哈尔滨工业大学计算学部

实验报告

课程名称：数据结构与算法

课程类型：专业核心基础课（必修）

实验项目：图型结构及其应用

实验题目：最短路径算法

实验日期：2023/10/25

班级：2203101

学号：2022113573

姓名：张宇杰

设计成绩	报告成绩	指导老师
		张岩

一、实验目的

最短路径问题研究的主要有：单源最短路径问题和所有顶点对之间的最短路径问题。在计算机领域和实际工程中具有广泛的应用，如集成电路设计、GPS/游戏地图导航、智能交通、路由选择、铺设管线等。本实验要求设计和实现 Dijkstra 算法和 Floyd-Warshall 算法，求解最短路径问题。

二、实验要求及实验环境

实验要求：

1．实现单源最短路径的 Dijkstra 算法，输出源点及其到其他顶点的最短路径长度和最短路径

2．实现全局最短路径的 Floyd-Warshall 算法。计算任意两个顶点间的最短距离矩阵和最短路径矩阵，并输出任意两个顶点间的最短路径长度和最短路径

3．利用 Dijkstra 或 Floyd-Warshall 算法解决单目标最短路径问题：找出图中每个顶点 v 到某个指定顶点 c 最短路径

4．利用 Dijkstra 或 Floyd-Warshall 算法解决单顶点对间最短路径问题：对于某对顶点 u 和 v，找出 u 到 v 和 v 到 u 的一条最短路径

