# “网络科学基础”-第三次上机报告

班级：物联网 2303　　姓名：邱佳亮　　学号：3230611072

上机日期：2024.11.15，第十一周周五下七八节课

2024 秋-网络科学基础（物联网 23）-第三次上机报告提交

## 一、上机题目

Matlab 代码调用 Cpp 代码。

## 二、上机目的

1. 利用 Cpp 代码抽取 Matlab 矩阵的某些行、某些列上的元素，返回提取到的矩阵数据到 Matlab。
2. C++代码读取 Matlab 中的 matrix 类型数据。
3. C++代码读取 struct 类型数据。
4. 模仿 3.1 的例程，完成 Matlab 环境下调用 C++代码（实现 Dijkstra 迪杰斯特拉算法），计算网络的单源最短路径。

## 三、功能描述、上机程序（含必要的注释）、上机调试运行结果

1. 利用 Cpp 代码抽取 Matlab 矩阵的某些行、某些列上的元素，返回提取到的矩阵数据到 Matlab：

新建一个 MtoCResizeArray.cpp 文件(MtoCResizeArray 将作为 MATLAB 环境中调用的函数名)，代码如下：

```
#include "mex.h"

// 定义 mexFunction 函数，这是 MEX 文件中必须定义的函数，用于实现 MATLAB 和 C/C++之间的接口
void mexFunction(int nlhs, mxArray* plhs[], int nrhs, const mxArray* prhs[])
{
    // 检查输入参数的数量是否正确，应该有 3 个输入参数
    if (nrhs != 3)
    {
        mexErrMsgTxt("参数个数不正确!");
    }

    // 获取第一个输入参数（矩阵）的行数和列数
    int rowNum = mxGetM(prhs[0]);
```

```
    int colNum = mxGetN(prhs[0]);

    // 获取第一个输入参数（矩阵）的指针，用于访问矩阵元素
    double* pArr = (double*)mxGetPr(prhs[0]);

    // 获取选择的行和列的信息
    // 无论是行向量还是列向量均支持
    double* pSelRows = (double*)mxGetPr(prhs[1]); // 选择的行
    double* pSelCols = (double*)mxGetPr(prhs[2]); // 选择的列

    // 获取选择的行的维度信息
    int selRowsRowNum = mxGetM(prhs[1]);
    int selRowsColNum = mxGetN(prhs[1]);
    // 检查选择的行是否为行向量或列向量
    if (selRowsRowNum != 1 && selRowsColNum != 1)
    {
        mexErrMsgTxt("行参数不正确！");
    }
    // 计算选择的行的数量
    int selRowsNum = selRowsRowNum * selRowsColNum;

    // 获取选择的列的维度信息
    int selColsRowNum = mxGetM(prhs[2]);
    int selColsColNum = mxGetN(prhs[2]);
    // 检查选择的列是否为行向量或列向量
    if (selColsRowNum != 1 && selColsColNum != 1)
    {
        mexErrMsgTxt("列参数不正确！");
    }
    // 计算选择的列的数量
    int selColsNum = selColsRowNum * selColsColNum;

    // 创建输出参数1，用于返回选择的行和列的数量
    plhs[1] = mxCreateDoubleMatrix(2, 1, mxREAL);
    double* resizedDims = (double*)mxGetPr(plhs[1]);
    resizedDims[0] = selRowsNum; // 选择的行数
    resizedDims[1] = selColsNum; // 选择的列数

    // 创建输出参数0，用于返回选择后的矩阵
    plhs[0] = mxCreateDoubleMatrix(selRowsNum, selColsNum, mxREAL);
    double* pResizedArr = (double*)mxGetPr(plhs[0]);

    // 定义宏，用于简化数组访问，MATLAB 中数据是按列优先存储的
#define ARR(row,col) pArr[(col)*rowNum+row]
```

```
#define RARR(row,col) pResizedArr[(col)*selRowsNum+row]

    // 遍历选择的行和列，将原矩阵中对应的元素复制到新的矩阵中
    for (int ri = 0; ri < selRowsNum; ri++)
    {
        for (int ci = 0; ci < selColsNum; ci++)
        {
            RARR(ri, ci) = ARR((int)pSelRows[ri] - 1, (int)pSelCols[ci] - 1);
        }
    }

    // 打印信息，表示函数执行成功
    mexPrintf("OK!\n");
}
```

编译 C++函数为 MEX 函数：将 MtoCResizeArray.cpp 放在 MATLAB 当前目录中，在 MATLAB 中输入 mex MtoCResizeArray.cpp，编译成功后将会生成 MtoCResizeArray.mexW32：

```
>> mex MtoCResizeArray.cpp
使用 'Microsoft Visual C++ 2022' 编译。
MEX 已成功完成。
>>
```

