Floyd算法

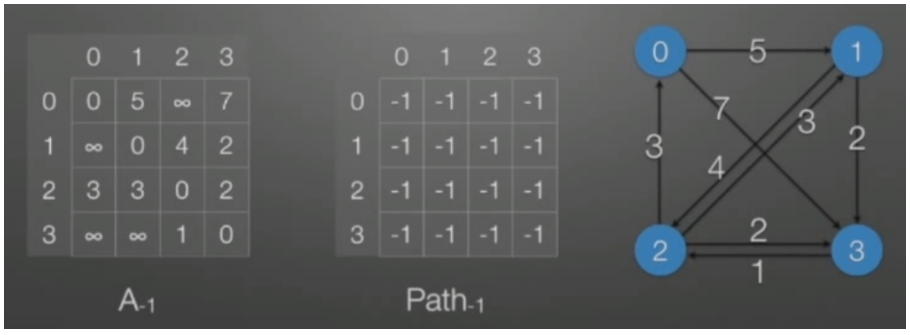
笔记本:	建模竞赛		
创建时间:	2020/9/3 10:44	**更新时间:**	2020/9/4 12:14
作者:	泡芙❀		
URL:	https://baike.baidu.com/item/Floyd%E7%AE%97%E6%B3%95		

Floyd算法

Floyd算法又称为插点法，是一种利用动态规划的思想**寻找给定的加权图中多源点之间最短路径的算法**，与Dijkstra算法类似。该算法名称以创始人之一、1978年图灵奖获得者、斯坦福大学计算机科学系教授罗伯特·弗洛伊德命名。

1. 构造矩阵

ps: 该状态转移矩阵的构造与官方存在差异，算法参考b站up主 CoCo可乐刨冰 的教程：https://www.bilibili.com/video/BV1LE411R7CS?from=search&seid=14923425004593178373

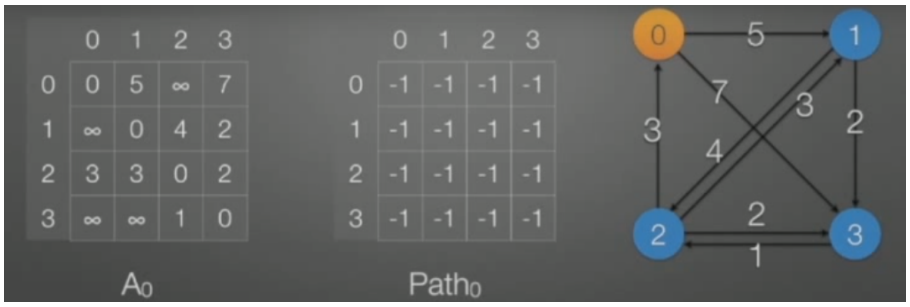


- A_{-1}储存了任意两顶点之间的**当前的**最短路径长（**A为最短路径矩阵**）
- $path_{-1}$任意两个顶点它们所在最短路径上的中间点，如：$path(1,0)=3$说明从顶点0到顶点0，需要经过顶点3（**path为状态转移矩阵**）

  对于每个顶点v，和任一顶点对 (i, j)，i ≠ j，v ≠ i，v ≠ j，
  如果A[i][j] > A[i][v] + A[v][j]，则将 A[i][j] 更新为 A[i][v] + A[v][j] 的值，
  并且将 Path[i][j] 改为v。
- 以v为中间点做后面的检测和更新操作
- path矩阵代表顶点之间在最短路径上的中间点，初值取-1。
- A与path的下标是当前的中间点，-1代表还没选中间点（初值）

- 若某个顶点与另一个顶点有直接的边相连，则以这两个顶点为下标的值为其边的长度，若没有直接的边相连，则初值为无穷大∞

1.1 第一步：以0为中间点



- 下列为待检测的顶点对（尖括号表示有向边），待检测表明可能是一条路径/一条边/无。

  {0, 1}, {0, 2}, {0, 3}, {1, 0}, {1, 2}, {1, 3},
  {2, 0}, {2, 1}, {2, 3}, {3, 0}, {3, 1}, {3, 2}

