|  |
| --- |
| from collections import defaultdict |
|  |  |
|  | class Graph: |
|  |  |
|  | def \_\_init\_\_(self): |
|  | self.graph=defaultdict(list) |
|  |  |
|  | def addEdge(self,vertex,neighbours): |
|  | self.graph[vertex].extend(neighbours) |
|  |  |
|  | def bestFirstSearch(self,goal,distance): |
|  | queue=[] |
|  | state='A' |
|  | path=['A'] |
|  | while True: |
|  | if state==goal: |
|  | print("SUCCESS: Path is",','.join(path)) |
|  | return True |
|  |  |
|  | queue=self.graph[state]+queue |
|  | queue.sort(key=lambda x:distance[ord(x)-65]) |
|  |  |
|  | if queue==[]: |
|  | return False |
|  | state=queue.pop(0) |
|  | path.append(state) |
|  |  |
|  | def main(): |
|  | numOfNodes=int(input("Enter the number of nodes in graph:")) |
|  | g=Graph() |
|  |  |
|  | distance=[] |
|  | goal=input("Enter the goal node:") |
|  |  |
|  | for i in range(numOfNodes): |
|  | successors=input("Enter the successors of node %s:"%chr(65+i)).split() |
|  | distance.append(int(input("Enter the straight line distance from node %s to the goal node %s:"%(chr(65+i),goal)))) |
|  | g.addEdge(chr(65+i),successors) |
|  |  |
|  | if not g.bestFirstSearch(goal,distance): |
|  | print("FAILURE!!!") |
|  |  |
|  | if \_\_name\_\_=="\_\_main\_\_": |
|  | main() |