**图 1　编译程序**

在 Matlab 环境中调用函数实现预定的功能，理解观察程序运行：

```
arr=[11:19;21:29;31:39;41:49;51:59;61:69]
selRows=[1 3];
selCols=[2:4 5 9];
[rarr,rdims]=MtoCResizeArray(arr,selRows,selCols)
```

```
arr=[11:19;21:29;31:39;41:49;51:59;61:69]

arr = 6×9
    11    12    13    14    15    16    17    18    19
    21    22    23    24    25    26    27    28    29
    31    32    33    34    35    36    37    38    39
    41    42    43    44    45    46    47    48    49
    51    52    53    54    55    56    57    58    59
    61    62    63    64    65    66    67    68    69

selRows=[1 3];
selCols=[2:4 5 9];
[rarr,rdims]=MtoCResizeArray(arr,selRows,selCols)

OK!
rarr = 2×5
    12    13    14    15    19
    32    33    34    35    39

rdims = 2×1
     2
     5
```

**图 2　运行结果**

C++代码读取 Matlab 中的 matrix 类型数据：利用 Matlab 产生一个 5*3 矩阵，利用 c++读取其中位置为(row,col)的值，同时 c++向 Matlab 传递一个新矩阵，代码如下：

```
#include "mex.h"
#include <cstdio> // 包含标准输入输出库，用于printf函数

// 定义mexFunction函数，这是MEX文件中必须定义的函数，用于实现MATLAB和C/C++之间的接口
void mexFunction(int nlhs, mxArray* plhs[], int nrhs, mxArray* prhs[])
 {
    // 获取第一个输入参数（矩阵）的指针，用于访问矩阵元素
    double* input = mxGetPr(prhs[0]);

    // 获取第一个输入参数（矩阵）的行数和列数
     size_t m = mxGetM(prhs[0]); // 行数
     size_t n = mxGetN(prhs[0]); // 列数

    // 打印矩阵的行数和列数
    printf("the row of matrix is %d\n", m);
    printf("the column of matrix is %d\n", n);

    // 获取第二个和第三个输入参数（行和列索引），这里假设它们是标量
    size_t row = mxGetScalar(prhs[1]); // 行索引
    size_t col = mxGetScalar(prhs[2]); // 列索引

    // 打印指定位置的元素值
    // 注意：MATLAB索引从1开始，C/C++索引从0开始，所以需要减1
    printf("the data of row %d column %d is: %f\n", row, col, *(inpu
t + (col - 1) * m + row - 1));

    // 创建一个与输入矩阵同样大小的输出矩阵
    plhs[0] = mxCreateDoubleMatrix((mwSize)m, (mwSize)n, mxREAL);

    // 获取输出矩阵的指针，用于设置矩阵元素
    double* c = mxGetPr(plhs[0]);

    // 设置输出矩阵的特定元素值
    c[0] = 5; // 设置第一个元素的值
    c[1] = 6; // 设置第二个元素的值
    c[10] = 33; // 设置第11个元素的值（索引从0开始）
}
```

在 matlab 控制台运行 mex readMatrix.cpp：

```
>> mex readMatrix.cpp
使用 'Microsoft Visual C++ 2022' 编译。
MEX 已成功完成。
fx >>
```

**图 3　编译程序**

在 Matlab 中的操作，理解观察程序运行：

```
1. A=rand(5,3)
2. row=4;col=2;
3. ans=readMatrix(A,row,col)
```

```
A=rand(5,3)

A = 5×3
    0.8147    0.0975    0.1576
    0.9058    0.2785    0.9706
    0.1270    0.5469    0.9572
    0.9134    0.9575    0.4854
    0.6324    0.9649    0.8003

row=4;col=2;
ans=readMatrix(A,row,col)

the row of matrix is 5
the column of matrix is 3
the data of row 4 column 2 is:  0.957507
 ans = 5×3
     5     0    33
     6     0     0
     0     0     0
     0     0     0
     0     0     0
```

