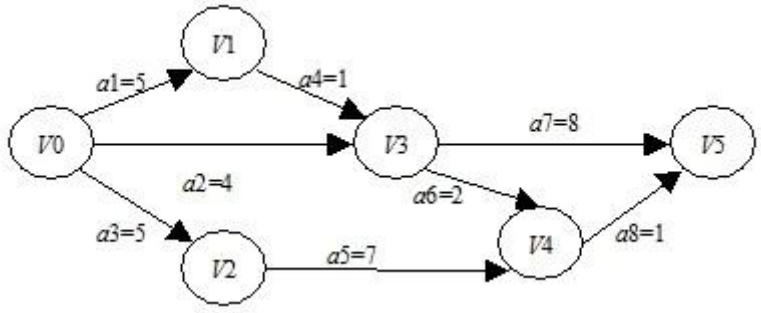
1、题目要求：

在一个有向无环图（DAG）中，有节点 Vertices，连接两个节点的叫做边 Edges，每条边都有权重 Weight，指定一个起点，一个终点和 X 个中间点，用 C++编写程序，找出经过所有这些指定点的权重之和的前 TopN 条路径。需要自己设计图的数据结构，构造相应单元测试用例（可以用 Google Test），用例要覆盖 X 为 1 个或者多个，N 为 1 条或者多条，并能运行通过测试用例。



2、主要算法流程：

（1）构造有向无环图的数据结构，通过邻接矩阵存储边与边之间的权重信息，根据题目要求输入起始点、终点、以及 x 个中间节点

（2）采用深度优先遍历，找到所有的从起点到终点的路径

（3）筛选经过了 x 个中间节点的路径，并按照 weight 从大到小顺序排序

3、数据结构构造：

```
// 定义 DAG 图
class Graph{
public:
   int _vertex_num;                              // 顶点数量
   int _adj_matrix[MAX_SIZE][MAX_SIZE]; // 定义邻接矩阵
   int _begin;                                   // 起点
   int _end;                                     // 终点
   vector<int> _middle_vertices;            // 中间点
   vector<int> _visited;                        // 是否已访问
   void init_adj_matrix();                   // 初始化邻接矩阵
   void construct_path();                     // 构造路径的主函数。输入参数：起点、终点、x 个中间点（必须要经过至少 1 个中间点）
   void print_adj_matrix();                  // 打印邻接矩阵
   void print_result();                      // 打印结果
   void dfs_compute(int tmp_begin, vector<int> tmp_list, int &tmp_weight);   // 计算具体路径
};
```

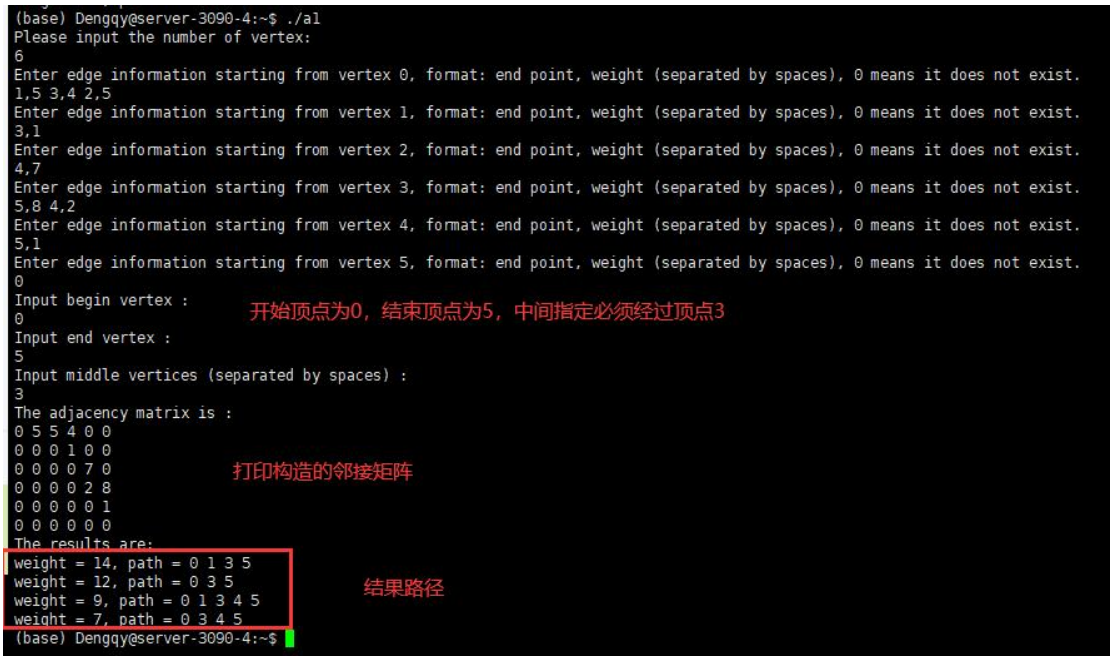
4、DFS 算法构造：

```
oid Graph::dfs_compute(int tmp_begin, vector<int> tmp_list, int &tmp_weight){
   if(tmp_begin == _end){
      int count = 0;
      // 判断是否包含 x 个中间节点
      for(int i = 0; i < _middle_vertices.size(); i++){
```

```
            for(int j = 0; j < tmp_list.size(); j++){
                if(_middle_vertices[i] == tmp_list[j]){
                    count++;
                    break;
                }
            }
        }
        if(count == _middle_vertices.size()){
            path_list.push_back(tmp_list);
            weight_list.push_back(tmp_weight);
        }
        return;
    }
    for(int i = 0; i < _vertex_num; i++){
        if(_adj_matrix[tmp_begin][i] > 0 && _visited[i] == 0){
            tmp_list.push_back(i);
            tmp_weight += _adj_matrix[tmp_begin][i];
            _visited[i] = 1;
            dfs_compute(i, tmp_list, tmp_weight);
            _visited[i] = 0;
            tmp_weight -= _adj_matrix[tmp_begin][i];
            tmp_list.pop_back();
        }
    }
}
```

