**《数据结构》实验报告（第二章）**

**学号：\_\_58122231\_\_\_\_ 姓名 ：\_\_\_\_陆文韬\_\_\_\_\_\_\_**

**实验题号：P93, 2; P107, 4;P118,1 实验日期：\_2023.11.07\_**

**实验一**

**1．问题描述：**描述实验内容和要求以及需要解决的问题。

假设a，b为两个有序表，如果对于0<=i<j，且aj<bj，有ai=bi；或者对于0<=i<n，且n<m，有ai=bi，则称a<b，试编写一个函数，使得当a<b，a=b，a>b时，分别返回-1，0，1。假定每个ai，及每个bj都是整数。

**2．算法思想：**详细描述解决相应问题所需要的算法设计思想。

首先使用a、b两个数组分别存储要比较的两个线性表，先用min\_length来计算两个数组的最小长度，对这一段数组进行比较，如果出现相同下标的字符不一致，就返回结果，如果min\_length长度内的字符都相等，则计算两个数组的长度，如果长度相同，则返回0，如果长度不相等，则长度更长的数组更大。

**3．功能函数：**描述所设计的功能函数。如果有多个函数，需要描述它们之间的关系。

读取两个数组，根据题目所给出的规则，计算两个数组的长度，返回结果。

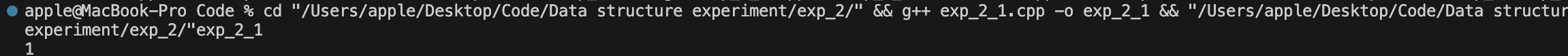
**4．测试数据：**设计测试数据，或具体给出测试数据。

提示：要求测试数据能全面地测试所设计程序的功能。

int a[N] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 10};

int b[M] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 11, 12, 13};

**5．测试情况：**给出程序的测试情况，分析运行结果，显示实验结果截图。

****

**6．实验总结：**写出实验过程中遇到的问题，以及问题的解决过程。分析算法的时间复杂度和空间复杂度，总结实验心得体会。

空间复杂度：4\*（M+N）字节

时间复杂度：O（min（M，N））

**7. 源代码：**给出项目所有源程序清单。

// P93 2

#include <iostream>

using namespace std;

#define N 10

#define M 15

int main()

{

// 存储两个有序表

int a[N] = {0, 1, 2, 3, 4, 5, 6, 7, 8};

int b[M] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 10, 11, 12, 13};

// 计算两个数组的较小长度

int min\_length = sizeof(a) / 4 <= sizeof(b) / 4 ? sizeof(a) : sizeof(b);

cout << min\_length << endl;

for (int i = 0; i < min\_length; i++)

{

if (a[i] == b[i])

continue;

else if (a[i] > b[i])

{

cout << 1 << endl;

return 0;

}

else

{

cout << -1 << endl;

return 0;

}

}

// 两个数组字符相同且长度相等

if (sizeof(a) == sizeof(b))

cout << 0 << endl;

// 长度不等

else

{

sizeof(a) > sizeof(b) ? (cout << 1 << endl) : (cout << -1 << endl);

}

return 0;

}

**实验二**

**1．问题描述：**描述实验内容和要求以及需要解决的问题。

重写快速转置函数，使它利用一个数组存储RowSize和RowStart，而不是两个数组。

**2．算法思想：**详细描述解决相应问题所需要的算法设计思想。

首先用一个三元组类存储稀疏矩阵中的非零元素，将这些元素存储在稀疏矩阵类的smArray中，定义一个行指针数组，先用循环使得行指针数组每个元素存储其下标所对应的行内的非零元素个数，再采用一个从后往前的循环使得行指针数组内的元素存储其下标对应的行内的第一个非零元素在smArray中的位置，根据行指针数组对数组内存储的非零元素进行转置，得出最终的稀疏矩阵。

**3．功能函数：**描述所设计的功能函数。如果有多个函数，需要描述它们之间的关系。

SparseMatrix FastTranspose()：将原来的稀疏矩阵对象转置为新的稀疏矩阵并返回。

**4．测试数据：**设计测试数据，或具体给出测试数据。

SparseMatrix s1(8, 7, 9);

int rows[9] = {0, 3, 0, 1, 1, 2, 4, 5, 6};

int cols[9] = {1, 3, 5, 1, 2, 3, 0, 2, 4};

int vals[9] = {15, 23, 56, 12, 3, 33, 1, 28, 4};

**5．测试情况：**给出程序的测试情况，分析运行结果，显示实验结果截图。

****

**6．实验总结：**写出实验过程中遇到的问题，以及问题的解决过程。分析算法的时间复杂度和空间复杂度，总结实验心得体会。

空间复杂度：4\*3\*terms\*2字节

时间复杂度：O(terms)

**7. 源代码：**给出项目所有源程序清单。

// P107 4

#include <iostream>

using namespace std;

// 声明稀疏矩阵类

class SparseMatrix;

// 定义三元组类

class MatrixTerm

{

friend class SparseMatrix;

public:

int row, col, value;

