人工智能课程实验 1

武自厚 20336014 保密管理

2022 年 2 月 25 日

1 实验目的

熟悉 Python 语言的语法及特性, 并以此为基础实现无向图的最短路径算法 (Dijkstra 算法).

2 算法原理

2.1 策略

本算法基于"贪心策略"实现, 具体表现为: 在一次遍历中只考虑已访问结点和未访问结点之间**最短**的边.

2.2 数据结构

无向图的存储使用邻接表, 结点已访问与否的判断以及最短路径表示使用并查集, "贪心策略"的实现使用优先队列.

2.3 自然语言描述

首先, 对任意结点 v 赋予"距离"属性且其值为无穷大, 即 $\texttt{dist}[v] = \infty$, 随后将起点的"距离"赋值为 0. 再对任意节点 v 赋予"前驱"属性, 并其值赋值为结点自身, 即 $v.\pi = v$

直到所有结点都加入时, 重复以下步骤: 遍历无向图的每一条边 (权为 w), 如果其中一个端点 u 已经访问且另一个端点 v 未访问, 且 $\texttt{dist}[u] + w < \texttt{dist}[v]$ 则将这条边纳入考虑. 再将所有已经考虑的边中取出权最小的, 将其终点的"距离"用起点的"距离"与边权的和取代 (即松弛过程), 并且将终点设为"已访问", 将终点的"前驱"赋值为起点.

算法结束之后, 目标点的"距离"属性即位最短路径长度, 通过迭代访问"前驱"属性可以获得该路径上所有结点的序列.

3 伪代码实现

3.1 主算法

Algorithm 1: Dijkstra 算法

输入: 带权图 $G = \langle V, E \rangle$, 初始结点 a, 终止结点 b

输出: G 中从 a 到 b 的最短路径及其长度 $\langle L, l \rangle$

// 初始化

for $v \in V$ **do**
  $v.\texttt{dist} \coloneqq \infty$
  $v.\pi \coloneqq v$
while $\exists v \in V : \texttt{ancestor}(v) \neq a$ **do**
  $E^* \coloneqq \emptyset$
  for $e \in E$ **do**
    $\langle u, v, w \rangle \coloneqq e$
    if $u.\texttt{dist} + w < v.\texttt{dist}$ **then**
      $E^* \coloneqq E^* \cup e$
  $\langle u_0, v_0, w_0 \rangle \coloneqq \min_{e \in E^*} e$
  $v_0.\texttt{dist} \coloneqq u_0.\texttt{dist} + w_0$
  $v_0.\pi \coloneqq u_0$
$L \coloneqq \texttt{path}(V, b)$
$l \coloneqq b.\texttt{dist}$
return $\langle L, l \rangle$

3.2 并查集相关算法

Algorithm 2: 获得结点的"祖先"`ancestor`()

输入: 记录前驱的结点 x

输出: x 的"祖先"a

while $x.\pi \neq x$ **do**

  $x := x.\pi$

return x

Algorithm 3: 通过前驱获得路径 `path`()

输入: 记录前驱的结点 x

输出: 记录从 x 的"祖先"到 x 的最短路径上的所有结点的列表 L

$L := []$

L.append(x)

while $x.\pi \neq x$ **do**

  $x := x.\pi$

  L.append(x)

L.reverse()

return L

4 代码展示

4.1 并查集模块 dijoint_set.py

```python
from typing import List


class DisjointSet:
    def __init__(self, data: List[str]):
        self.fa = dict()  # "fa" means the precursor of the node
        for datum in data:
            self.fa[datum] = datum

    def find(self, x: str) -> str:  # find the ancestor of the node
        while self.fa[x] != x:
            x = self.fa[x]
        return x
```

```
    def union(self, x: str, y: str) -> None:  # union x and y, that is, make x be y's father
        self.fa[y] = x

    def is_linked(self, x: str, y: str) -> bool:  # judge whether x's ancestor is the same
                                                    as y's
        return self.find(x) == self.find(y)

    def path(self, start: str, end: str) -> List[str]:  # make the shortest path from
                                                    records of precursor of the
        # "end" node
        res = [end]
        while self.fa[end] != end:
            end = self.fa[end]
            res.append(end)

        res.reverse()
        if res[0] != start:  # that means "end" is unconnected with "start"
            return []
        else:
            return res
```