**图 4　运行结果**

2. C++代码读取 struct 类型数据

利用 MATLAB 产生 struct 类型的数据。以下 stu 是一个向量（一维数组），每个元素是一个结构，该结构包括 3 个字段，分别是 name、list、id，其类型分别是 string、matrix、double，代码如下：

```
1. #include "mex.h"
2. #include <cstdio> // 包含标准输入输出库，用于printf函数
3.
4. // 定义mexFunction函数，这是MEX文件中必须定义的函数，用于实现MATLAB和C/C++之间的接口
5. void mexFunction(int nlhs, mxArray* plhs[], int nrhs, mxArray* prhs[])
 {
6.     size_t m, n;
7.     // 检查第一个输入参数是否为结构体数组
8.     if (mxIsStruct(prhs[0])) {
9.         m = mxGetM(prhs[0]); // 获取结构体数组的行数，即结构体的数量
10.         n = mxGetN(prhs[0]); // 获取结构体数组的列数，这里应该是1
11.         size_t numStruct = m * n; // 计算结构体的总数
12.         printf("the total number of struct is %d\n", numStruct);
13.
```

```
        // 遍历每个结构体
        for (size_t i = 0; i < numStruct; i++) {
            printf("===the info of the %d student====\n", i + 1);

            // 读取结构体中的标量字段"id"
            mxArray* ID = mxGetField(prhs[0], i, "id"); // 获取字段"id"的mxArray 指针
            if (ID) {
                double id = mxGetScalar(ID); // 获取标量值
                printf("id=%f\n", id);
            }

            // 读取结构体中的矩阵字段"list"
            printf("matrix=\n");
            mxArray* tmpMatrix = mxGetField(prhs[0], i, "list"); // 获取字段"list"的mxArray 指针
            if (tmpMatrix) {
                double* tmpmatrix = mxGetPr(tmpMatrix); // 获取矩阵的指针

                m = mxGetM(tmpMatrix); // 获取矩阵的行数
                n = mxGetN(tmpMatrix); // 获取矩阵的列数
                // 遍历矩阵并打印每个元素
                for (size_t i = 0; i < m; i++) {
                    for (size_t j = 0; j < n; j++) {
                        printf("%f ", *(tmpmatrix + j * m + i));
                    }
                    printf("\n");
                }
            }

            // 读取结构体中的字符串字段"name"
            mxArray* tmpString = mxGetField(prhs[0], i, "name"); // 获取字段"name"的mxArray 指针
            if (tmpString) {
                char* tmp = mxArrayToString(tmpString); // 将mxArray转换为C 字符串
                if (tmp)
                    printf("name=%s\n", tmp);
                // 释放由mxArrayToString 分配的内存
                mxFree(tmp);
            }
        }
    }
}
```

先在 matlab 控制台运行 mex readStruct.cpp：

```
>> mex readStruct.cpp
使用 'Microsoft Visual C++ 2022' 编译。
MEX 已成功完成。
fx >>
```

**图 5　编译程序**

在 Matlab 中的操作，观察、理解程序运行：

```
1. stu(1).name='aa';
2. stu(2).name='bb';
3. stu(3).name='cc';
4. for i=1:3
5. stu(i).list=rand(5,4);
6. stu(i).id=randi(100);
7. end
8. readStruct(stu);
```

```
stu(1).name='aa';
stu(2).name='bb';
stu(3).name='cc';
for i=1:3
stu(i).list=rand(5,4);
stu(i).id=randi(100);
end
readStruct(stu);

the total number of struct is 3
===the info of the 1 student====
id=28.000000
matrix=
0.706046 0.823458 0.438744 0.489764
0.031833 0.694829 0.381558 0.445586
0.276923 0.317099 0.765517 0.646313
0.046171 0.950222 0.795200 0.709365
0.097132 0.034446 0.186873 0.754687
name=aa
===the info of the 2 student====
id=26.000000
matrix=
0.679703 0.959744 0.255095 0.547216
0.655098 0.340386 0.505957 0.138624
0.162612 0.585268 0.699077 0.149294
0.118998 0.223812 0.890903 0.257508
0.498364 0.751267 0.959291 0.840717
name=bb
===the info of the 3 student====
id=54.000000
matrix=
0.814285 0.251084 0.585264 0.753729
0.243525 0.616045 0.549724 0.380446
0.929264 0.473289 0.917194 0.567822
0.349984 0.351660 0.285839 0.075854
0.196595 0.830829 0.757200 0.053950
name=cc
```

**图 6　运行结果**

3. 模仿 3.1 的例程，完成 Matlab 环境下调用 C++代码（实现 Dijkstra 迪杰斯特拉算法），计算网络的单源最短路径

算法思想：

设 G=(V,E)是一个带权有向图，把图中顶点集合 V 分为两组，第一组为已

求出最短路径的顶点集合（用 S 表示，初始时 S 中只有一个源点，以后每求得一条最短路径 ，就将加入到集合 S 中，直到全部顶点都加入到 S 中，算法就结束了）,第二组为其余未确定最短路径的顶点集合（用 U 表示）,按最短路径的的递增次序依次把第二组中的顶点加入 S 中。在加入的过程中，总保持从源点 v 到 S 中各个顶点的最短路径长度不大于从源点 v 到 U 中任何路径的长度。此外，每个顶点对应一个距离，S 中的顶点的距离就是从 v 到此顶点的最短路径长度，U 中的顶点的距离，是从 v 到此顶点只包括 S 中的顶点为中间顶点的当前路径的最短长度。

