19110482_main

November 20, 2021

#Nhập môn Trí tuệ Nhân tạo

- Bài thực hành tuần 1
- Sinh viên: Huỳnh Thị Bảo Trân
- MSSV: 19110482

Đề bài: Cho đồ thị hình vẽ, tìm đường đi ngắn nhất từ trường Đại học Khoa học Tự nhiên (V_1) đến sân bay Tân Sơn Nhất (V_18) với các thuật toán: BFS, DFS, UCS.

1. Thư viện được sử dụng

[ ]:
```
from queue import Queue, PriorityQueue
import numpy as np
from collections import defaultdict
```

2. Đọc dữ liệu

* Thực hiện đọc file bằng hàm open('địa chỉ file', mode="r").
* Đọc dữ liệu theo từng dòng (không lấy các ký tự xuống dòng).
* Ghép các dòng thành list và đưa chúng thành ma trận kề.

[ ]:
```
# load du lieu cua BFS va DFS

data1 = open("data_bfs_dfs.txt", "r")
vertice = int(data1.readline())
start, end = [int(num) for num in data1.readline().split()]
graph = [[int(num) for num in line.split()] for line in data1]
print("vertice", vertice)
print("start", start)
print("end", end)
graph
```

```
vertice 18
start 0
end 17
```

[ ]:
```
[[0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
```

```
 [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0],
 [0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0],
 [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0],
 [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1],
 [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]]
```

[ ]:
```
# load du lieu cua UCS

data2 = open("data_ucs.txt", "r")
vertice = int(data2.readline())
start, end = [int(num) for num in data2.readline().split()]
cost = [[int(num) for num in line.split()] for line in data2]
print("vertice", vertice)
print("start", start)
print("end", end)
cost
```

```
vertice 18
start 0
end 17
```

[ ]:
```
[[0, 50, 350, 300, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 600, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 100, 0, 900, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 1300, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1400, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 700, 0, 0, 0, 0, 0, 0],
 [0, 0, 800, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 0, 0, 790, 300, 0, 0, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1200, 0, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 0, 0, 0, 800, 0, 0, 0, 0, 400, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 950, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 600, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1300, 0, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1300, 0, 0],
 [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 770, 0],
 [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1200],
 [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]]
```

3. Xây dựng hàm Breadth First Search (BFS)

Ý tưởng:
* Tại mỗi bước chọn trạng thái để phát triển là trạng thái được sinh ra trước các trạng thái chờ phát triển khác.
* Danh sách L được sử lý như hàng đợi (queue).

Cách thực hiện:
* Cho *frontier*, *list_father*, *explored* strack rỗng, truyền vào *frontier* dữ liệu ở vị trí *start=1*.
* Giả sử đúng, ta kiểm *frontier* nếu không có dữ kiệu thì ta xuất ra màn hình là "no way" và dừng lại.
* Sau đó loại vị trí đầu.
* Ta xét *node*, nếu *node* ở vị trí cuối thì thêm vị trí cuối vào *result[]*, gán *end_index=0* sau đó chạy vòng for.
* Nếu vị trí *end-1* tới *sons*, gán *end_index=i* sau đó dừng lại.
* Gán *find=father*, thêm *find+1* và *result*.
* Thêm *father+1* vào *result* [], gán *find=father*.
* Sao đó chạy ngược lại *result* và gán lại *result*, gán *result =[str(num) for num in result]*.
* Tạo tem[]. For chạy I đến chiều dài của node.
* Thêm *i* vào *frontier* và thêm *temp* và *node* vào *list_father*.

[ ]:
```
# thuat toan BFS

def BFS(graph, start, end):
    frontier = [] #queue su dung list queue.put() ~ list.append(); queue.get()␣
 ↪~ list.pop(0)
    list_father = []
    explored= []
    frontier.append(start)

    while True:
        if frontier==[]:
            print("no way")
            return False,

        current_node = frontier.pop()
        explored.append(current_node)

        if current_node == end - 1:
            result = []
            result.append(end)
            end_index = 0

            for i in range(-1, -len(list_father)-1, -1):
                sons, father = list_father[i]
                if end -1 in sons:
                    end_index = i
                    break
```

```
            find = father
            result.append(find + 1)

            for i in range(end_index - 1, -len(list_father)-1, -1):
                sons, father = list_father[i]
                if find in sons:
                    result.append(father + 1)
                    find = father
            result = result[::-1]
            result = [str(num) for num in result]
            return True, '->'.join(result)

        temp = []
        for i in range(len(graph[current_node])):
            if graph[current_node][i] and i not in explored:
                frontier.append(i)
                temp.append(i)

        list_father.append((temp, current_node))
```

4. Xây dựng hàm Depth First Search (DFS)

Ý tưởng:
* Tại mỗi bước trạng thái được chọn để phát triển là trạng thái được sinh ra sau cùng trong số các trạng thái chờ phát triển.
* Danh sách L được sử lý như ngăn xếp (stack).