4.2 最短路径算法模块 dijkstra.py

```
import queue
from typing import Tuple

from disjoint_set import *


def dijkstra(vertices: List[str], edges: List[Tuple[str, str, int]], start: str, end: str) -
                                              > Tuple[List[str], int]:
    dist = dict()
    dj = DisjointSet(vertices)  # get a disjiont set from the vertices
    for item in vertices:
        dist[item] = -1

    dist[start] = 0
    finished = False
    while not finished:
        q = queue.PriorityQueue()
        for edge in edges:
            (u, v, weight) = edge  # u and v are two ends of the edge and u is considered
                                                      while v isn't
            if dj.is_linked(start, u) and not dj.is_linked(start, v) and (dist[v] == -1 or
                                                      dist[u] + weight < dist[v]):
                q.put((weight, edge))

        if not q.empty():  # get the shortest edge in considered edges
```

```
            (u, v, weight) = q.get()[1]
            dist[v] = dist[u] + weight
            dj.union(u, v)

        finished = True

        for vertex in vertices:  # test whether all vertices are visited
            if not dj.is_linked(start, vertex):
                finished = False
    return dj.path(start, end), dist[end]
```

4.3 主模块 main.py

```
from dijkstra import dijkstra

if __name__ == '__main__':
    edges = []
    vertices = []
    v_dict = dict()
    file = open("Romania.txt", "r")
    v_size, e_size = file.readline().split()
    v_size = int(v_size)
    e_size = int(e_size)
    print("The size of vertices is {} and one of edges is {}".format(v_size, e_size))

    # input a graph
    for i in range(e_size):
        start, end, weight = file.readline().split()

        # process for capital insensitivity
        start = start.lower()
        end = end.lower()

        if vertices.count(start) == 0:
            vertices.append(start.lower())
            v_dict[start[0]] = start  # for searching by first letter
        if vertices.count(end) == 0:
            vertices.append(end.lower())
            v_dict[end[0]] = end  # for searching by first letter

        weight = int(weight)
        edges.append((start, end, weight))
        edges.append((end, start, weight))  # because all edges are undirected

    file.close()
    # it's time for the user
    while True:
        st = input("Please type your start city(type 'quit' to quit this program): ").lower
()
```

```
        if st == 'quit':
            break
        elif len(st) == 1:
            st = v_dict.get(st)
            if st is None:  # error detect
                print("There is no such city!!!! Type again please.")
                continue
        elif vertices.count(st) == 0:  # error detect
            print("There is no such city!!!! Type again please.")
            continue

        ed = input("Please type your end city(type 'quit' to quit this program): ").lower()
        if ed == 'quit':
            break
        elif len(ed) == 1:
            ed = v_dict.get(ed)
            if ed is None:  # error detect
                print("There is no such city!!!! Type again please.")
                continue
        elif vertices.count(ed) == 0:  # error detect
            print("There is no such city!!!! Type again please.")
            continue

        print("start is \'{}\' and end is \'{}\'".format(st, ed))
        path, length = dijkstra(vertices, edges, st.lower(), ed.lower())
        log = open("log.txt", "a")
        if length != -1:
            print("The shortest path is {} and its length is {}.".format(path, length))  #
                                                          put results
            log.write("{} to {}\npath: {}\nlength: {}\n\n".format(st, ed, path, length))  #
                                                          logging
        else:
            print("These two vertices are not connected.")  # put results
            log.write("{} to {}\nfailed\n\n".format(st, ed))  # logging
        log.close()
```