  先看前四个顶点对，因为包含顶点0，由于中间点为0，因此前4对不属于检测情况，略过（需要$v \neq i, v \neq j$）
- 检测点{1，2}：

  A[1][2]=4,A[1][0]=∞,A[0][2]=∞

  因此A[1][2]>A[1][0]+A[0][2]不成立，不更新中间点
- 检测点{1，3}：

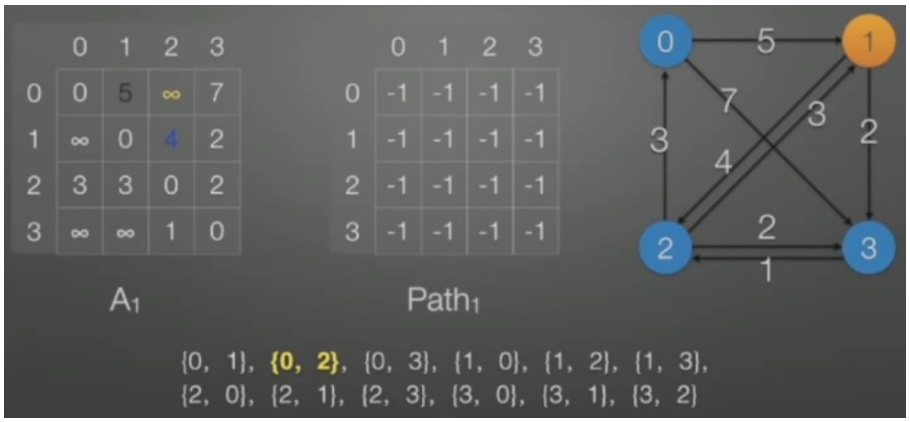
  A[1][3]=2,A[1][0]=∞,A[0][3]=7

  因此A[1][3]>A[1][0]+A[0][3]不成立，不更新中间点
- 检测点{2，0}：含中间点0，略过
- 检测点{2，1}：

  A[2][1]=3,A[2][0]=3,A[0][1]=5

  因此A[2][1]>A[2][0]+A[0][1]不成立，不更新中间点
- 同理检测所有点，最终所有点均无更新

1.2 第二步：以1为中间点



- 待测点中包含1的直接略过
- 待测点{0，2}：

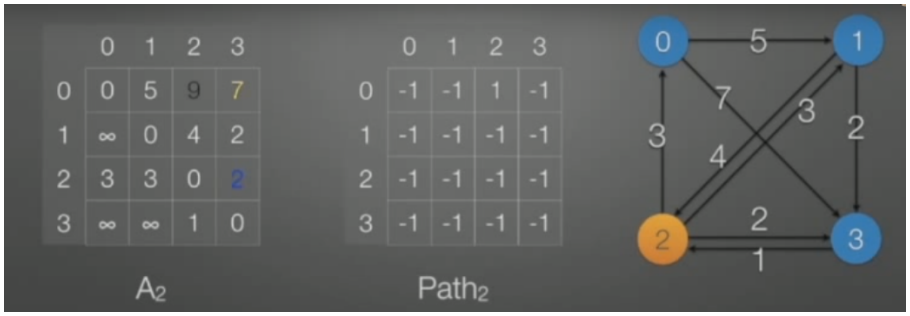
  A[0][2]=∞,A[0][1]=5,A[1][2]=4

  因此A[1][2]>A[1][0]+A[0][2]成立，则更新最短路径A[0][2]=9，更新中间点path[0][2]=1
- 待测点{0，3}：

  A[0][3]=7,A[0][1]=5,A[1][3]=2

  因此A[1][2]=A[1][0]+A[0][2]，不更新
- 待测点{2，0}：不更新
- 同理，注意∞>∞+∞不成立

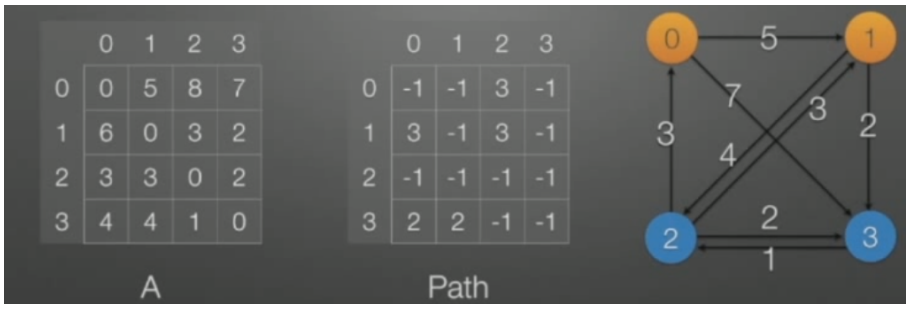
1.3 第三步：以2为中间点



其中，path是每步不断累加更新

1.4 第四步：以3为中间点

最终结果：



2. 利用path数组找最短路径

- 找从1到0的最短路径：
  - 若$A[1,0]=\infty$，则不存在最短路径，否则：
    - 查path[1][0]=3，说明中间点为3
    - 查path[1][3]=-1，说明从1到3有直接的路径
    - 查path[3][0]=2，说明中间点为2
    - 查path[3][2]=-1，说明从3到2有直接的路径

    - 查path[2][0]=-1，说明从2到0有直接的路径
- 找到最短路径**1-3-2-0**

3. 优缺点

Floyd算法适用于APSP(All Pairs Shortest Paths，多源最短路径)，是一种动态规划算法，稠密图效果最佳，边权可正可负。此算法简单有效，由于三重循环结构紧凑，对于稠密图，效率要高于执行|V|次Dijkstra算法，也要高于执行|V|次SPFA算法。

- 优点：容易理解，可以算出任意两个节点之间的最短距离，代码编写简单。
- 缺点：时间复杂度比较高，不适合计算大量数据。 [5]

c++代码

```
void printPath(int u,int v,
               int path[][max])
{
  if(path[u][v] == -1)
    直接输出;
  else
  {
    int mid = path[u][v];
    printPath(u, mid, path);
    printPath(mid, v, path);
  }
}
```

```
void Floyd(int n, float MGraph[][n], int Path[][n])
{
    int i, j, v;
    int A[n][n];
    for(i = 0; i < n; ++i)
        for(j = 0; j < n; ++j)
        {
            A[i][j] = MGraph[i][j];
            Path[i][j] = -1;
        }

    for(v = 0; v < n; ++v)
        for(i = 0; i < n; ++i)
            for(j = 0; j < n; ++j)
                if(A[i][j] > A[i][v]+A[v][j])
                {
                    A[i][j] = A[i][v]+A[v][j];
                    Path[i][j] = v;
                }
}
```