5．以文件形式输入图的顶点和边，并以适当的方式展示相应的结果。要求顶点不少于 10 个，边不少于 13 个

6．选做：实现 Warshall 算法，计算有向图的可达矩阵，理解可达矩阵的含义

7．选做：利用堆结构（优先级队列）改进和优化 Dijkstra 算法，实现改进和优化的 Dijkstra 算法，并与原算法进行实验比较

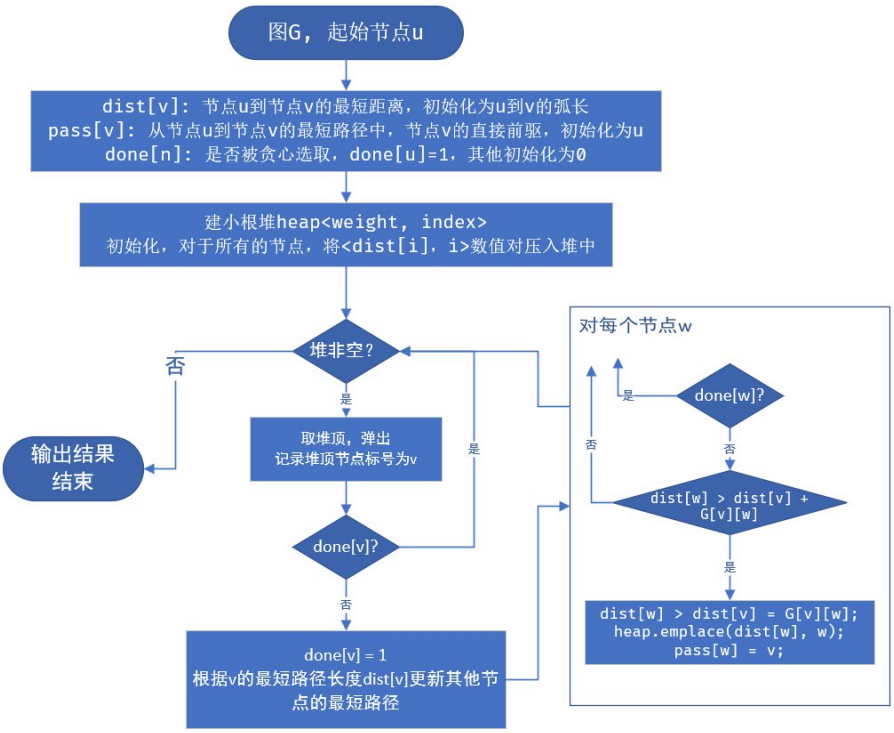
实验环境：

Windows 11, g++, gdb, VS2022

三、设计思想（本程序中的用到的所有数据类型的定义，主程序的流程图及各程序模块之间的调用关系、核心算法的主要步骤）

1．逻辑设计

核心算法
dijkstra流程图



floyd 本身太简单了就没画了

2．物理设计（即存储结构设计）

使用自定义类 AdjacencyMatrixGraph

使用 STL 容器 vector, stack, priority_queue

所有算法函数原型如下：

```
1.    namespace MyGraph {
2.
3.     void dijkstra(const AdjacencyMatrixGraph& G, label label_from, :
           :std::ostream& out_dest = ::std::cout);
4.
5.     void __dijkstra(const AdjacencyMatrixGraph& G, index u, ::std::v
           ector<weight>& dist, ::std::vector<index>& pass);
6.
7.     void __dijkstra_show_path(const AdjacencyMatrixGraph& G, index u
           , index v, ::std::vector<weight>& dist, ::std::vector<index
           >& pass, ::std::ostream& out_dest);
8.
9.     void floyd(const AdjacencyMatrixGraph& G, ::std::ostream& out_de
           st = ::std::cout);
10.
11.    void warshall(const AdjacencyMatrixGraph& G, ::std::ostream& out
           _dest = ::std::cout);
```

```
12.
13.     void all_shortest_to(const AdjacencyMatrixGraph& G, label label_
            to, ::std::ostream& out_dest = ::std::cout);
14.
15.     void shortest_pair(const AdjacencyMatrixGraph& G, label label_fr
            om, label label_to, ::std::ostream& out_dest = ::std::cout)
            ;
16.
17.   } // namespace MyGraph
```

自定义数据类型 AdjacencyMatrixGraph 的定义

```
1.    class AdjacencyMatrixGraph: public Graph<matrixAdjIter> {
2.
3.            friend class matrixAdjIter;
4.            friend AdjacencyListGraph matrixToList(const AdjacencyMat
            rixGraph& G);
5.            friend AdjacencyMatrixGraph listToMatrix(const AdjacencyL
            istGraph& G);
6.
7.        public:
8.            using adjIterator = matrixAdjIter;
9.
10.       private:
11.           using matrix = ::std::vector<::std::vector<weight>>;
12.           matrix m_matrix;
13.
14.           static constexpr int initialMatrixSize = 20;
15.           static constexpr int extendMatrixSize = 5;
16.
17.       private:
18.           void extendMatrix();
19.
20.       public:
21.           AdjacencyMatrixGraph();
22.
23.           virtual void addVertex(label vertex_add);
24.
25.           virtual void addEdge(index vertex_from, index vertex_to,
            weight w) noexcept;
26.
27.           virtual weight getEdgeWeight(index vertex_from, index ver
            tex_to) const noexcept;
28.
29.           virtual void removeVertex(index vertex_index) noexcept;
```

```
30.
31.            virtual void removeEdge(index vertex_from, index vertex_t
           o) noexcept;
32.
33.            virtual adjIterator beginAdjacentIterOf(index idx);
34.
35.            virtual adjIterator endAdjacentIterOf(index idx);
36.
37.            void showMatrix(::std::ostream& out = ::std::cout) const;
38.
39.        }; // class AdjacencyMatrixGraph
```

