上 海 大 学

2021-2022 冬季学期

《数据结构（1）》实验报告

实 验 组 号：08

上 课 老 师：沈 俊

小 组 成 绩：

小组成员成绩表

序号	学号	姓名	贡献因子	成绩
1	20120500	王静颐	20	
2	20120796	康高熙	20	
3	20121034	胡才郁	20	
4	20121076	刘元	20	
5	20124633	金之谦	20	

注：小组所有成员的贡献因子之和为 100.

计算机工程与科学学院

2021 年 02 月 26 日

实验四 串、数组与广义表

1 设计性实验

1.1 稀疏矩阵的二元组表示

1.1.1 基本功能介绍

在稀疏矩阵的二元组表示中，用一个二元组存放矩阵的非零元素，其中每个二元组只记录非零元素的列号和元素值，且各二元组按行号增加的顺序排列。另外，设一个行指针数组，其第 i 个元素表示稀疏矩阵中第 i 行的第一个非零元素在二元组中的存放位置。

1.1.2 主要算法设计

分析题意之后了解到，初始我们得到一个 n*m 的矩阵，矩阵中有若干个位置存放的值是 0，而 0 对我们来说没用，也就是说有用的值的个数远小于 n*m，即稀疏矩阵。为了更有效的存放，我们按照二元组存放。我的思路是，类似图论建边那样，写一个链式向前星。在存边的时候，我们令 ver[x]存储编号为 x 的这条边到达的节点，lin[x]存储从 x 节点出发的若干条边中的第一条边的编号，Next[i]表示编号为 i 的这条边的下一条边的编号。类似的，在本题当中，行指针数组其实就是我们的 lin 数组，而对于每一行的非零元素遍历的时候，我们是通过 for 循环遍历 Next 数组。

1.1.3 主要数据组织

自定义 Node 类进行处理，其数据成员以及主要成员函数如下图所示：

```
node类有数据成员:
int ver;          // 当前节点所属第几列
int edge;         // 节点内存储的数值
node *nxt;        // 当前节点所属行的下一个有效元素
成员函数:
    node(int y,int z,node *Next=NULL){
    ver=y;edge=z;nxt=Next;
}                          // 有参数的构造函数
int getver();              // 访问node类的ver并将值返回
int getedge();             // 访问node类的edge并将值返回
node * getnxt();           // 访问node类的nxt指针并返回
void setnxt(node *a);      // 将a赋值给nxt指针
node *lin[1100];           // 行指针数组，lin[i]存储的是第i行第一个非零元素节点的地址
```

1.1.4 测试分析

题目中案例如下图：

$$\begin{bmatrix} 12 & 0 & 11 & 0 & 0 & 13 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 14 \\ 0 & -4 & 0 & 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 8 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -9 & 0 & 0 & 2 & 0 & 0 \end{bmatrix}$$

编写如下的测试函数，并进行程序测试：

```
FILE* fp = fopen("matrix.txt","r");
fscanf(fp,"%d%d",&n,&m);
int x;
for(int i=1;i<=n;i++){
    bool flag=0;
    node *tmp;
    for(int j=1;j<=m;j++){
        fscanf(fp, "%d",&x);
        if(x!=0){
            if(!flag){
                lin[i]=new node(j,x,NULL);
                flag=1;tmp=lin[i];
            }else{
                node *p=new node(j,x,NULL);
                tmp->setnxt(p);
                tmp=p;
            }
        }
    }
}
for(int i=1;i<=n;i++){
    node *tmp=lin[i];
    printf("查看第%d行非零元素: \n",i);
    if(tmp==NULL){
        printf("不存在任何非零元素! \n");
    }
    while(tmp!=NULL){
        cout<<'('<<i<<','<<(tmp->getver())<<')'
            <<" = "<<(tmp->getedge())<<endl;
        tmp=tmp->getnxt();
    }
}
```

测试结果如下：

```
6 7
12 0 11 0 0 13 0
0 0 0 0 0 0 14
0 -4 0 0 0 3 0
0 0 0 8 0 0 0
0 0 0 0 0 0 0
0 -9 0 0 2 0 0
查看第1行非零元素:
(1,1) = 12
(1,3) = 11
(1,6) = 13
查看第2行非零元素:
(2,7) = 14
查看第3行非零元素:
(3,2) = -4
(3,6) = 3
查看第4行非零元素:
(4,4) = 8
查看第5行非零元素:
不存在任何非零元素!
查看第6行非零元素:
(6,2) = -9
(6,5) = 2
```

经检测，输出符合预期结果。

2 综合性实验

2.1文学研究助手

2.1.1实验内容

文学研究人员需要统计英文小说中某些词出现的次数和位置。试编写一个实现这一目 标的文字统计系统，称为“文学研究助手”。 英文小说存于一个文本文件中，并假设小说中的单词一律不跨行，每行的长

