Path in Graph

Single-source shortest path (SSSP)

考察一个有权图 G ，边权值非负，图中有一给定源点 s ， 求从 s 到图中其他点的最短路径

当所有边的权值相等时简单的 BFS 即可解决

Dijkstra

基本思想：类似 Prim 在 fringe 中贪心选择权值最小的节点。fringe 即当前可达的节点，而贪心选择的权值即是当前已知的源点到节点的最短路径的权值

随着 fringe 中的节点被确定最短路径，其邻居中本来不可达的被加入 fringe，已在 fringe 中的检查其最短路径权值是否需要更新

实现如下

Dijkstra(G, s)

```
Initialize all vertices as UNSEEN
Initialize queNode as empty
s.dis := 0
foreach neighbor w of s do
    w.pathEdge := sw
    queNode.INSERT(w, sw.weight)
while queNode != empty do
    x := queNode.EXTRACT-MIN()
    x.dis := x.priority
    Classify x.pathEdge as SHORTEST-PATH-EDGE
    UPDATE-FRINGE(x, queNode)
```

UPDATE-FRINGE(v, queNode)

```
foreach neighbor w of v do
    newPriority := v.priority + vw.weight
    if w is UNSEEN then
        w.pathEdge := vw
        queNode.INSERT(w, newPriority)
    else
        if newPriority < w.priority then
            w.pathEdge := vw
            queNode.decreaseKey(w, newPriority)
```

> 在 CLRS 中，定义 $v.d$ 为从 s 到 v 的最短路径的 upper-bound，定义 $v.\pi$ 为 v 最短路径上的前驱，对于边 uv 引入 RELAX 操作
>
> ```
> RELAX(u, v, w)
>     if v.d > u.d + w(u, v)
>         v.d = u.d + w(u, v)
>         v.pi = u
> ```
>
> 同时引入 INITIALIZE-SINGLE-SOURCE
>
> ```
> INITIALIZE-SINGLE-SOURCE(G, s)
>     foreach vertex v in G
>         v.d = INF
>         v.pi = NIL
>     s.d = 0
> ```
>
> 则 DIJKSTRA 可以表示为
>
> ```
> DIJKSTRA(G, w, s)
>     INITIALIZE-SINGLE-SOURCE(G, s)
>     S = empty
>     Q = G.V
>     while Q != empty
>         u = EXTRACT-MIN(Q)
>         S = S + u
>         foreach neighbor v of u
>             RELAX(u, v, w)
> ```

Correctness of Dijkstra

Dijkstra 的执行阶段可以分为 n 个阶段

$$D^{(1)}, D^{(2)}, \ldots, D^{(n)}$$

其中第 k 个阶段 $D^{(k)}$ 表示从源点到图中 k 个节点（包括源点）的最短路径已经确定。

可以基于数学归纳法对 Dijkstra 执行的阶段归纳证明其正确性

Basis. $D^{(0)}$ 即只确定源点到源点的最短路径，正确性显然。

I.H. 对任意 $0 \leqslant j < k$ ，$D^{(j)}$ 能确定源点到图中 j 个节点的最短路径

Ind.Step. 考虑 $D^{(k)}$ 选择的节点 z ，假设 z 与已确定最短路径的节点 y 相连，则需要证明路径

$$p_1 = s \rightsquigarrow y \to z$$

即是 s 到 z 的最短路径。采用反证法，假设存在一条从 s 到 z 的路径 p_2 满足

$$p_2.weight < p_1.weight$$

则 p_2 中一定有一部分属于已确定最短路径的节点。假设

$$p_2 = s \rightsquigarrow z_{a-1} \to z_a \rightsquigarrow z$$

其中 $s \rightsquigarrow z_{a-1}$ 上的点均已确定最短路径。由于 Dijkstra 的贪心原则，可知

$$(s \rightsquigarrow z_{a-1} \to z_a).weight \geqslant (s \rightsquigarrow y \to z).weight$$

而又由于图中**边权值非负**

$$(z_a \rightsquigarrow z).weight \geqslant 0$$

故可得

$$\begin{gathered}
(s \rightsquigarrow z_{a-1} \to z_a).weight + (z_a \rightsquigarrow z).weight = p_2.wieght \\
(s \rightsquigarrow y \to z).weight + 0 = p_1.weight \\
p_2.weight \geqslant p_1.weight
\end{gathered}$$

而这与之前的假设矛盾。故正确性得证

> **定理 11.1** 对于有权图 G （有向或无向），每条边的权值非负，Dijkstra 算法总能计算出从指定源点到其他所有点的最短路径

Analysis

Dijkstra 算法的代价本质上和 Prim 算法是一致的。当使用数组实现 priority queue 时代价为 $O(n^2+m)$ ，当使用堆实现 priority queue 时代价为 $O((m+n)\log n)$