四、测试结果（包括测试数据、结果数据及结果的简单分析和结论，可以用截图得形式贴入此报告）

以下语境中的图 1(graph_1)指：

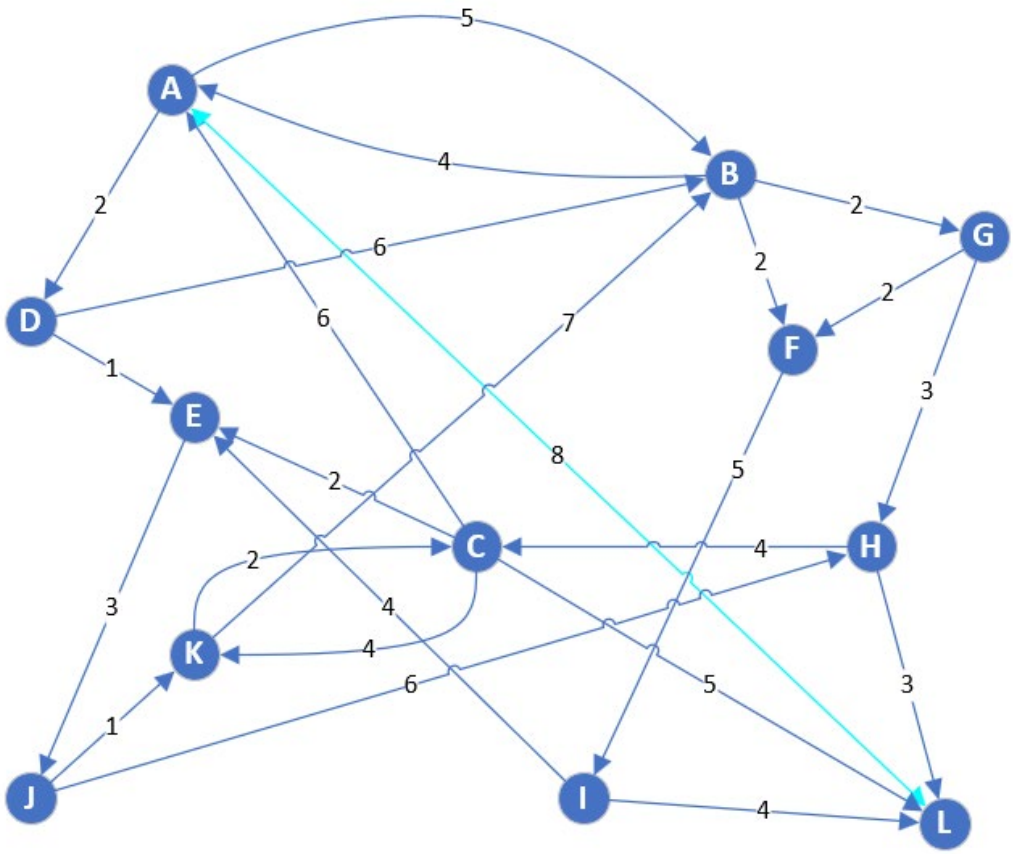
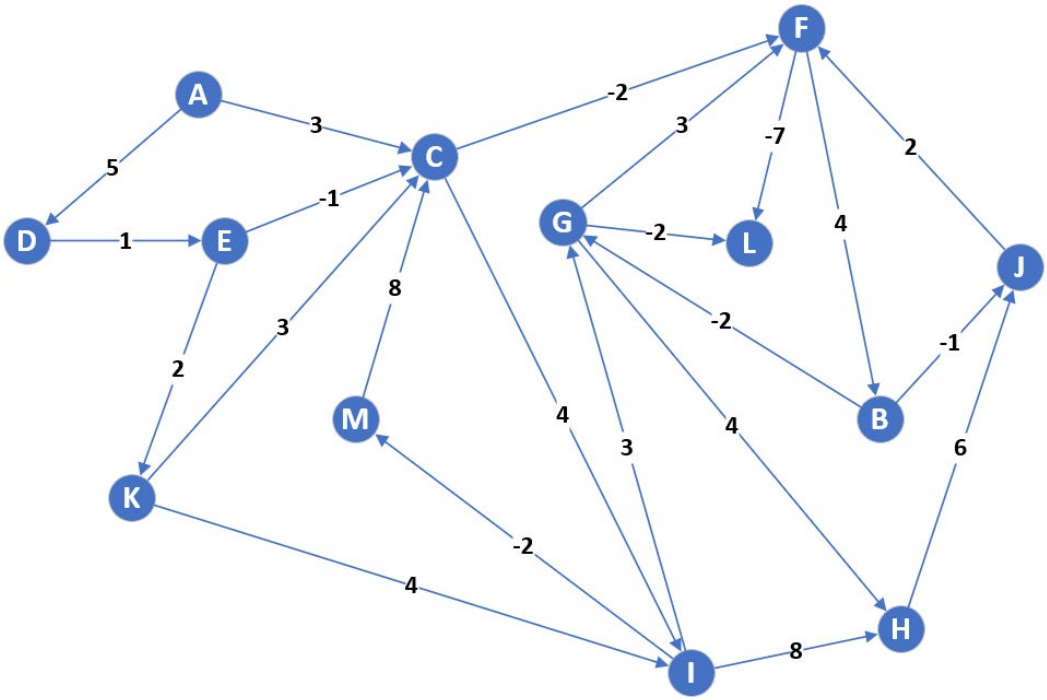


图 2(graph_2)指：



1．实现单源最短路径的 Dijkstra 算法，输出源点及其到其他顶点的最短路径长度和最短路径

运行程序：

```
#include <iostream>
#include <fstream>

#include ".\src\header\MyGraph"
#include ".\src\ShortestPathAlgorithm"
using namespace MyGraph;
using namespace std;

int main()
{
    system("chcp 65001"); // set terminal to UTF-8 mode, for showing Chinese characters currectly
    ifstream command("./graph_1.txt");

    AdjacencyMatrixGraph G1;
    G1.commandSequence(command);

    dijkstra(G1, "B");

    return 0;
}
```

输出结果：

```
Active code page: 65001
B->A:   4       Path: B->A
B->C:   9       Path: B->G->H->C
B->D:   6       Path: B->A->D
B->E:   7       Path: B->A->D->E
B->F:   2       Path: B->F
B->G:   2       Path: B->G
B->H:   5       Path: B->G->H
B->I:   7       Path: B->F->I
B->J:   10      Path: B->A->D->E->J
B->K:   11      Path: B->A->D->E->J->K
B->L:   8       Path: B->G->H->L
```

