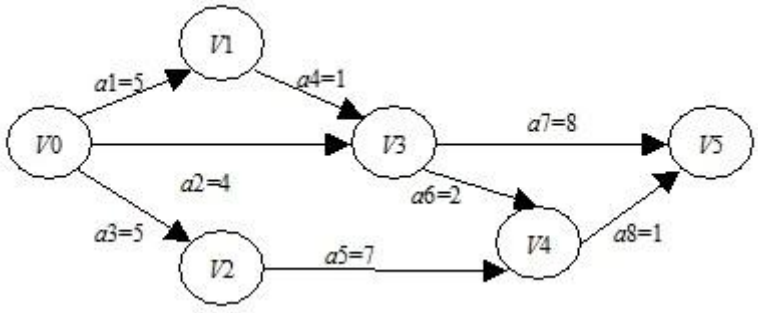
1 题目要求

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，考虑多种情况：

- 起始点与终点之间不可达
- 起点和终点可达，但无法经过所有中间节点
- 起点和终点可达，且可以经过所有中间节点

3 数据结构构造

数据结构主要包括：定义路径，包括起点、终点、中间节点列表，定义了有向无环图，包括顶点数量和邻接矩阵。

```
// Define the path
struct Path
{
   int _begin;                                  // Begin vertex
   int _end;                                    // End vertex
   std::vector<int> _middle_vertices;   // List of middle_vertices
   Path(int b, int e) : _begin(b), _end(e) {}
};

// Define a DAG graph
class Graph
{
  public:
```

```
    Graph(int v) : _vertex_num(v){};
    Graph() = default;
    ~Graph() = default;
    void get_adj_matrix();    // Get and print adjacency matrix
    void set_adj_matrix();    // Initilaize adjacency matrix
    inline int get_vertex_num() { return _vertex_num; }
    Path constructPath();    // Construct path
    void dfsCompute(
            int begin, std::vector<int> vertex_list, int& weight, Path& path, std::vector<int> _visited,
            std::vector<std::pair<std::vector<int>, int>> path_list);    // Calculate specific path
    void printResult(std::vector<std::pair<std::vector<int>, int>> path_list,
                                    int begin);    // Print path result

  private:
    int _vertex_num;                                    // Number of vertices
    int _adj_matrix[MAX_SIZE][MAX_SIZE];    // Adjacency Matrix
};
```

4 核心算法构造

采用深度优先遍历算法进行搜索。

```
/**
  * @brief Calculate specific path
  * @param begin temporary begin vertex
  * @param vertex_list temporary vertices list
  * @param weight current path weight list
  * @param path current path list
  * @param _visited current visited list
  * @param path_list final result list
  */
void Graph::dfsCompute(int begin, std::vector<int> vertex_list, int& weight, Path& path,
                                    std::vector<int> visited,
                                    std::vector<std::pair<std::vector<int>, int>> path_list)
{
    if (begin == path._end) {
        int count = 0;
        // Determine whether it contains x intermediate vertices
        for (int i = 0; i < path._middle_vertices.size(); i++) {
            for (int j = 0; j < vertex_list.size(); j++) {
                if (path._middle_vertices[i] == vertex_list[j]) {
                    count++;
                    break;
                }
            }
        }
```

```
        if (count == path._middle_vertices.size()) {
            path_list.push_back(std::make_pair(vertex_list, weight));
        }
        return;
    }
    for (int i = 0; i < _vertex_num; i++) {
        if (_adj_matrix[begin][i] > 0 && visited[i] == 0) {
            vertex_list.push_back(i);
            weight += _adj_matrix[begin][i];
            visited[i] = 1;
            dfsCompute(i, vertex_list, weight, path, visited, path_list);
            visited[i] = 0;
            weight -= _adj_matrix[begin][i];
            vertex_list.pop_back();
        }
    }
}
```