算法步骤：

a.初始时，只包括源点，即 S = {v}，v 的距离为 0。U 包含除 v 以外的其他顶点，即：U ={其余顶点}，若 v 与 U 中顶点 u 有边，则(u,v)为正常权值，若 u 不是 v 的出边邻接点,则(u,v)权值 ∞;

b.从 U 中选取一个距离 v 最小的顶点 k，把 k，加入 S 中（该选定的距离就是 v 到 k 的最短路径长度）。

c.以 k 为新考虑的中间点，修改 U 中各顶点的距离；若从源点 v 到顶点 u 的距离（经过顶点 k）比原来距离（不经过顶点 k）短，则修改顶点 u 的距离值，修改后的距离值的顶点 k 的距离加上边上的权。

d.重复步骤 b 和 c 直到所有顶点都包含在 S 中。

代码如下：

```
#include "mex.h"
#include <vector>
#include <limits>
#include <cmath>

const int INF = 32767;
const int MAXN = 10;

// Dijkstra 算法的实现
void Dijkstra(int v, const double A[MAXN][MAXN], double dis[MAXN], int path[MAXN]) {
    bool S[MAXN] = { false };
    int n = MAXN;
    for (int i = 0; i < n; ++i) {
        dis[i] = A[v][i];
        if (dis[i] == INF) dis[i] = NAN; // 如果不可达，设置为NaN
```

```
16.         path[i] = -1;
17.     }
18. 
19.     dis[v] = 0;
20.     S[v] = true;
21. 
22.     for (int i = 0; i < n; ++i) {
23.         int u = -1;
24.         double minDist = INF;
25. 
26.         // 寻找不在 S 中且距离最小的顶点 u
27.         for (int j = 0; j < n; ++j) {
28.             if (!S[j] && (u == -1 || dis[j] < minDist)) {
29.                 u = j;
30.                 minDist = dis[j];
31.             }
32.         }
33. 
34.         if (u == -1) break; // 剩余顶点都不可达
35. 
36.         S[u] = true;
37. 
38.         // 更新顶点 u 相邻的顶点的距离
39.         for (int j = 0; j < n; ++j) {
40.             if (!S[j] && A[u][j] != INF && dis[u] + A[u][j] < dis[j])
    {
41.                 dis[j] = dis[u] + A[u][j];
42.                 path[j] = u;
43.             }
44.         }
45.     }
46. }
47. 
48. // MEX 文件的入口点
49. void mexFunction(int nlhs, mxArray* plhs[], int nrhs, const mxArray*
    prhs[]) {
50. 
51.     // 获取输入参数
52.     int src = static_cast<int>(mxGetScalar(prhs[0]));
53.     double* graphPtr = mxGetPr(prhs[1]);
54. 
55.     // 创建输出参数
56.     plhs[0] = mxCreateDoubleMatrix(MAXN, 1, mxREAL);
57.     plhs[1] = mxCreateDoubleMatrix(MAXN, 1, mxREAL);
```

```
58.
59.     // 将输入图转换为C++数组
60.     double A[MAXN][MAXN];
61.     for (int i = 0; i < MAXN; ++i) {
62.         for (int j = 0; j < MAXN; ++j) {
63.             A[i][j] = graphPtr[i + j * MAXN];
64.         }
65.     }
66.
67.     // 调用Dijkstra 算法
68.     double* disPtr = mxGetPr(plhs[0]);
69.     int* pathPtr = reinterpret_cast<int*>(mxGetPr(plhs[1])); // 使用
reinterpret_cast 进行类型转换
70.     double dis[MAXN];
71.     int path[MAXN];
72.     Dijkstra(src, A, dis, path);
73.
74.     // 将结果复制回Matlab 矩阵
75.     for (int i = 0; i < MAXN; ++i) {
76.         disPtr[i] = dis[i];
77.         if (path[i] == -1) {
78.             pathPtr[i] = 0; // 如果没有路径，则设置为0
79.         }
80.         else {
81.             pathPtr[i] = path[i] + 1; // MATLAB 索引从1 开始
82.         }
83.     }
84. }
```

在 matlab 控制台运行 mex dijkstra.cpp：

```
>> mex dijkstra.cpp
使用 'Microsoft Visual C++ 2022' 编译。
MEX 已成功完成。
>>
```