5 实验结果及分析

进行三次查询。终端截屏如下:

```
/Users/naphtholmizuha/.conda/envs/AI/bin/python /Users/naphtholmizuha/Course/AI/lab1/main.py
The size of vertices is 20 and one of edges is 23
Please type your start city(type 'quit' to quit this program): a
Please type your end city(type 'quit' to quit this program): Bucharest
start is 'arad' and end is 'bucharest'
The shortest path is ['arad', 'timisoara', 'lugoj', 'mehadia', 'dobreta', 'craiova', 'pitesti', 'bucharest'] and its length is 733.
Please type your start city(type 'quit' to quit this program): FAGARAS
Please type your end city(type 'quit' to quit this program): D
start is 'fagaras' and end is 'dobreta'
The shortest path is ['fagaras', 'sibiu', 'rimnicuvilcea', 'pitesti', 'craiova', 'dobreta'] and its length is 534.
Please type your start city(type 'quit' to quit this program): menhadia
There is no such city!!!! Type again please.
Please type your start city(type 'quit' to quit this program): mehadia
Please type your end city(type 'quit' to quit this program): Sibiu
start is 'mehadia' and end is 'sibiu'
The shortest path is ['mehadia', 'dobreta', 'craiova', 'pitesti', 'rimnicuvilcea', 'sibiu'] and its length is 510.
Please type your start city(type 'quit' to quit this program):
```

而 log.txt 中的日志信息如下：

```
arad to bucharest
path: ['arad', 'timisoara', 'lugoj', 'mehadia', 'dobreta', 'craiova', 'pitesti', 'bucharest']
length: 733

fagaras to dobreta
path: ['fagaras', 'sibiu', 'rimnicuvilcea', 'pitesti', 'craiova', 'dobreta']
length: 534

mehadia to sibiu
path: ['mehadia', 'dobreta', 'craiova', 'pitesti', 'rimnicuvilcea', 'sibiu']
length: 510
```

不难看出，程序运行正确。主要体现在以下方面：

1. 输入首字母或城市全名均能正确识别。
2. 实现了大小写无关。
3. 成功得出了最短路径及其长度。
4. 正确生成日志信息。

6 思考题

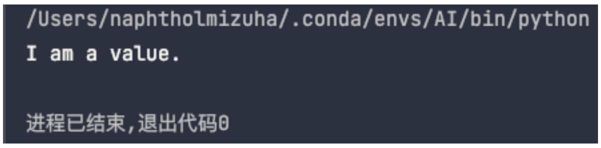
6.1 字典的键

使用 thinking.py 脚本验证:

```
if __name__ == '__main__':
    arr = [1, 1, 4, 5, 1, 4]
    tup = (1, 1, 4, 5, 1, 4)
    tup_ = (1, 1, 4, 5, 1, 4)
    dic = dict()
```

```
    dic[arr] = 1
    dic[tup] = 'I am a value.'
    print(dic[tup_])
```

发现解释器在第 6 行报错，并提示 list 并不是一个可哈希的类型。将第 6 行注释后，再次运行得到如下结果：根据观察可以得出：列表作为字典的键将会报错，而元组将可以正常使



用运行。

6.2 可变/不可变

使用 mutability.py 脚本验证:

```
if __name__ == '__main__':
    i = 114514
    print(id(i))
    i += 1
    print(id(i))

    f = 3.14
    print(id(f))
    f *= 2.23
    print(id(f))

    b = True
    print(id(b))
    b = not b
    print(id(b))

    s = 'hello'
    print(id(s))
    s = s.replace('he', 'Fe')
    print(id(s))

    t = (1, 'what', 2.13)
    print(id(t))
    t = (2, 'which', 0.01)
    print(id(t))

    print('\n')

    li = [1, 1, 4, 5, 1, 4]
    print(id(li))
```

```
li.append(7)
print(id(li))

d = dict([('this', 'fish'), ('that', 'meat')])
print(id(d))
d.clear()
print(id(d))

s = {'group', 'ring', 'field'}
print(id(s))
s.union({'plus', 'ring'})
print(id(s))
```

发现整数、浮点数、字符串、元组和布尔值变量在修改时会改变内存区域，即这些是不可变变量。而列表、集合和字典变量在修改后依然位于同一内存，即为可变变量。