结果正确

2．实现全局最短路径的 Floyd-Warshall 算法。计算任意两个顶点间的最短距离矩阵和最短路径矩阵，并输出任意两个顶点间的最短路径长度和最短路径

输入程序：

```
1.    #include <iostream>
2.    #include <fstream>
3.
4.    #include ".\src\header\MyGraph"
5.    #include ".\src\ShortestPathAlgorithm"
6.    using namespace MyGraph;
7.    using namespace std;
8.
9.    int main()
10.   {
11.       system("chcp 65001"); // set terminal to UTF-8 mode, for show
            ing Chinese characters currectly
12.       ifstream command("./graph_2.txt");
13.
14.       AdjacencyMatrixGraph G2;
15.       G2.commandSequence(command);
16.
17.       floyd(G2);
18.
19.       return 0;
20.   }
```

输出结果：

```
(base) PS D:\File\大二秋\DSA\实验3 图上算法\shortest-path> ./test
Active code page: 65001
最短距离矩阵:
        A       B       C       D       E       F       G       H       I       J       K       L       M
A       0       5       3       5       6       1       3       7       7       4       8       -6      5
B       inf     0       inf     inf     inf     1       -2      2       inf     -1      inf     -6      inf
C       inf     2       0       inf     inf     -2      0       4       4       1       inf     -9      2
D       inf     2       0       0       1       -2      0       4       4       1       3       -9      2
E       inf     1       -1      inf     0       -3      -1      3       3       0       2       -10     1
F       inf     4       inf     inf     inf     0       2       6       inf     3       inf     -7      inf
G       inf     7       inf     inf     inf     3       0       4       inf     6       inf     -4      inf
H       inf     12      inf     inf     inf     8       10      0       inf     6       inf     1       inf
I       inf     8       6       inf     inf     4       3       7       0       7       inf     -3      -2
J       inf     6       inf     inf     inf     2       4       8       inf     0       inf     -5      inf
K       inf     5       3       inf     inf     1       3       7       4       4       0       -6      2
L       inf     inf     inf     inf     inf     inf     inf     inf     inf     inf     inf     0       inf
M       inf     10      8       inf     inf     6       8       12      12      9       inf     -1      0

最短路径矩阵:
        A       B       C       D       E       F       G       H       I       J       K       L       M
A               F                       D       C       F       G       C       F       E       F       I
B                                       G               G                               G
C               F                               F       G               F               F       I
D               F       E                       E       F       G       E       F       E       F       I
E               F                               C       F       G       C       F               F       I
F                                               B       G               B
G               F                                                       F               F
H               J                               J       J                               J
I               M       M                       M               G               M               M
J               F                                       F       G                               F
K               F                               C       F       G               F               F       I
L
M               F                               C       F       G       C       F               F

最短路径展示:
A->B:   5       Path: A->C->F->B
```

```
最短路径展示:
A->B:   5       Path: A->C->F->B
A->C:   3       Path: A->C
A->D:   5       Path: A->D
A->E:   6       Path: A->D->E
A->F:   1       Path: A->C->F
A->G:   3       Path: A->C->F->B->G
A->H:   7       Path: A->C->F->B->G->H
A->I:   7       Path: A->C->I
A->J:   4       Path: A->C->F->B->J
A->K:   8       Path: A->D->E->K
A->L:   -6      Path: A->C->F->L
A->M:   5       Path: A->C->I->M
B->A:   inf
B->C:   inf
B->D:   inf
B->E:   inf
B->F:   1       Path: B->G->F
B->G:   -2      Path: B->G
B->H:   2       Path: B->G->H
B->I:   inf
B->J:   -1      Path: B->J
B->K:   inf
B->L:   -6      Path: B->G->F->L
B->M:   inf
C->A:   inf
C->B:   2       Path: C->F->B
C->D:   inf
C->E:   inf
C->F:   -2      Path: C->F
C->G:   0       Path: C->F->B->G
C->H:   4       Path: C->F->B->G->H
C->I:   4       Path: C->I
C->J:   1       Path: C->F->B->J
C->K:   inf
C->L:   -9      Path: C->F->L
C->M:   2       Path: C->I->M
D->A:   inf
D->B:   2       Path: D->E->C->F->B
D->C:   0       Path: D->E->C
D->E:   1       Path: D->E
D->F:   -2      Path: D->E->C->F
D->G:   0       Path: D->E->C->F->B->G
D->H:   4       Path: D->E->C->F->B->G->H
D->I:   4       Path: D->E->C->I
D->J:   1       Path: D->E->C->F->B->J
D->K:   3       Path: D->E->K
D->L:   -9      Path: D->E->C->F->L
D->M:   2       Path: D->E->C->I->M
E->A:   inf
E->B:   1       Path: E->C->F->B
E->C:   -1      Path: E->C
E->D:   inf
E->F:   -3      Path: E->C->F
E->G:   -1      Path: E->C->F->B->G
E->H:   3       Path: E->C->F->B->G->H
E->I:   3       Path: E->C->I
E->J:   0       Path: E->C->F->B->J
E->K:   2       Path: E->K
E->L:   -10     Path: E->C->F->L
E->M:   1       Path: E->C->I->M
F->A:   inf
F->B:   4       Path: F->B
F->C:   inf
F->D:   inf
F->E:   inf
F->G:   2       Path: F->B->G
F->H:   6       Path: F->B->G->H
F->I:   inf
F->J:   3       Path: F->B->J
F->K:   inf
F->L:   -7      Path: F->L
F->M:   inf
G->A:   inf
G->B:   7       Path: G->F->B
G->C:   inf
G->D:   inf
G->E:   inf
G->F:   3       Path: G->F
G->H:   4       Path: G->H
G->I:   inf
G->J:   6       Path: G->F->B->J
G->K:   inf
G->L:   -4      Path: G->F->L
G->M:   inf
H->A:   inf
H->B:   12      Path: H->J->F->B
H->C:   inf
H->D:   inf
H->E:   inf
H->F:   8       Path: H->J->F
H->G:   10      Path: H->J->F->B->G
H->I:   inf
H->J:   6       Path: H->J
H->K:   inf
H->L:   1       Path: H->J->F->L
H->M:   inf
I->A:   inf
I->B:   8       Path: I->M->C->F->B
I->C:   6       Path: I->M->C
I->D:   inf
I->E:   inf
I->F:   4       Path: I->M->C->F
I->G:   3       Path: I->G
I->H:   7       Path: I->G->H
I->J:   7       Path: I->M->C->F->B->J
I->K:   inf
```