**图 7 编译程序**

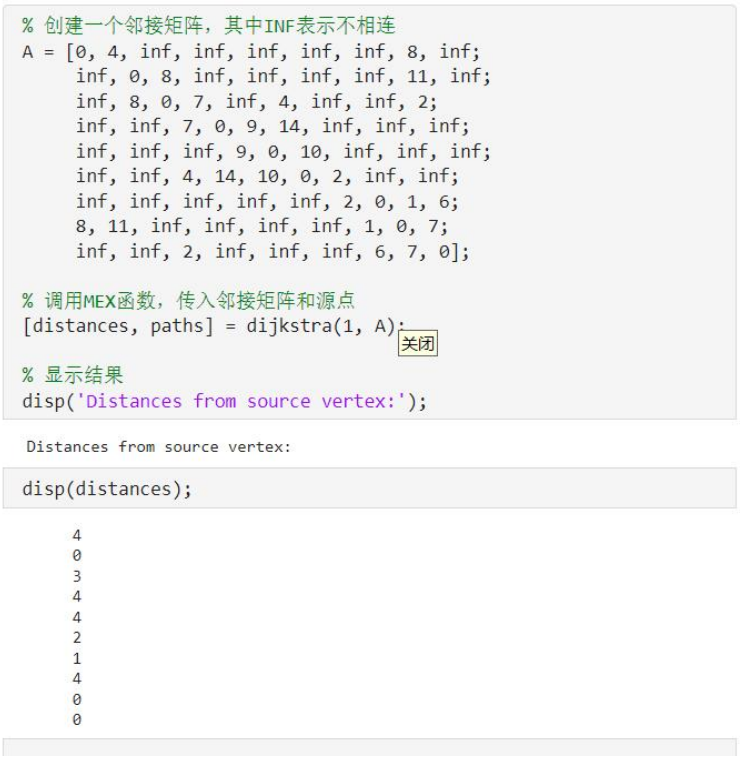
在 Matlab 中的操作，观察、理解程序运行：

```
1. % 创建一个邻接矩阵，其中 INF 表示不相连
2. A = [0, 4, inf, inf, inf, inf, inf, 8, inf;
3.      inf, 0, 8, inf, inf, inf, inf, 11, inf;
4.      inf, 8, 0, 7, inf, 4, inf, inf, 2;
5.      inf, inf, 7, 0, 9, 14, inf, inf, inf;
6.      inf, inf, inf, 9, 0, 10, inf, inf, inf;
7.      inf, inf, 4, 14, 10, 0, 2, inf, inf;
```

```
      inf, inf, inf, inf, inf, 2, 0, 1, 6;
      8, 11, inf, inf, inf, inf, 1, 0, 7;
       inf, inf, 2, inf, inf, inf, 6, 7, 0];

% 调用 MEX 函数，传入邻接矩阵和源点
[distances, paths] = Dijkstra(1, A);

% 显示结果
disp(distances);
```



```
% 创建一个邻接矩阵，其中INF表示不相连
A = [0, 4, inf, inf, inf, inf, inf, 8, inf;
     inf, 0, 8, inf, inf, inf, inf, 11, inf;
     inf, 8, 0, 7, inf, 4, inf, inf, 2;
     inf, inf, 7, 0, 9, 14, inf, inf, inf;
     inf, inf, inf, 9, 0, 10, inf, inf, inf;
     inf, inf, 4, 14, 10, 0, 2, inf, inf;
     inf, inf, inf, inf, inf, 2, 0, 1, 6;
     8, 11, inf, inf, inf, inf, 1, 0, 7;
     inf, inf, 2, inf, inf, inf, 6, 7, 0];

% 调用MEX函数，传入邻接矩阵和源点
[distances, paths] = dijkstra(1, A);
关闭
% 显示结果
disp('Distances from source vertex:');
```

```
Distances from source vertex:
```

```
disp(distances);
```

```
     4
     0
     3
     4
     4
     2
     1
     4
     0
     0
```

**图 8　运行结果**

## 四、上机总结及感想

在本次网络科学基础第三次上机实验中，我深入学习了 Matlab 与 C++代码的交互，包括如何在 C++中处理 Matlab 矩阵和结构体数据，并将结果反馈给 Matlab。通过编写和调用 MEX 函数，我不仅提升了编程技能，还加深了对数据结构和算法的理解。特别是在实现 Dijkstra 迪杰斯特拉算法计算网络单源最短路径的过程中，我对图论和网络分析有了更深刻的认识。这次实验不仅增强了我的问题解决能力，也激发了我对网络科学和编程深入学习的热情，我期待将这些知识应用到未来的研究和项目中。