

Guide of Python

内容概要： 数学建模算法

创建时间： 2022/4/7 13:41

更新时间： 2022/4/17 15:27

作者： TwinkelStar

Dijkstra 算法

Dijkstra Algorithm

1、操作系统相关环境

1) 硬件环境：

- 电脑

2) 软件环境：

- Python3.7(向下兼容 Python3)(程序设计语言)
- Numpy1.19.5(兼容大部分版本)(科学计算库)

3) 操作系统(2 选 1)：

- Windows7
- Windows10
- Windows11

2、最短路径问题

最短路径问题是在图论研究中的一个经典算法问题，旨在图论问题（如无向图、有向图、加权无向图、加权有向图等）寻找图中两结点之间的最短路径。

1) 应用场景

最短路径问题通常的形式为：确定起点的最短路径问题，即已知

起始结点，求最短路径的问题。最短路径问题常运用在旅行商问题，旅行商问题（TSP）是一个经典的 NP 难问题，可描述为：给定一组城市的坐标集，一个旅行商从起点城市出发，考虑到自身资源、天气等原因，如何经过各个城市并回到起始点的问题。TSP 问题的最优解就是旅行商所经历的最短路径。

3、Dijkstra 算法

Dijkstra 算法是荷兰科学家狄克斯特拉于 1959 年提出的启发式搜索方法，姑也叫狄克斯特拉算法，其基本思想是，从某一顶点 V_0 出发，搜索从它到其他各顶点的最短路径。把图的顶点集 V 分为两个子集 S 和 T ， S 为已求出最短路径顶点的集合， T 为未求出最短路径顶点的集合，循环遍历集合 T ，将每次找出的最短顶点放入集合 S ，直到集合 T 为空，如图 1 所示为 Dijkstra 算法示意图：

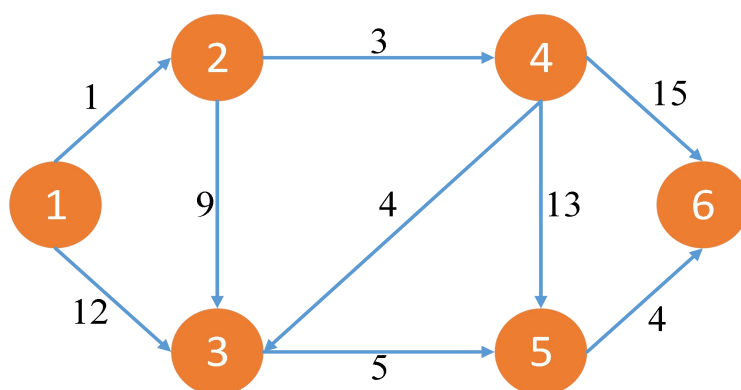


图 1.Dijkstra 算法示意图
fig1.Dijkstra Algorithm Diagram

通俗的来说，就是在编程的过程中，我们利用两个集合来存储我们的结点，一个集合存放的是我们除开出发点之外所有结点，另外一个集合存放的是我们走过的哪些结点。然后我们利用循环遍历的方法（有一点暴力枚举的方法）对我们的两节点之间的最短路径进行筛选判断，最终得到我们一个最短路径的集合。

1) 算法的使用范围及思路：

算法的使用范围：在图论问题中（有向图、无向图和混合图），当权非负时，寻找从一固定顶点到其余各点之间的最短路径。

算法的思路在于：采用标记作业法，将每次迭代的结果做存储产生一个永久标记，从而生长一颗以 V_0 为根的最短路径树，在这颗树上每个顶点于根结点之间的路径皆为最短路径，我们最终只需要求解最短路径的结点就能计算出最短路径，算法流程图如图 2 所示：

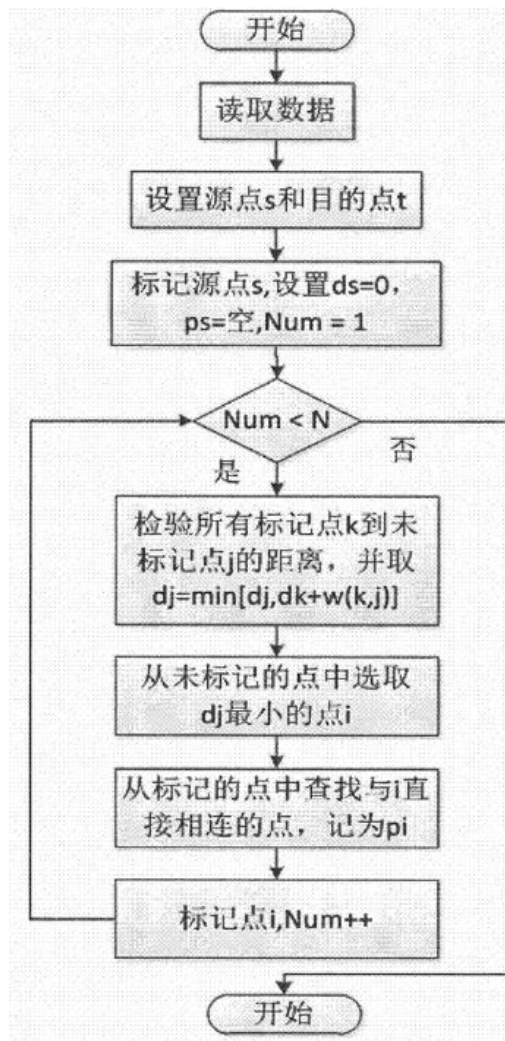


图 2.Dijkstra 程序框图
fig2.Dijkstra Algorithm ProgramCharts

2) 算法步骤:

为了方便程序撰写，我们对符号加以描述，如表 1 所示：

表 1.符号描述图表
table1.Symbol Description

Symbol	Description
S	具有永久标记的顶点集合
$L(v)$	v 的标记
$F(v)$	v 的父顶点，用于确定最短路径
$w = [w(v_i, v_j)]_{n \times m}$	待输入的加权图的带权邻接矩阵

① 初始化 L , F 集合，创建邻接矩阵；

② 寻找不在 S 中的顶点 u ，使得 $L(v)$ 最小，把 u 加入到 S 中，对不在 S 的顶点 v ，如 $L(v) > l(u) + w(u, v)$ ，则更新 $L(v)$, $F(v)$ ，即 $L(v) = L(u) + w(u, v)$, $F(v) = u$ ；

③ 重复步骤 2，直到所有顶点在 S 中为止

邻接矩阵的创建，参考图 1 的有向图，创建如矩阵（1）所示：

$$w = \begin{bmatrix} 0, 1, 12, \text{inf}, \text{inf}, \text{inf}; \\ \text{inf}, 0, 9, 3, \text{inf}, \text{inf}; \\ \text{inf}, \text{inf}, 0, \text{inf}, 5, \text{inf}; \\ \text{inf}, \text{inf}, 4, 0, 13, 15; \\ \text{inf}, \text{inf}, \text{inf}, \text{inf}, 0, 4; \\ \text{inf}, \text{inf}, \text{inf}, \text{inf}, \text{inf}, 0 \end{bmatrix} \quad (1)$$

对于无向图，即全向连通图，只需要把结点 u 到结点 n 的距离描述皆可；对于有向图，两个不连通之间的结点则用 inf （无穷大）替代。

3) Python 代码如图 3 所示:

```
import numpy as np

def startwith(start: int, mgraph: list) -> list:
    passed = [start]
    nopass = [x for x in range(len(mgraph)) if x != start]
    dis = mgraph[start]

    while len(nopass):
        idx = nopass[0]
        for i in nopass:
            if dis[i] < dis[idx]:
                idx = i

        nopass.remove(idx)
        passed.append(idx)

        for i in nopass:
            if dis[idx] + mgraph[idx][i] < dis[i]: dis[i] = dis[idx] + mgraph[idx][i]
    return dis

if __name__ == "__main__":
    inf = np.inf
    mgraph = [
        [0, 1, 12, inf, inf, inf],
        [inf, 0, 9, 3, inf, inf],
        [inf, inf, 0, inf, 5, inf],
        [inf, inf, 4, 0, 13, 15],
        [inf, inf, inf, inf, 0, 4],
        [inf, inf, inf, inf, inf, 0]
    ]

    dis = startwith(4, mgraph)
```

图 3.Python Dijkstra 算法代码
fig3.Python Dijkstra Algorithm Code

从程序中我们可以知道,我们创建一个 `passed` 变量用于存放已经走过的结点(最开始为起始结点), `nopass` 变量是除开起始结点之外的所有点,也就是未走过的结点。

然后在 `while` 循环中我们进行遍历查找, `dis` 为起始结点到其他结点的距离,我们先利用 `for` 循环将 `dis` 里面的起始点到其他结点的最短路径的索引求出来,然后更新我们的 `passed` 和 `nopass` 表,最后,若该点到最短路径的点的距离加上最短路径的点到下一结点的距离,小于起始结点到最短路径的点的下一结点的距离,则更新我们的 `dis` 表,完成最短路径的规划问题。

4) 函数调用方式:

直接运行代码，运行结果如图 3 所示:

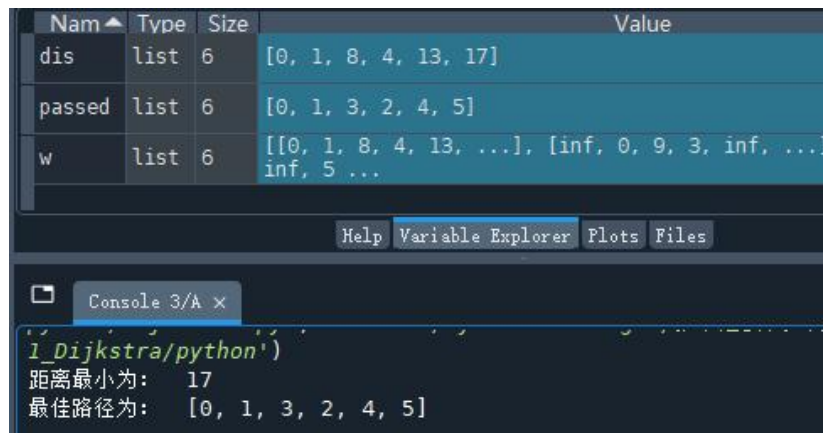


图 3.运行结果
fig3.code result

dis 变量存放的是最佳距离列表，w 存放是邻接矩阵，值得注意的是，与 Matlab 不同，Python 的索引下标是从 0 开始的，所以函数的运行结果为：从结点 0 到结点 5，最佳路径为[0, 1, 3, 2, 4, 5]，最短路径为 17。