输出符合预期

3．利用 Dijkstra 或 Floyd-Warshall 算法解决单目标最短路径问题：找出图中每个顶点 v 到某个指定顶点 c 最短路径

运行程序：

```cpp
#include <iostream>
#include <fstream>

#include ".\src\header\MyGraph"
#include ".\src\ShortestPathAlgorithm"
using namespace MyGraph;
using namespace std;

int main()
{
    system("chcp 65001"); // set terminal to UTF-8 mode, for showing Chinese characters currectly
    ifstream command("./graph_1.txt");

    AdjacencyMatrixGraph G1;
    G1.commandSequence(command);

    all_shortest_to(G1, "F");

    return 0;
}
```

输出结果：

```
(base) PS D:\File\大二秋\DSA\实验3 图上算法\shortest-path> ./test
Active code page: 65001
A->F:   7       Path: A->B->F
B->F:   2       Path: B->F
C->F:   13      Path: C->A->B->F
D->F:   8       Path: D->B->F
E->F:   13      Path: E->J->K->B->F
G->F:   2       Path: G->F
H->F:   17      Path: H->C->A->B->F
I->F:   17      Path: I->E->J->K->B->F
J->F:   10      Path: J->K->B->F
K->F:   9       Path: K->B->F
L->F:   15      Path: L->A->B->F
(base) PS D:\File\大二秋\DSA\实验3 图上算法\shortest-path>
```

结果正确

4．利用 Dijkstra 或 Floyd-Warshall 算法解决单顶点对间最短路径问题：对于某对顶点 u 和 v，找出 u 到 v 和 v 到 u 的一条最短路径

运行程序：

```cpp
#include <iostream>
#include <fstream>

#include ".\src\header\MyGraph"
#include ".\src\ShortestPathAlgorithm"
using namespace MyGraph;
using namespace std;

int main()
{
    system("chcp 65001"); // set terminal to UTF-8 mode, for showing Chinese characters currectly
    ifstream command("./graph_1.txt");

    AdjacencyMatrixGraph G1;
    G1.commandSequence(command);

    shortest_pair(G1, "F", "C");

    return 0;
}
```

输出结果：

```
(base) PS D:\File\大二秋\DSA\实验3 图上算法\shortest-path> ./test
Active code page: 65001
F->C:   15      Path: F->I->E->J->K->C
C->F:   13      Path: C->K->B->F
(base) PS D:\File\大二秋\DSA\实验3 图上算法\shortest-path>
```

