离散数学大作业说明文档

问题背景

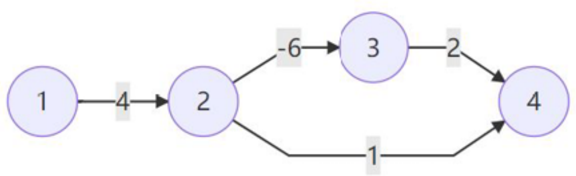
试设计算法，对任意的有向加权图（允许存在重边，负边权及环），对图中任意一个节点，求该节点到其他任意节点的最短路径，倘若图中存在负环（存在环且环上所有边权和为负，即存在两点间最短路为负无穷）需要判断。

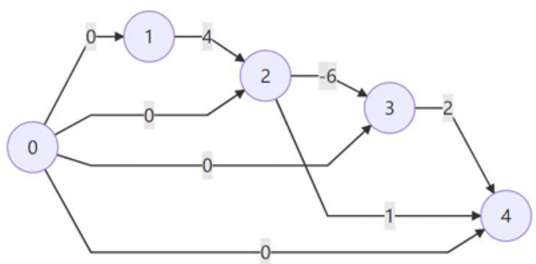
解决方案

该问题为一般情况下的求最短路径问题，需要对负环做出判断，也需要处理负边权存在的情况。首先列出常用的最短路算法：

算法	算法类型	应用条件	时间复杂度
朴素的$Dijkstra$算法	单源最短路	非负权图	$O(n^2)$
堆优化的$Dijkstra$算法	单源最短路	非负权图	$O(mlogm)$
$SPFA$算法（$Bellman-Ford$+队列优化）	单源最短路	负权图（可判断负环）	$O(m)\sim O(nm)$
$Floyd$算法	多源最短路	负权图（无法判断负环）	$O(n^3)$
$Johnson$算法	多源最短路	负权图（可判断负环）	$O(nmlogm)$

考虑到问题的一般性，我们采取的解决方案是基于$Dijkstra$算法和$SPFA$算法实现的$Johnson$算法。我们假定读者熟知$Dijkstra$算法和$SPFA$算法（具体内容将于PPT中展示），以下主要介绍$Johnson$算法。首先，我们将以下图为实例阐述$Johnson$算法的基本思想：





我们新建一个虚拟节点（我们设虚拟节点编号为0）。从这个节点到其他所有节点连一条边权为0的边，如上图右所示。接下来用$SPFA$算法求出从0号节点到其它节点的最短路，记为h_i。假如有一条从u到v，权值为w的边，则我们将该边权值重新设为$w+h_u-h_v$。接下来只需以每个点为起点，利用$Dijkstra$计算n次即可。

该算法的正确性证明如下：

- 在重新标记后的图上，从 s点到t 点的一条路径 的长度表达式如下：

  $$(w(s,p_1)+h_s-h_{p_1})+(w(p_1,p_2)+h_{p_1}-h_{p_2})+...+(w(p_k,t)+h_{p_k}-h_t)$$

  化简后得到：

  $$w(s,p_1)+w(p_1,p_2)+...+w(p_k,t)+h_s-h_k$$

  无论我们从 s到t 走的是哪一条路径，h_s-h_t 的值是不变的，这正与物理概念中势能的性质相吻合！为了方便，下面我们就把 h_i称为 i点的势能。新图中$s-t$的最短路的长度表达式由两部分组成，前面的边权和为原图中 $s-t$的最短路，后面则是两点间的势能差。因为两点间势能的差为定值，因此原图上$s-t$ 的最短路与新图上 $s-t$的最短路相对应。如此便证明了重新标注边权

后图上的最短路径仍然是原来的最短路径。

- 接下来我们需要证明新图中所有边的边权非负，因为在非负权图上，$Dijkstra$ 算法能够保证得出正确的结果。根据三角形不等式，图上任意一边 $<u,v>$ 上两点满足：$h_v \le h_u + w(u,v)$。这条边重新标记后的边权为 $w_{new} = w(u,v) + h_u - h_v \ge 0$ 。这样我们证明了新图上的边权均非负。

如此一来，我们就证明了 $Johnson$ 算法的正确性。

源码设计

本次大作业文件树如下：

```
├── data1.txt
├── data2.txt
├── data3.txt
├── data_generate.py
├── util.py
├── arithmetic.py
├── main.py
├── floyd.txt
├── johnson.txt
```

- `data1.txt, data2.txt, data3.txt`：

  存放自动生成的测试数据，分别对应非负图，有负边无负环图，含负环图
- `data_generate.py`：测试数据的自动生成器
- `util`：工具包，内部实现了`Graph`,`Heap`,`Distance`类
- `arithmetic`：算法实现代码
- `main`：验证算法正确性的主程序
- `floyd.txt, johnson.txt`：存放运行结果，分别对应$Floyd$算法和$Johnson$算法的运行结果

使用命令行 `fc` 命令验证 `floyd.txt, johnson.txt` 结果可以发现无负环时输出一致，有负环时可肉眼比对。同时，大作业中实现的所有算法均通过了洛谷强测数据，正确性得以检验。以下为 `Johnson` 算法的具体实现：

```python
def johnson(g: Graph):
    o, dis, tp = copy.deepcopy(g), g.get_dict(), g.head.keys()
    for key in tp:
        o.add(magic_num, key, 0)
    h = spfa(magic_num, o)
    if h is None:
        return None
    o = copy.deepcopy(g)
    for u in o.head.keys():
        e: Graph.Node = o.head[u]
        while e is not None:
            e.w, e = e.w + h[u] - h[e.to], e.nxt
    for u in o.head.keys():
        d = dijkstra(u, o)
        for v in d.keys():
            dis[u][v] = inf if d[v] == inf else d[v] - h[u] + h[v]
    return dis
```