5、运行结果：

g++ assignment1.cpp -o a1 ./a1



6、附注，使用 cmake 构造工程

（1）首先编写/home/Dengqy/qinyi/assignment1/CMakeLists.txt 文件

```
cmake_minimum_required(VERSION 3.0.0)
project(assignment1 VERSION 0.1.0)

include(CTest)
enable_testing()
include_directories(${PROJECT_SOURCE_DIR}/include)

add_library(graph ${PROJECT_SOURCE_DIR}/src/graph.cpp)
add_executable(main ${PROJECT_SOURCE_DIR}/test/main.cpp)
target_link_libraries(main graph)

set(CPACK_PROJECT_NAME ${PROJECT_NAME})
set(CPACK_PROJECT_VERSION ${PROJECT_VERSION})
include(CPack)
```

（2）然后创建 build 目录，mkdir build，进入 build 目录，输入 cmake ../.，通过 cmake 将 CMakeLists.txt 文件转化为 Make 所需要的 Makefile 文件，然后通过 make 命令编译源码即可生成可执行程序。

```
-- Build files have been written to: /home/Dengqy/qinyi/assignment1/build
(base) Dengqy@server-3090-4:~/qinyi/assignment1/build$ make
Scanning dependencies of target graph
[ 25%] Building CXX object CMakeFiles/graph.dir/src/graph.cpp.o
[ 50%] Linking CXX static library libgraph.a
[ 50%] Built target graph
Scanning dependencies of target main
[ 75%] Building CXX object CMakeFiles/main.dir/test/main.cpp.o
[100%] Linking CXX executable main
[100%] Built target main
(base) Dengqy@server-3090-4:~/qinyi/assignment1/build$ ls
CMakeCache.txt  cmake_install.cmake    CPackConfig.cmake        CTestTestfile.cmake    libgraph.a  Makefile
CMakeFiles      compile_commands.json  CPackSourceConfig.cmake  DartConfiguration.tcl  main        Testing
```

```
(base) Dengqy@server-3090-4:~/qinyi/assignment1$ cmake .
-- The C compiler identification is GNU 9.4.0
-- The CXX compiler identification is GNU 9.4.0
-- Check for working C compiler: /usr/bin/cc
-- Check for working C compiler: /usr/bin/cc -- works
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Detecting C compile features
-- Detecting C compile features - done
-- Check for working CXX compiler: /usr/bin/c++
-- Check for working CXX compiler: /usr/bin/c++ -- works
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Configuring done
-- Generating done
-- Build files have been written to: /home/Dengqy/qinyi/assignment1
```

（2）运行 main 可执行程序，结果如下：

```
(base) Dengqy@server-3090-4:~/qinyi/assignment1/build$ ./main
Please input the number of vertex:
6
Enter edge information starting from vertex 0, format: end point, weight (separated by spaces), 0 means it does not exist.
1,5 3,4 2,5
Enter edge information starting from vertex 1, format: end point, weight (separated by spaces), 0 means it does not exist.
3,1
Enter edge information starting from vertex 2, format: end point, weight (separated by spaces), 0 means it does not exist.
4,7
Enter edge information starting from vertex 3, format: end point, weight (separated by spaces), 0 means it does not exist.
5,8 4,2
Enter edge information starting from vertex 4, format: end point, weight (separated by spaces), 0 means it does not exist.
5,1
Enter edge information starting from vertex 5, format: end point, weight (separated by spaces), 0 means it does not exist.
0
Input begin vertex :
0
Input end vertex :
5
Input middle vertices (separated by spaces) :
3 4
The adjacency matrix is :
0 5 5 4 0 0
0 0 0 1 0 0
0 0 0 0 7 0
0 0 0 0 2 8
0 0 0 0 0 1
0 0 0 0 0 0
The results are:
weight = 9, path = 0 1 3 4 5
weight = 7, path = 0 3 4 5
```