符合预期

五、经验体会与不足

体会：巩固了各类最短路径算法的实现

不足：使用邻接矩阵作为图的数据结构，以上算法未对邻接表数据结构进行实现

六、附录：源代码（带注释）

ShortestPathAlgorithm.h

```cpp
#ifndef _SHORTEST_PATH_ALGORITHM_H_INCLUDED_
#define _SHORTEST_PATH_ALGORITHM_H_INCLUDED_

#include <iostream>
```

```
5.  #include "./header/MyGraph"
6.
7.  namespace MyGraph {
8.
9.   void dijkstra(const AdjacencyMatrixGraph& G, label label_from, :
         :std::ostream& out_dest = ::std::cout);
10.
11.  void __dijkstra(const AdjacencyMatrixGraph& G, index u, ::std::v
         ector<weight>& dist, ::std::vector<index>& pass);
12.
13.  void __dijkstra_show_path(const AdjacencyMatrixGraph& G, index u
         , index v, ::std::vector<weight>& dist, ::std::vector<index
         >& pass, ::std::ostream& out_dest);
14.
15.  void floyd(const AdjacencyMatrixGraph& G, ::std::ostream& out_de
         st = ::std::cout);
16.
17.  void warshall(const AdjacencyMatrixGraph& G, ::std::ostream& out
         _dest = ::std::cout);
18.
19.  void all_shortest_to(const AdjacencyMatrixGraph& G, label label_
         to, ::std::ostream& out_dest = ::std::cout);
20.
21.  void shortest_pair(const AdjacencyMatrixGraph& G, label label_fr
         om, label label_to, ::std::ostream& out_dest = ::std::cout)
         ;
22.
23. } // namespace MyGraph
24.
25. #endif // _SHORTEST_PATH_ALGORITHM_H_INCLUDED_
```

dijkstra.cpp

```
1.  #include "ShortestPathAlgorithm.h"
2.
3.  #include <vector>
4.  #include <stack>
5.  #include <queue>
6.
7.  void MyGraph::__dijkstra(const AdjacencyMatrixGraph& G, index u,
         ::std::vector<weight>& dist, ::std::vector<index>& pass) {
8.   using namespace std;
9.
10.  // dist[v]: 节点 u 到节点 v 的最短路径长度
11.  // pass[v]: 从节点 u 到节点 v 的最短路径中，节点 v 的直接前驱
```

```
12.
13.    size n = G.countVertex(); // 图的顶点数
14.
15.    vector<char> done(n, 0); // done[u]: 节点是否被贪心选取
16.
17.    using pair = pair<weight, index>;
18.    priority_queue<pair, vector<pair>, greater<pair>> heap; // 小根
           堆，对 dijkstra 算法的优化
19.
20.    // ***** 初始化工作 *****
21.    for (index v = 0; v < n; ++v) {
22.     dist.at(v) = G.getEdgeWeight(u, v); // dist 数组初始化为 u 至 v 的弧
            长
23.     pass.at(v) = u;
24.     heap.emplace(dist.at(v), v); // 将<权值，标号>数值对压入堆中，将自
            动将这些数值对按照 weight 优先，index 其次进行排序
25.    }
26.    done.at(u) = 1; // 显然 u 已经完成计算
27.
28.    while (!heap.empty()) {
29.     pair p = heap.top(); // 取出堆顶
30.     heap.pop();
31.
32.     index v = p.second; // 当前未选取的节点中节点 u 到其距离最短的那个节
            点
33.
34.     if (done.at(v)) // 如果已经被贪心选取
35.      continue; // 抛弃这个节点
36.     else {
37.      done.at(v) = 1;
38.
39.      for (index w = 0; w < n; ++w) // 通过这个贪心选取的节点，去更新其
            他节点最短路径
40.       if (!done.at(w))
41.        if (dist.at(w) > dist.at(v) + G.getEdgeWeight(v, w)) {
42.         dist.at(w) = dist.at(v) + G.getEdgeWeight(v, w);
43.         heap.emplace(dist.at(w), w);
44.         pass.at(w) = v;
45.        }
46.     }
47.    }
48.  }
49.
```

```
50.  void MyGraph::__dijkstra_show_path(const AdjacencyMatrixGraph& G,
          index u, index v, ::std::vector<weight>& dist, ::std::vect
         or<index>& pass, ::std::ostream& out_dest) {
51.   using namespace std;
52.
53.   static stack<index> st; // 栈，用于输出路径
54.
55.   if (v == u)
56.    return;
57.
58.   out_dest << G.getLabel(u) << "->" << G.getLabel(v) << ":\t";
59.   if (dist.at(v) >= infinity) {
60.    out_dest << "inf" << endl; // 不可达
61.    return;
62.   }
63.   else
64.    out_dest << dist.at(v) << "\tPath: "; // 可达，先输出路径长度，再
        输出路径
65.
66.   index i = v;
67.   while (i != u) {
68.    st.push(i);
69.    i = pass.at(i);
70.   }
71.
72.   out_dest << G.getLabel(u);
73.   while (!st.empty()) {
74.    out_dest << "->" << G.getLabel(st.top());
75.    st.pop();
76.   }
77.
78.   out_dest << endl;
79.  }
80.
81.  void MyGraph::dijkstra(const AdjacencyMatrixGraph& G, label label
        _from, ::std::ostream& out_dest) {
82.   using namespace std;
83.
84.   if (!G.isVertex(label_from)) {
85.    cout << "Invalid label" << endl;
86.    return;
87.   }
88.
89.   size n = G.countVertex(); // 图的顶点数
```

```
    index u = G.getIndex(label_from); // 起点的编号u

    vector<weight> dist(n); // dist[v]: 节点u到节点v的最短路径长度
    vector<index> pass(n); // pass[v]: 从节点u到节点v的最短路径中，节点v的直接前驱

    __dijkstra(G, u, dist, pass);

    // ***** 结果展示 *****
    stack<index> st; // 栈，用于输出路径

    for (index v = 0; v < n; ++v) // 对每个节点，输出节点u到其的最短路径
     __dijkstra_show_path(G, u, v, dist, pass, cout);
}
```

