**数据结构与算法 project2**

**1. 数据结构选择**

**图的信息、邻接矩阵和路径存储：**

* 使用 整型变量numVer和numEdge表示图的点的个数和边的条数。
* 使用二维数组 adjMatrix 表示图的连接关系和权重。
* 使用 vector<Path> 存储多条路径结果，每条路径由 Path 结构体表示，其中包含路径的顶点序列和总长度。

**顶点管理和限制条件：**

* string \* nameVer：存储顶点名称。
* bool \* blacklist：记录顶点的黑名单状态。
* int maxTrans：记录中继点的最大数量限制。

**2. 主要功能实现和解题思路**

1. **按照题意，应当设置适当的函数和主函数，满足一系列功能，包括：**
2. 构造一个图，程序能够接受输入的点的个数和边的条数，以及各顶点名、边的起点和终点及边的权重。
3. 当输入字串命令为“ban”时，将指定的点加入中继点黑名单。
4. 当输入字串命令为“unban”时，将指定的点从中继点黑名单中移除。
5. 设置最大中继点数目限制maxTrans，下次找的任意路径都要求中继点数目小于等于给定的整数X个；若Y是负值，则默认为没有限制中继点的数目。
6. 给出符合当前要求的前K条最短路径，K是输入的一个整数。
7. 当输入字串命令为“quit”时，程序结束。
8. **对应i中功能，以下为具体函数实现。**
9. main函数

读取输入的点数和边数，依次调用 addVer 和 addEdge 构造图的结构。使用循环读取用户输入的命令，根据命令调用相应的函数实现功能，直到用户输入 "quit" 退出程序。

1. 处理黑名单

banned(const string& x): 将指定顶点加入黑名单。

unbanned(const string& x): 将指定顶点从黑名单移除。

isbanned(const string& x): 检查顶点是否在黑名单中。

1. 设置最大中继点数目限制

setmaxTrans(int max): 设置路径中的最大中继点数目限制。

getmaxTrans(): 获取当前设置的最大中继点数目限制。

1. **寻找前K条最短路径**

findKShortestPaths（string stVer, string endVer, int K）：找出从起始顶点stVer到终点endVer的前K条最短路径，并将这些路径存储在 vector<Path> 类型的容器中返回。使用Dijkstra算法的变形。

1. **初始化过程**：
   * 首先，清空存储路径的 paths 和候选路径的 candidates。
   * 获取起始顶点 stVer 和目标顶点 endVer 在顶点数组中的索引，这里使用 getVerIndex 函数来获取索引值。
2. **构建初始候选路径**：
   * 遍历起始顶点 stVer 相连的所有顶点，并且这些顶点不能在黑名单中。对于每个相邻的顶点，构造一个新的路径对象 tmpPath，将其添加到 candidates 中。
   * 每个 tmpPath 包含起始顶点 stVer 和相邻顶点的名称，以及它们之间的路径长度。
3. **路径选择过程**：
   * 使用一个循环来选择最短的路径，直到找到 K 条路径或者候选路径集为空。
   * 在 candidates 中寻找路径长度最短的路径。如果有多条路径长度相同，则选择第一条遍历到的路径。
   * 将选择的路径从 candidates 中移除，并检查路径的最后一个顶点是否为目标顶点 endVer。如果是，则将该路径添加到 paths 中，并增加已找到路径的计数器 pathCount。
4. **路径扩展过程**：
   * 如果选择的路径尚未到达目标顶点 endVer，则从路径的最后一个顶点出发，寻找它相邻的顶点。
   * 对于每个相邻顶点，检查是否已经在当前路径中，避免形成环路。如果不是，则创建一个新的路径对象，将其添加到 candidates 中。
   * 新路径的长度为原路径长度加上到相邻顶点的边的权重。
   * 添加路径时，还需检查路径中的中继点数量是否超过了设定的 maxTrans，确保路径中中继点的数量不超过限制。
5. **返回结果**：
   * 当找到 K 条路径或者无法找到更多路径时，函数结束循环并返回存储了前 K 条最短路径的 paths 容器。

这个函数的核心在于使用贪心策略，每次选择当前看来最短的路径，并在路径扩展时通过限制条件来控制路径的生成和选择。

**3. 实现难点**

1. **用户交互和命令解析**：本代码实现了通过命令行接口进行用户交互，包括添加顶点、边，设置黑名单、最大中继点数限制和查找路径等功能。难点在于需要确保命令的解析和功能的正确执行。
2. **动态路径管理**：
3. **路径的动态生成与管理**：在每一步选择当前看似最短的路径时，需要动态地生成可能的下一步路径，并进行管理。这包括将新路径添加到候选路径集合中，以及在路径选择后将路径从候选路径集合中移除。
4. **避免形成环路**：在路径扩展过程中，必须确保不会形成环路。这意味着每个路径都必须记录已经访问过的顶点，以避免重复访问同一顶点而导致无限循环或无效路径的生成。
5. **处理已访问过的顶点**：当路径扩展到新的顶点时，需要检查该顶点是否已经在当前路径中。如果是，则不应将其添加到候选路径中，以避免路径中出现重复顶点。
6. **代码必须能够正确处理并返回少于 K 条路径的结果。**这要求算法在结束条件判断时能够正确地处理这种情况，以避免陷入无限循环或者返回错误的结果。