5 编译与运行

5.1 通过 GNU 编译

```
g++ test/main.cpp src/graph.cpp -I include/ -o a1
./a1
```

```
(base) Dengqy@server-3090-4:~$ ./a1
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
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 = 14, path = 0 1 3 5
weight = 12, path = 0 3 5
weight = 9, path = 0 1 3 4 5
weight = 7, path = 0 3 4 5
(base) Dengqy@server-3090-4:~$
```

开始顶点为0，结束顶点为5，中间指定必须经过顶点3

打印构造的邻接矩阵

结果路径

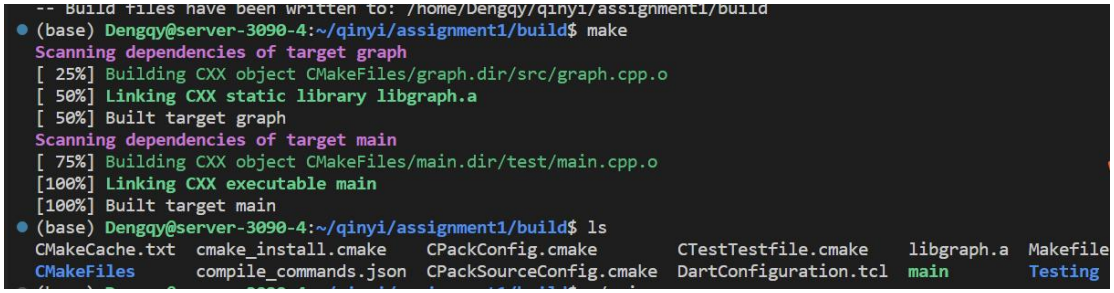
5.2 通过 camke 编译

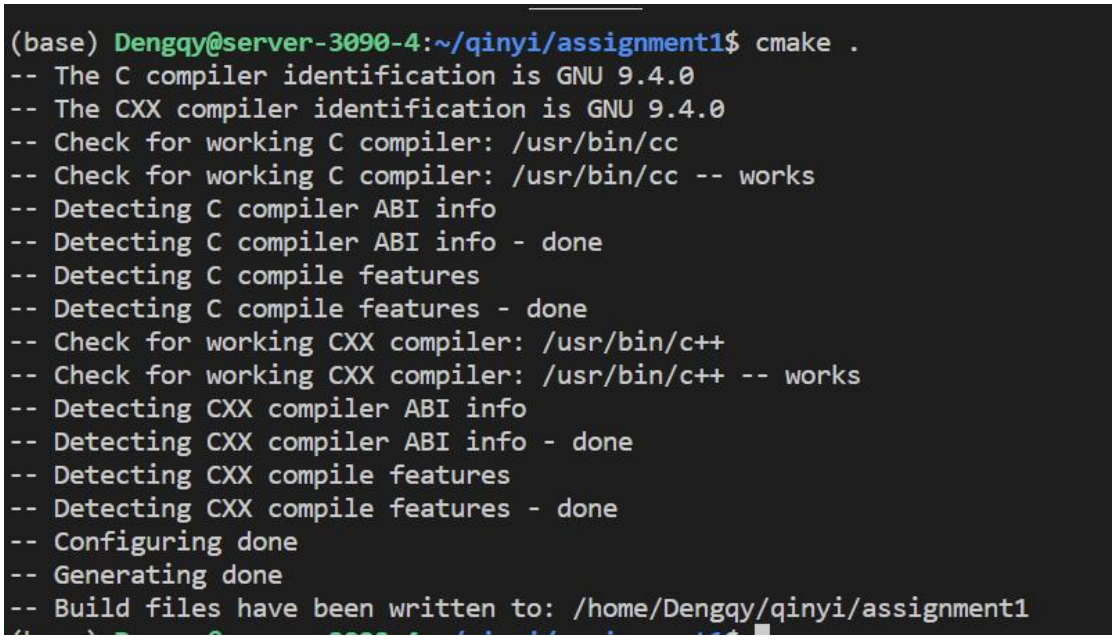
（1）首先编写/home/Dengqy/qinyi/assignment1/CMakeLists.txt 文件