floyd.cpp

```
#include "ShortestPathAlgorithm.h"

#include <vector>

void __show_path(const ::MyGraph::AdjacencyMatrixGraph& G, ::std::vector<::std::vector<::MyGraph::index>>& pass, ::MyGraph::index u, ::MyGraph::index v) {
 using namespace std;
 using namespace MyGraph;

 index k = pass[u][v];

 if (k != -1) {
  __show_path(G, pass, u, k);
  cout << G.getLabel(k) << "->";
  __show_path(G, pass, k, v);
 }
}

void MyGraph::floyd(const AdjacencyMatrixGraph& G, ::std::ostream& out_dest) {
 using namespace std;

 size n = G.countVertex(); // 图的顶点数

 vector<vector<weight>> dist(n, vector<weight>(n, 0)); // dist[u][v]: u到v的最短距离
```

```
24.   vector<vector<index>> pass(n, vector<index>(n, -1)); // pass[u][
          v]: u 到 v 的路径中包含的点, -1 为没有包含
25.
26.   for (index i = 0; i < n; ++i) // 初始化距离矩阵
27.    for (index j = 0; j < n; ++j) {
28.     if (i == j)
29.      dist[i][j] = 0;
30.     else
31.      dist[i][j] = G.getEdgeWeight(i, j);
32.    }
33.
34.   // floyd 算法核心
35.   for (index k = 0; k < n; ++k)
36.    for (index i = 0; i < n; ++i)
37.     for (index j = 0; j < n; ++j)
38.      if (dist[i][j] > dist[i][k] + dist[k][j] && dist[i][k] < infi
           nity && dist[k][j] < infinity) { // 动态规划
39.       dist[i][j] = dist[i][k] + dist[k][j];
40.       pass[i][j] = k;
41.      }
42.
43.   // ***** 结果展示 *****
44.
45.   out_dest << "最短距离矩阵:" << endl;
46.   for (index i = 0; i < n; ++i)
47.    out_dest << '\t' << G.getLabel(i);
48.   out_dest << endl;
49.
50.   for (index i = 0; i < n; ++i) {
51.    out_dest << G.getLabel(i) << '\t';
52.
53.    for (index j = 0; j < n; ++j)
54.     if (dist[i][j] >= infinity)
55.      out_dest << "inf" << '\t';
56.     else
57.      out_dest << dist[i][j] << '\t';
58.
59.    out_dest << endl;
60.   }
61.   out_dest << endl;
62.
63.   out_dest << "最短路径矩阵:" << endl;
64.   for (index i = 0; i < n; ++i)
65.    out_dest << '\t' << G.getLabel(i);
```

```
66.   out_dest << endl;
67.
68.   for (index i = 0; i < n; ++i) {
69.    out_dest << G.getLabel(i) << '\t';
70.
71.    for (index j = 0; j < n; ++j)
72.     out_dest << G.getLabel(pass[i][j]) << '\t';
73.
74.    out_dest << endl;
75.   }
76.   out_dest << endl;
77.
78.   out_dest << "最短路径展示:" << endl;
79.   for (index i = 0; i < n; ++i) // 对任意两个不相同的节点，都输出二者
          之间的最短路径
80.    for (index j = 0; j < n; ++j) {
81.     if (i == j)
82.      continue;
83.
84.     out_dest << G.getLabel(i) << "->" << G.getLabel(j) << ":\t";
85.     if (dist[i][j] >= infinity)
86.      out_dest << "inf" << endl; // 不可达
87.     else {
88.      out_dest << dist[i][j] << "\tPath: " << G.getLabel(i) << "->"
          ;
89.      __show_path(G, pass, i, j);
90.      out_dest << G.getLabel(j) << endl;
91.     }
92.    }
93.  }
```