Cách thực hiện:
* Khởi tạo hàm DFS với ma trận kề *(graph)*, vị trí đầu *(start)* và vị trí cuối *(end)*.
* Ta tạo *frontier* [], *explored* [] là các sack rỗng và gán vị trí đầu vào *frontier[]*.
* Giả sử đúng, nếu *frontier* rỗng thì ta sẽ xuất không có đường đi và kết thúc.
* Node là *frontier* đã lấy dữ liệu, xuất vị trí *node+1* ra và truyền giá trị vào *explored*.
* Nếu node ở vị trí cuối ta dừng lại, ngược lại tiếp tục chạy dòng for kiểm tra *explored*.
* Thực hiện lặp lại đến node của *graph* ở vị trí cuối.

[ ]:
```
# thuat toan DFS

def DFS(graph, start, end):
  frontier = [] #stack sử dụng list stack.put() ~ list.append(); stack.get() ~
 ↪list.pop()
  frontier.append(start)
  explored = []

  while True:
    if frontier == []:
      print("NO WAY")
      return
    current_node = frontier.pop()
```

```
    print("V" + str(current_node + 1))
    explored.append(current_node)

    if current_node == end:
      return

    for i in range(len(graph[current_node])):
      if graph[current_node][i] == 1 and i not in explored:
        frontier.append(i)
        explored.append(i)
```

5. Xây dựng hàm UCS

Ý tưởng: * Hàng đợi ưu tiên PQ là cấu trúc dữ liệu lưu trữ các phần tử cùng với độ ưu tiên của nó.
* Khi lấy phần tử ra khỏi hàng đợi sẽ căn cứ vào độ ưu tiên nhỏ nhất.

Cách thực hiện: * Khởi tạo chi phí tối thiểu tối đa, trạng thái mục tiêu kể và hàng đợi ưu tiên.
* Đặt vectơ câu trả lời thành giá trị lớn nhất.
* Chèn chỉ mục bắt đầu và bản đồ để lưu trữ.
* Lấy phần tử hàng đầu, giá trị ban đầu và kiểm tra xem phần tử có phải thuộc danh sách mục tiêu.
* Đặt các giả định kiểm tra các nút có truy cập hay không, có tiệm cận với nút hiện tại không...
* Giá trị được nhân với -1 để ưu tiên ít nhất là ở trên cùng và đánh dấu.

[ ]:
```
# thuat toan UCS

def UCS(end, start):
        global graph, cost
        answer = []
        queue = []

        for i in range(len(end)):
                answer.append(10**8)

        queue.append([0, start])
        visited = {}
        count = 0

        while (len(queue) > 0):
                queue = sorted(queue)
                p = queue[-1]
                del queue[-1]
                p[0] *= -1

                if (p[1] in end):
                        index = end.index(p[1])
                        if (answer[index] == 10**8):
```

```
                                count += 1

                        if (answer[index] > p[0]):
                                answer[index] = p[0]
                        del queue[-1]

                        queue = sorted(queue)
                        if (count == len(end)):
                                return answer

                if (p[1] not in visited):
                        for i in range(len(graph[p[1]])):
                                queue.append( [(p[0] + cost[(p[1],␣
↪graph[p[1]][i])])* -1, graph[p[1]][i]])

                visited[p[1]] = 1

        return answer
```

6. Xuất kết quả

[ ]:
```
print("Algorithm BFS: ")
BFS(graph, start, end+1)
```

```
Algorithm BFS:
```

[ ]:
```
(True, '1->4->5->14->11->15->16->17->18')
```

[ ]:
```
print("Algorithm DFS: ")
DFS(graph, start, end)
```

```
Algorithm DFS:
V1
V4
V5
V14
V11
V15
V16
V17
V18
```

[ ]:
```
print("Algorithm UCS: ")
UCS(cost, start)
```