import sys

def dijkstra(graph, start, end):

# 初始化距离

distances = {node: sys.maxsize for node in graph}

distances[start] = 0

# 初始化前驱节点

predecessors = {node: None for node in graph}

# 标记节点是否已访问

visited = []

while True:

# 选择距离最小的节点

min\_distance = sys.maxsize

min\_node = None

for node in graph:

if node not in visited and distances[node] < min\_distance:

min\_distance = distances[node]

min\_node = node

if min\_node is None:

break

# 更新距离和前驱节点

for neighbor, distance in graph[min\_node].items():

if distances[min\_node] + distance < distances[neighbor]:

distances[neighbor] = distances[min\_node] + distance

predecessors[neighbor] = min\_node

# 标记节点为已访问

visited.append(min\_node)

# 构建最短路径

path = []

node = end

while node is not None:

path.insert(0, node)

node = predecessors[node]

return path, distances[end]

# 创建图

graph = {

'A': {'B': 5, 'C': 2},

'B': {'A': 5, 'C': 1, 'D': 3},

'C': {'A': 2, 'B': 1, 'D': 6},

'D': {'B': 3, 'C': 6}

}

# 求解最短路径

start\_node = 'A'

end\_node = 'D'

shortest\_path, shortest\_distance = dijkstra(graph, start\_node, end\_node)

print(f"The shortest path from {start\_node} to {end\_node} is: {shortest\_path}")

print(f"The shortest distance is: {shortest\_distance}")

图论算法

图论算法是一类用于解决关于图的各种问题的算法。在图论中，图被定义为一组节点和连接它们的边的集合。图论算法通常被用于寻找一条路径、测量节点之间的关联以及识别图中的相关模式。

一些图论算法包括：

1. Dijkstra算法：用于计算给定图中的单源最短路径，找到某一节点到其他所有节点的最短距离。Dijkstra算法是一种贪心算法，每次找到当前距离最短的节点（即距离源点最近的节点），然后以该节点为基础进行迭代更新，最终找到所有节点的最短路径。

2. Bellman-Ford算法: 用于计算有向或无向带权图中每个节点到其他所有节点的最短路径，支持权重为负数的情况。Bellman-Ford算法通过松弛边的方式来求解，其基本思路是进行|V|-1次循环迭代更新，从而逐步遍历出所有边。

3. Kruskal算法：用于解决最小生成树问题，生成一棵生成树以使得所有节点都连接起来，并且总权值最小。Kruskal算法是一种贪心算法，先将所有的边按权值从小到大排序，依次加入MST中，如果该边不会使得生成的树形成环，则加入到MST中，否则过滤掉。

4. Prim算法：用于解决最小生成树问题，生成一棵生成树以使得所有节点都连接起来，并且总权值最小。Prim算法同样也是一种贪心算法，以某一起始节点为基础，依次找到与之相连通的那些边中权重最小的那一条，并将其加入MST中。

5. Floyd-Warshall算法：用于计算所有节点之间的最短路径，用于求解所有节点的最短路径距离。Floyd-Warshall算法基于动态规划思想，将所有两两节点之间的最短路径的求解分解为若干个子问题，通过迭代更新求得最短路径。For 中间 for 起点 for 终点

6、最大流算法：最大流算法是一种经典的网络流算法。它基于残留网络的概念，通过不断增广来求解最大流问题。具体来说，最大流算法使用一个二维数组cap记录每条边的容量，同时构建残留网络。然后，通过不断搜索增广路径，并更新残留网络的容量，来求解最大流问题。具体来说，每次在残留网络中寻找一条从源点到汇点的增广路径，并计算该路径上的最小剩余流量。然后，将该最小剩余流量添加到该条路径上的所有边中，并更新残留网络的容量。重复执行此过程，直到无法找到增广路径为止。最终，所有边上的流量之和即为最大流。最大流算法可用于解决带容量限制的网络流问题，如物流调度、任务分配等应用场景。

def Power(b, k):

flag=1

re=1

tmp=b

if k==0: #等于0的情况

return 1

if k<0:

flag=0

k=abs(k)

while k>0: #大于0的情况

if k&1==1:

re=re\*tmp

k=k>>1

tmp=tmp\*tmp

return re if flag else 1/re

def fastExpMod(b, e, m):

result = 1

while e != 0:

if (e&1) == 1:

# ei = 1, then mul

result = (result \* b) % m

e >>= 1

b = (b\*b) % m

return result