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

```
set(EXECUTABLE_OUTPUT_PATH ${PROJECT_SOURCE_DIR}/bin)

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)

add_executable(test ${PROJECT_SOURCE_DIR}/test/test.cpp)
target_include_directories(test PRIVATE ${PROJECT_SOURCE_DIR}/third_party)
link_directories(${PROJECT_SOURCE_DIR}/third_party)
target_link_libraries(test libgtest.a libgtest_main.a graph pthread)

set(CMAKE_EXPORT_COMPILE_COMMANDS ON)
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。





（3）运行 main 可执行程序

5.3 GoogleTest 编写测试用例运行

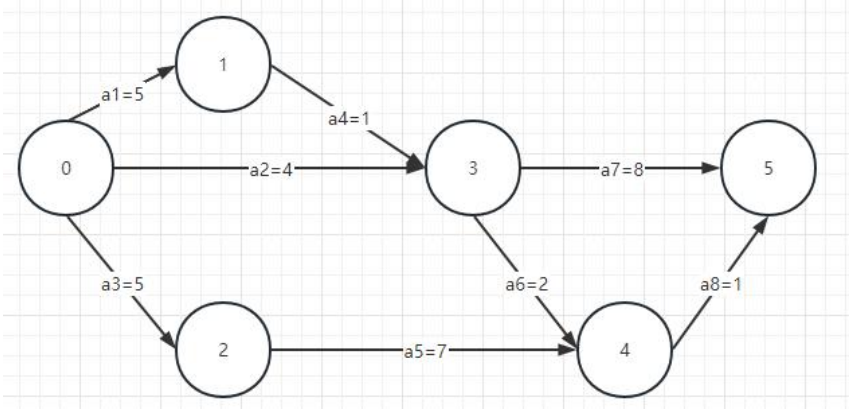
```
6
1,5 3,4 2,5
3,1
4,7
5,8 4,2
5,1
0
0
5
3 4
```

- 用例 1：起点、终点可达，且经过中间节点的唯一路径
- 用例:2：起点、终点可达，且经过中间节点存在多条路径
- 用例 3：起点、终点可达，但无法满足经过所有中间节点
- 用例 4：起点、终点不可达

```
(base) Dengqy@server-3090-6:~/projects/assignment1/bin$ ./test
Running main() from /home/Dengqy/projects/googletest/googletest/src/gtest_main.cc
[==========] Running 4 tests from 1 test suite.
[----------] Global test environment set-up.
[----------] 4 tests from PathTest
[ RUN      ] PathTest.JudgePathCondition1
0
[       OK ] PathTest.JudgePathCondition1 (0 ms)
[ RUN      ] PathTest.JudgePathCondition2
0
[       OK ] PathTest.JudgePathCondition2 (0 ms)
[ RUN      ] PathTest.JudgePathCondition3
0
[       OK ] PathTest.JudgePathCondition3 (0 ms)
[ RUN      ] PathTest.JudgePathCondition4
4
[       OK ] PathTest.JudgePathCondition4 (0 ms)
[----------] 4 tests from PathTest (0 ms total)

[----------] Global test environment tear-down
[==========] 4 tests from 1 test suite ran. (0 ms total)
[  PASSED  ] 4 tests.
```

6 实验结果

（1）原始 DAG 图



（2）选择 0 作为起点，5 作为终点，中间节点为 3,4（其他 condition 可自行测试）

```
(base) Dengqy@server-3090-6:~/projects/assignment1/bin$ ./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 :    起点
0
Input end vertex :  终点
5
Input middle vertices (separated by spaces) : 中间节点
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
```

打印邻接矩阵

结果路径

结果路径有两条，分别为：

weight = 9, path = 0 1 3 4 5

weight = 7, path = 0 3 4 5