warshall.cpp

```
1.    #include "ShortestPathAlgorithm.h"
2.
3.    #include <vector>
4.
5.    void MyGraph::warshall(const AdjacencyMatrixGraph& G, ::std::ostr
          eam& out_dest) {
6.     using namespace std;
7.
8.     size n = G.countVertex(); // 图的顶点数
9.
10.    vector<vector<char>> reachable(n, vector<char>(n, 0)); // dist[u
          ][v]: u 到 v 的最短距离
```

```
11.
12.     for (index i = 0; i < n; ++i)
13.      for (index j = 0; j < n; ++j) {
14.       if (i == j)
15.        reachable[i][j] = 1;
16.       else
17.        reachable[i][j] = G.getEdgeWeight(i, j) >= infinity ? 0 : 1;
18.      }
19.
20.     for (index k = 0; k < n; ++k)
21.      for (index i = 0; i < n; ++i)
22.       for (index j = 0; j < n; ++j)
23.        reachable[i][j] |= reachable[i][k] & reachable[k][j];
24.
25.     // *****结果展示*****
26.
27.     out_dest << "可达矩阵:" << endl;
28.     for (index i = 0; i < n; ++i)
29.      out_dest << '\t' << G.getLabel(i);
30.     out_dest << endl;
31.
32.     for (index i = 0; i < n; ++i) {
33.      out_dest << G.getLabel(i) << '\t';
34.
35.      for (index j = 0; j < n; ++j)
36.       out_dest << reachable[i][j] + 0 << '\t';
37.
38.      out_dest << endl;
39.     }
40.    }
```

all_shortest_to.cpp

```
1.     #include "ShortestPathAlgorithm.h"
2.
3.     #include <vector>
4.     #include <queue>
5.
6.     void MyGraph::all_shortest_to(const AdjacencyMatrixGraph& G, labe
           l label_to, ::std::ostream& out_dest) {
7.      using namespace std;
8.
9.      if (!G.isVertex(label_to)) {
10.      cout << "Invalid label" << endl;
11.      return;
```

```
    }

    size n = G.countVertex(); // 图的顶点数
    index v = G.getIndex(label_to); // 终点的编号 u

    vector<weight> dist(n); // dist[v]: 节点 u 到节点 v 的最短路径
    vector<index> pass(n); // pass[v]: 从节点 u 到节点 v 的最短路径中，节
        点 v 的直接前驱

    AdjacencyMatrixGraph G_T; // 图 G 的转置

    // 求 G 的转置
    for (index i = 0; i < n; ++i)
     G_T.addVertex(G.getLabel(i));
    for (index i = 0; i < n; ++i)
     for (index j = 0; j < n; ++j)
      if (G.isEdge(i, j))
       G_T.addEdge(j, i, G.getEdgeWeight(i, j));

    __dijkstra(G_T, v, dist, pass);

    // *****结果展示*****
    for (index u = 0; u < n; ++u) {
     if (v == u)
      continue;

     out_dest << G.getLabel(u) << "->" << label_to << ":\t";
     if (dist.at(u) >= infinity) {
      out_dest << "inf" << endl;
      continue;
     }
     else
      out_dest << dist.at(u) << "\tPath: ";

     index i = u;
     while (i != v) {
      out_dest << G.getLabel(i) << "->";
      i = pass.at(i);
     }
     out_dest << label_to << endl;
    }
   }
```

shortest_pair.cpp

```cpp
#include "ShortestPathAlgorithm.h"

#include <vector>

void MyGraph::shortest_pair(const AdjacencyMatrixGraph& G, label label_from, label label_to, ::std::ostream& out_dest) {
 using namespace std;

 if (!G.isVertex(label_from) || !G.isVertex(label_to) || label_from == label_to) {
  cout << "Invalid label" << endl;
  return;
 }

 size n = G.countVertex(); // 图的顶点数
 index u = G.getIndex(label_from), v = G.getIndex(label_to);

 vector<weight> dist_u_to(n), dist_v_to(n);
 vector<index> pass_u_to(n), pass_v_to(n);

 // 使用两次 dijkstra 算法，求出 u 到 v 的最短路径，以及 v 到 u 的最短路径
 __dijkstra(G, u, dist_u_to, pass_u_to);
 __dijkstra(G, v, dist_v_to, pass_v_to);

 // ***** 结果展示 *****

 // 输出 u 到 v 的路径
 __dijkstra_show_path(G, u, v, dist_u_to, pass_u_to, cout);

 // 输出 v 到 u 的路径
 __dijkstra_show_path(G, v, u, dist_v_to, pass_v_to, cout);
}
```