MatrixTerm(int row = 0, int col = 0, int value = 0)

{

this->row = row;

this->col = col;

this->value = value;

}

};

// 定义稀疏矩阵类

class SparseMatrix

{

public:

SparseMatrix(int r, int c, int t)

{

rows = r;

cols = c;

terms = t;

smArray = new MatrixTerm[t];

}

SparseMatrix FastTranspose();

int rows, cols, terms;

MatrixTerm \*smArray;

};

// 快速转置函数

SparseMatrix SparseMatrix::FastTranspose()

{

SparseMatrix b(cols, rows, terms);

if (terms > 0)

{

int \*rowArray = new int[cols];

// 初始化rowArray的每个元素为零

fill(rowArray, rowArray + cols, 0);

// 计算转置后的矩阵每一行非零元素的个数

for (int i = 0; i < terms; i++)

rowArray[smArray[i].col]++;

rowArray[cols - 1] = terms - rowArray[cols - 1];

// rowArray每个元素表示该行第一个元素在smArray中的下标

for (int i = cols - 2; i >= 0; i--)

rowArray[i] = rowArray[i + 1] - rowArray[i];

for (int i = 0; i < terms; i++)

{

int j = rowArray[smArray[i].col];

b.smArray[j].row = smArray[i].col;

b.smArray[j].col = smArray[i].row;

b.smArray[j].value = smArray[i].value;

rowArray[smArray[i].col]++;

}

delete[] rowArray;

}

return b;

};

int main()

{

SparseMatrix s1(8, 7, 9);

int rows[9] = {0, 3, 0, 1, 1, 2, 4, 5, 6};

int cols[9] = {1, 3, 5, 1, 2, 3, 0, 2, 4};

int vals[9] = {15, 23, 56, 12, 3, 33, 1, 28, 4};

for (int i = 0; i < 9; i++)

{

s1.smArray[i].row = rows[i];

s1.smArray[i].col = cols[i];

s1.smArray[i].value = vals[i];

}

SparseMatrix s2(7, 8, 9);

s2 = s1.FastTranspose();

for (int i = 0; i < 9; i++)

{

cout << s2.smArray[i].row << " ";

cout << s2.smArray[i].col << " ";

cout << s2.smArray[i].value << endl;

}

return 0;

}

**实验三**

**1．问题描述：**描述实验内容和要求以及需要解决的问题。

编写一个函数String：：Frequency，以确定字符串中的不同字符各自出现的频度，并用适当的数据对函数进行测试。

**2．算法思想：**详细描述解决相应问题所需要的算法设计思想。

创建一个包含256个元素的数组，根据ASCII码，每个字符对应数组中的一个位置，存储字符出现的次数。

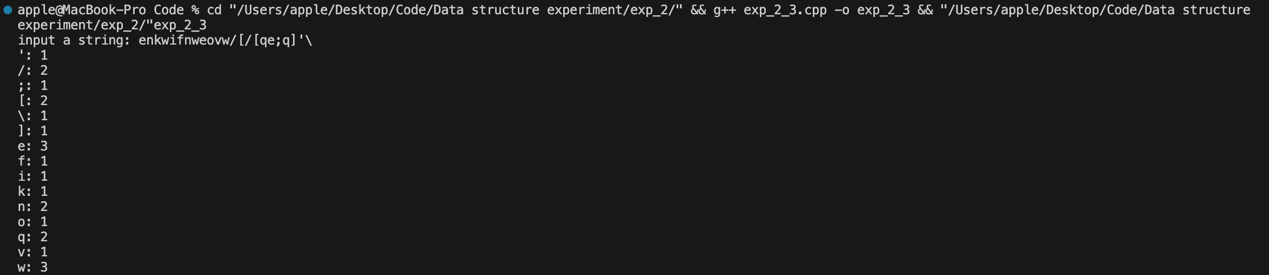
**3．功能函数：**描述所设计的功能函数。如果有多个函数，需要描述它们之间的关系。

void Frequency(string s)：读取一个字符串，输出各个字符出现的频度表

**4．测试数据：**设计测试数据，或具体给出测试数据。

****

**5．测试情况：**给出程序的测试情况，分析运行结果，显示实验结果截图。

****

**6．实验总结：**写出实验过程中遇到的问题，以及问题的解决过程。分析算法的时间复杂度和空间复杂度，总结实验心得体会。

空间复杂度：256\*4字节

时间复杂度：O(256+s.length())

**7. 源代码：**给出项目所有源程序清单。

//P118 1

#include <iostream>

using namespace std;

void Frequency(string s)

{

int a[256] = {0};

int len = s.length();

for (int i = 0; i < len; i++)

{

a[s[i]]++;

}

for (int j = 0; j < 256; j++)

{

if (a[j] != 0)

cout << char(j) << ": " << a[j] << endl;

}

}

int main()

{

cout << "input a string: ";

string s;

cin >> s;

Frequency(s);

return 0;

}

…