时没有传递参数，则可以根据标准输入的提示选择压缩/解压功能，以及指定输入和输出文件的位置
2. 执行可执行文件时传递的参数：第一个参数为 1 或 2，表示选择压缩/解压，之后两个参数分别为输入文件的路径和输出文件的路径，例如，执行

```
./main 1 ./input.in ./output.out
```

表示执行压缩功能，输入文件为本目录下的 input.in，输出文件为本目录下的 output.out

通过标准输出打印提示信息。如果输入合法并且程序正常运行结束，主函数返回值为 0。

6 测试结果

测试环节分为三个步骤。

6.1 测试第一部分

对 1.4 节给出的样例进行测试。

6.2 测试第二部分

测试边界条件。

【输入】 (samples/sample4.in)

(空文件)

【输出】 (samples/sample4.out)

(空文件)

【输入】 (samples/sample5.in)

文件内容为 1000000 个字符 a

【输出】 (samples/sample5.out)

压缩结果为一个 8873 字节的二进制文件。

【输入】

上一个例子的输出。

【输出】 (samples/sample6.out)

解压与 sample5.in 相同。

6.3 测试第三部分

测试在 macOS Big Sur 11.0.1 下进行。

在 $len \leq 10$, $len \leq 100$, $n \leq 1000000$ 的范围下分别随机生成 1000 组测试数据，进行压缩和解压操作，并且比对解压结果和原始输入是否相同。

3000 组数据中没有出现错误。

数据生成程序 (testing/data.cpp) 如下：

```
1 #include <bits/stdc++.h>
2 using namespace std;
3
4 int main(int argc, char *argv[])
5 {
6     srand(time(0));
7     FILE *file = fopen("./raw.in", "wb");
8     int n = atoi(argv[1]), m = atoi(argv[2]);
9     for (int i = 1; i <= n; ++i)
10         fprintf(file, "%c", rand() % m + 32);
11     fclose(file);
12     return 0;
13 }
```

传入两个参数，第一个参数是字符串的长度，第二个参数是字符集的大小。

比对脚本 (testing/chk.sh) 如下：

```
1 for i in {1..100}
2 do
3     sleep 1
4     ./data 100000 5
5     ./main 1 raw.in encode.out
6     ./main 2 encode.out decode.out
7     if ! diff raw.in decode.out
8     then
9         break
10    fi
11    echo "Correct"
12 done
```

7 可视化

随机生成若干组数据，使用该程序进行压缩，记录不同数据规模下的平均压缩率，并且使用 JavaScript 将结果可视化。

比对脚本 (testing/test.py) 如下：

```
1 import os
2 import time
3
4 with open("./result.txt", "w") as file:
5     for i in (10, 30, 50, 100, 500, 1000, 10000, 100000):
```



```
6     for j in (3, 6, 10, 20, 30, 50, 95):
7         tot = 0
8         for k in range(100):
9             time.sleep(1)
10            os.system("./data {0} {1}".format(i, j))
11            os.system("./main 1 raw.in out.out")
12            tot += os.path.getsize("./out.out") / \
13                   os.path.getsize("./raw.in")
14            file.write("[{0}, {1}, {2}],\n".format(i, j, str(tot / 10)[:9]))
```

数据规模如下：

$len = 10, 30, 50, 100, 500, 1000, 10000, 1000000$ ，字符集大小 = 3, 6, 10, 20, 30, 60, 95 的数据各 100 组，共 5600 组数据。

7.0.1 实现细节

利用已有的压缩率数据，使用 Highcharts 库绘制散点图。

7.0.2 用户使用说明

使用现代浏览器打开 Scatter/index.html，即可看到散点图，将鼠标指针放到某个点上可以看到对应的数据值，拖动图表可以从不同角度观察数据分布。

7.0.3 示例

见图 3。

图 3: 可视化示例

7.0.4 结论

可以观察到在字符串长度较小的时候和字符集大小较大的时候压缩率有明显上升，理想情况下在字符集大小在 20 以内，字符串大小在 100 以上会有比较好的压缩效果。