度不超过 120 个字符，待统计的词汇集合要一次输入完毕。要求对英文小说扫描一遍就完成统计工作。程序的输出结果是每个单词的出现次数和出现位置所在行的行号，其格式自行设计。

2.1.2 算法分析

实验内容要求扫描文章一遍统计出所有词汇的出现次数和第一次出现的行号。由于考虑实际中单词与单词之间有空格隔开以区分每个单词，我们只需要找出所有的单个单词，与词库进行匹配，而不需要使用 KMP 算法。

我们小组利用了字典树 Trie 来实现高效地把每个单词与词库进行匹配。简单来讲 Trie 就是将所有词库内的字符存于一棵树中，每条树边表示一个字符，来实现有序地存储多个字符串。对于每个文章中的串，使用 Trie 进行匹配的时间复杂度为 O(字符串长度)，相对于 O(n*字符串长度)来说有了很大的优化。

此外，实验要求使用文件读入文章，我们同样将词库存于文本文件中，利于读入。

2.1.3 主要数据组织

字典树类 Trie 有设计如下：

```
1  class Trie {                    //字典树类
2  public:
3      Trie() : cnt(0);             //构造函数
4      void insert(char *s, int bh);    //将词库中的单词插入Trie
5      void match(char *s,int row);  //将文章中的单词与词库匹配
6      void askcount(int bh);        //询问词库中每个词的出现次数和出现位置行号
7  private:
8      int cnt, trans[N][26], code[N], count[N];    //记录Trie的基本信息
9      vector<int>vis[N];          //记录词库中每个词的出现位置行号
10 } T;                           //创建字典树类的对象
11
```

表 1. Trie 类中数据成员与函数的声明

数据成员/函数原型	返回值类型	功能
Trie() : cnt(0);	无	构造函数，初始节点数为 0
int cnt;	无	Trie 内节点总数
int trans[N][26];	无	Trans[i][j]记录 i 号节点后接字符 j（小写英文字符-‘a’）的节点编号，若无则为 0
int code[N];	无	记录当前节点是否为某一单词的结尾，若不是则为 0，若是则为该串编号
int count[N] ;	无	记录每个单词的出现次数
vector<int>vis[N];	无	记录每个单词的出现位置行号
void insert(char *s, int bh);	void	将词库中的单词插入 Trie
void match(char *s,int row);	void	将文章进行匹配并修改 count 和 vis
void askcount(int bh);	void	std 中输出词库中每个单词的匹配结果

2.1.4 数据测试

对于文件操作，我们将词库单词存储于根目录下 cmd.txt ，将文章存储与根目录下 test.txt，输出则是使用 std 输出。具体代码实现在测试中体现。

测试函数完成了对文件的处理，实现如下：

```int main() {
    int n;
    FILE* fp1 = fopen("cmd.txt","r");
    fscanf(fp1, "%d", &n);
    char a[101][310];

    for (int i = 1; i <= n; i++) {
        fscanf(fp1, "%s", a[i]); //读入字典的单词
        T.insert(a[i], i);
    }

    FILE* fp2 = fopen("test.txt","r");

    char s[310];
    int row=0;
    while (fgets(s, 100, fp2)) {
        row++;
        T.match(s,row); //读入文章
    }
    for (int i = 1; i <= n; i++) {
        printf("字典中第 %d 个串:  %s \n", i, a[i]);
        T.askcount(i);
    }
    system("pause");
    return 0;
}
```

词库文件 cmd.txt 内容如下：

```5
Traveller
His
to
So
You
```

文章文件 test.txt 的内容如下：

```The Time Machine, by H. G. Wells [1898]



I


The Time Traveller (for so it will be convenient to speak of him)
was expounding a recondite matter to us. His grey eyes shone and
twinkled, and his usually pale face was flushed and animated. The
fire burned brightly, and the soft radiance of the incandescent
lights in the lilies of silver caught the bubbles that flashed and
passed in our glasses. Our chairs, being his patents, embraced and
caressed us rather than submitted to be sat upon, and there was that
luxurious after-dinner atmosphere when thought roams gracefully
free of the trammels of precision. And he put it to us in this
way--marking the points with a lean forefinger--as we sat and lazily
admired his earnestness over this new paradox (as we thought it)
and his fecundity.
```

运行代码后得到的结果如下（不区分大小写）：

经过基本比对，结果正确，在不区分大小写的情况下完成了单词的全字匹配。

3.1 课程设计中遇到的问题和解决方法

在设计文学研究助手时，经过了小组讨论与搜集资料，我们对算法的时间复杂度进行了优化，特殊位置、临界位置进行了特殊处理，并且对换行的情况进行了特殊处理，保证了程序的健壮性。

3.2 实验总结

验证性实验是有助于夯实基础，对于综合性实验，我们不仅仅应当注意是否能够解决问题，更要注重优化算法的时间复杂度。严谨性也同样重要，代码实现以后，一定要通过多组数据检验，确保其正确性。