Dijkstra skeleton

基于 Dijkstra 可以实现很多 SSSP 问题的变体

如节点也有权值需要考虑

或是求路径上最小权边最大的路径（水管流速受限于最细的水管，bottleneck）

或是求路径上最大权边最小的路径（汽车在城市间运行，最长距离不能超过油箱的容量）

All-pair shortest path (APSP)

对图中任意点对求其最短路径。问题的简化版：求图中邻接关系的 传递闭包

Transitive Closure

图中的传递闭包描述了图中顶点的可达性。设传递闭包为 R ，则 $v_i R v_j \iff$ 图中从 v_i 可达 v_j，显然传递关系是由图最基本的邻接关系组合而成的

基本思想：若图中有边 $s_i s_k, s_k s_j$ 则可插入一条 shortcut $s_i s_j$

朴素思想：对于所有点对，遍历所有可能的中间点，当添加了新的 shortcut 则要继续遍历

```
while(传递闭包矩阵R发生了变化)
    for(i = 1; i <= n; i++)
        for(j = 1; j <= n; j++)
            for(k = 1; k <= n; k++)
                R[i][j] = R[i][j] || (R[i][k] && R[k][j])
```

显然，这样做的代价为 $O(n^4)$

其余改进思想还有对每条边 xv ，遍历所有可能的源点 u ，若 ux 可达则添加 shortcut uv

```
while(传递闭包矩阵R发生了变化)
    for all vertices u
        for every edge (x, v)
            R[u][v] = R[u][v] || (R[u][x] && R[x][v])
```

代价为 $O(n^2m)$ ，而边的数量 m 可能为 $O(n^2)$ ，即最坏情况代价仍是 $O(n^4)$

还有基于可达路径边数递归，遍历所有路径长度可能性和点对

```
for(k = 1; k < n; k++)
    for all vertices u
        for all vertices v
            for all vertices x pointing to v
                r_k[u][v] = r_{k-1}[u][v] || (r_{k-1}[u][x] && r_{k-1}[x][v])
```

四重循环，显然代价仍是 $O(n^4)$

Warshall Algorithm

更改了循环的顺序，基于**中继节点序号范围**循环

```
for(k = 1; k <= n; k++)
    for(i = 1; i <= n; i++)
        for(j = 1; j <= n; j++)
            R[i][j] = R[i][j] || (R[i][k] && R[k][j])
```

只需要 $O(n^3)$ 的代价即可计算出 transitive closure

设第 k 次循环后的矩阵为 $R^{(k)}$，传递闭包矩阵元素 r_{ij} 的值在第 k 次循环执行后为 $r_{ij}^{(k)}$

$R^{(0)}$ 即是原本的邻接矩阵

$$r_{ij}^{(k)} = \begin{cases} a_{ij} & k = 0 \\ r_{ij}^{(k-1)} \vee (r_{ik}^{(k-1)} \wedge r_{kj}^{(k-1)}) & k > 0 \end{cases}$$

算法正确性基于如下定理

> 如果从节点 s_i 到 s_j （$i \neq j$）存在简单路径，且路径上中继节点的最高序号不超过 k，则 $r_{ij}^{(k)} = TRUE$
>
> 对 k 归纳
>
> Basis. $r_{ij}^{(0)} = TRUE \iff s_i s_j \in E$
>
> I.H. 对于 $0 \leqslant j < k$ 上述结论成立
>
> Ind.Step. s_i 到 s_j 的中继节点的最高序号不超过 k 的简单路径可看作从 s_i 到 s_k 的简单路径和从 s_k 到 s_j 的简单路径，且路径上中继节点序号最大值为 $h_1, h_2, h_1 < k, h_2 < k$，根据 I.H.，可得

> $$r_{ik}^{(h_1)} = TRUE$$
> $$r_{kj}^{(h_2)} = TRUE$$
>
> 由于矩阵元素从 FALSE 变为 TRUE 的过程不可逆，故
>
> $$r_{ik}^{(k-1)} = TRUE$$
> $$r_{kj}^{(k-1)} = TRUE$$
>
> 可得 $r_{ij}^{(k)} = TRUE$ ，得证

> 如果 $r_{ij}^{(k)} = TRUE$ ，则存在一条从节点 s_i 到 s_j （$i \neq j$）的简单路径，且路径上中继节点最大序号不超过 k，记作 $(s_i, s_j)^k$
>
> 对 k 归纳
>
> Basis. $r_{ij}^{(0)} = TRUE \iff s_i s_j \in E$ ，存在 $(s_i, s_j)^0$
>
> I.H. 对于 $0 \leqslant j < k$ 上述结论成立
>
> Ind.Step. 不失一般性设 r_{ij} 在第 k 次循环后变为 TRUE（否则根据 I.H. 得证），则可得
>
> $$r_{ik}^{(k-1)} = TRUE$$
> $$r_{kj}^{(k-1)} = TRUE$$
>
> 根据 I.H.，存在 $(s_i, s_k)^{(k-1)}, (s_k, s_j)^{(k-1)}$ ，则存在路径 $s_i \rightsquigarrow s_k \rightsquigarrow s_j$，显然，该路径为 $(s_i, s_j)^k$

