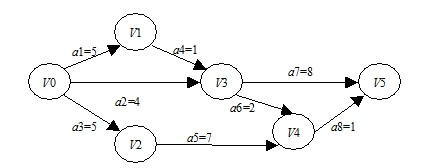
# 1题目要求

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 |



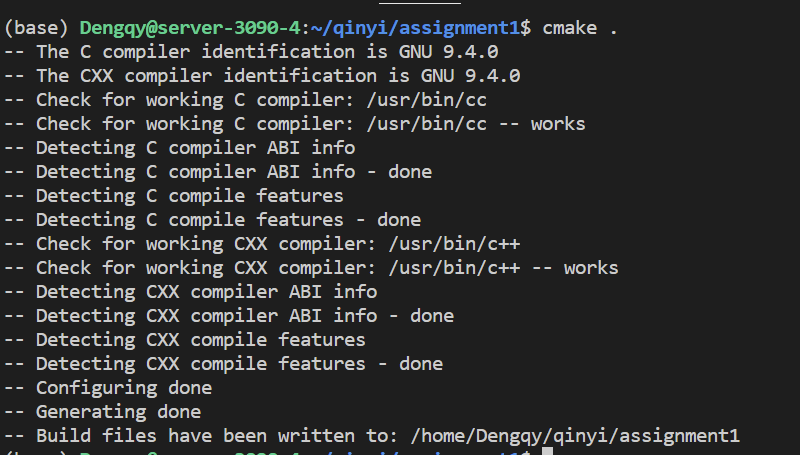
## 5.2 通过camke编译

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。



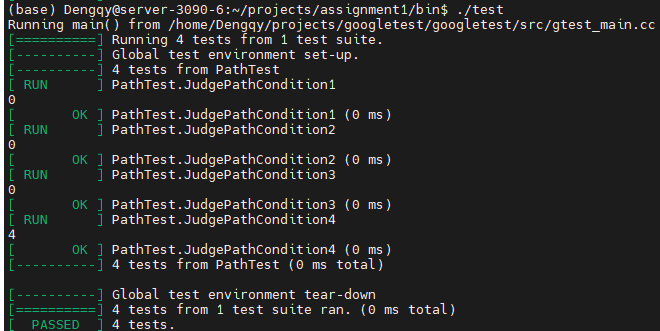


（3）运行main可执行程序

## 5.3 GoogleTest编写测试用例运行

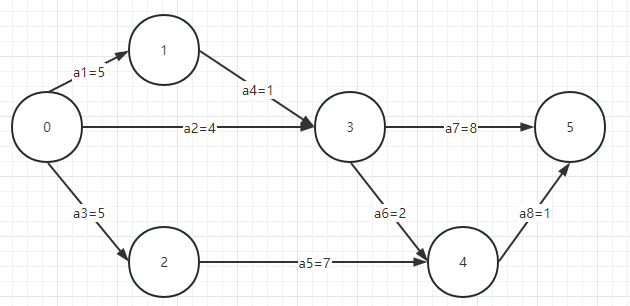
|  |
| --- |
| 6  1,5 3,4 2,5  3,1  4,7  5,8 4,2  5,1  0  0  5  3 4 |

* 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

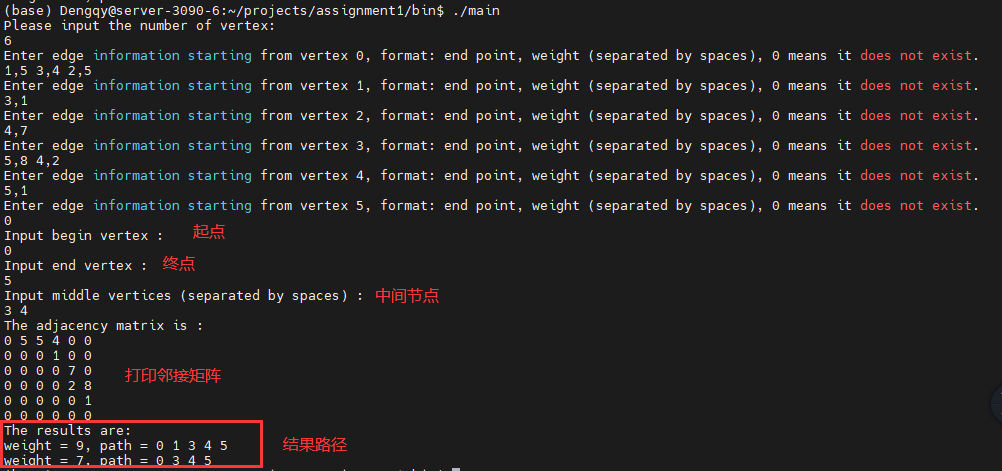


# 6 实验结果

（1）原始DAG图



1. 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）



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