APSP and Floyd-Warshall Algorithm

最短路径有一条很重要的属性，即最短路径的子路径仍为最短路径

> CLRS 3rd edition p.645
>
> Lemma 24.1 Subpaths of shortest paths are shortest paths
>
> Given a weighted, directed graph $G = (V, E)$ with weight function $w : E \to \mathbb{R}$, let $p =< v_0, v_1, \ldots, v_k >$ be a shortest path from vertex v_0 to vertex v_k and, for any i and j such that $0 \leqslant i \leqslant j \leqslant k$, let $p_{ij} =< v_i, v_{i+1}, \ldots, v_j >$ be the subpath of p from vertex v_i to v_j . Then, p_{ij} is a shortest path from v_i to v_j

基于求传递闭包的 Warshall 算法可以很简单地得到求 APSP 的 Floyd-Warshall 算法，只需要使用最短距离矩阵 D 代替传递闭包矩阵 R ，初始化时将其中的 1 替换为边权，0 替换为 ∞ ，并且将主对角元置为 0

定义子问题 $d(i,j,k)$ 为从顶点 v_i 到 v_j 的中继节点序号不超过 k 的最短路径，显然此路径只有两种可能：包含顶点 v_k 或不包含顶点 v_k 。对于第一种可能，问题降级为 $d(i,j,k-1)$ ，而对于第二种可能，若最短路径包含顶点 v_k ，则可将路径分为两段 $v_i \rightsquigarrow v_k, v_k \rightsquigarrow v_j$ ，显然这两段路径中继节点的序号均不超过 $k-1$ ，问题降级为 $d(i,k,k-1)+d(k,j,k-1)$ 。求这两种可能中的较小值即可。

将其加入 Warshall 算法的框架，对于第 k 次循环后的矩阵元素 $d_{ij}^{(k)}$ ，其代表了从 v_i 到 v_j 的中继节点序号不超过 k 的最短路径权值（此性质可通过对 k 归纳证明），显然根据上文对子问题的分析，有

$$d_{ij}^{(k)} = \begin{cases} w_{ij} & k=0 \\ \min\{d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\} & k>0 \end{cases}$$

```
for(k = 1; k <= n; k++)
    for(i = 1; i <= n; i++)
        for(j = 1; j <= n; j++)
            D[i][j] = min(D[i][j], D[i][k] + D[k][j])
```

> 思考：表面上需要 n 个距离矩阵，但实际上只需要在一个矩阵上迭代运算即可。因为对子问题的合理划分使得每次计算只用到前一级子问题的解，不需要更早的子问题的解，故直接覆盖即可。
>
> 进一步思考：假设已经获得了第 $k-1$ 步子问题的解，为矩阵 $D^{(k-1)}$ ，在逐步计算并修改矩阵中元素时为何不会影响之后的计算。
>
> 考虑计算矩阵元素 $d_{ij}^{(k)}$ 时，需要的上一步子问题的解为 $d_{ij}^{(k-1)}, d_{ik}^{(k-1)}, d_{kj}^{(k-1)}$ ，不难看出当 i,j 均不为 k 时，修改后的矩阵元素不会出现在之后计算中。不失一般性，令 $i=k$ ，则 $d_{ij}^{(k)} = d_{kj}^{(k)} = \min\{d_{kj}^{(k-1)}, d_{kk}^{(k-1)} + d_{kj}^{(k-1)}\}$ ，而在初始化及后续的计算中，主对角元的元素一直为 0，故 $d_{ij}^{(k)} = d_{kj}^{(k)} = d_{kj}^{(k-1)}$ ，事实上矩阵中这些元素（第 k 行及第 k 列）并没有被修改，有
>
> $$d_{ik}^{(k)} = d_{ik}^{(k-1)} \\ d_{kj}^{(k)} = d_{kj}^{(k-1)} \\ i,j = 1,2,\ldots,n$$
>
> 故逐步修改矩阵元素时之前修改的结果不会影响之后的计算

Floyd-Skeleton

基于 Floyd-Warshall 可实现很多 APSP 的变体

如在算法中插入一些步骤可实现构建前驱/后继路由表

或是解决所有点对间的路径上最小权边最大的路径/路径上最大权边最小的路径问题

Negative Weight Edge

当图中有权值为负的边时，最短路径算法能否正确工作?

- Dijkstra 当边权为负时不能正确工作，见其正确性证明
- Floyd-Warshall 算法即使图中有负权边也能正常工作，但要求图中不能有负权的环，因为当图中有负权的环的情况下“最短路径”不是良定义的
- Bellman-Ford 算法可以解决带负权边图的 SSSP 问题并检测出负权环