袁雨
PB20151804

25.2-5 假定我们修改式(25.7)对等式的处理办法如下：

$$\pi_{ij}^{(k)} = \begin{cases} \pi_{ij}^{(k-1)} & \text{若 } d_{ij}^{(k-1)} < d_{ik}^{(k-1)} + d_{kj}^{(k-1)} \\ \pi_{kj}^{(k-1)} & \text{若 } d_{ij}^{(k-1)} \geqslant d_{ik}^{(k-1)} + d_{kj}^{(k-1)} \end{cases}$$

请问这种前驱矩阵 Π 的定义正确吗？

$$\pi_{ij}^{(k)} = \begin{cases} \pi_{ij}^{(k-1)} & \text{若 } d_{ij}^{(k-1)} \leqslant d_{ik}^{(k-1)} + d_{kj}^{(k-1)} \\ \pi_{kj}^{(k-1)} & \text{若 } d_{ij}^{(k-1)} > d_{ik}^{(k-1)} + d_{kj}^{(k-1)} \end{cases} \tag{25.7}$$

正确.

$\pi_{ij}^{(k)}$ 为从结点 i 到结点 j 的一条所有中间结点都取自集合 $\{1,2,\cdots,k\}$ 的最短路径上 j 的前驱结点.

当 $d_{ij}^{(k-1)} = d_{ik}^{(k-1)} + d_{kj}^{(k-1)}$ 时，$i \rightsquigarrow j$ 与 $i \rightsquigarrow k \rightsquigarrow j$ 的最短路径长度相同，中间结点加上 k 后，j 的前驱结点既可取 $\pi_{ij}^{(k-1)}$ 也可取 $\pi_{kj}^{(k-1)}$.

25.3-3 假定对于所有的边 $(u, v) \in E$，我们有 $w(u, v) \geqslant 0$。请问权重函数 w 和 $\hat{w}$ 之间是什么关系？

$w = \hat{w}$

$\forall u, v \in V$, $\hat{w}(u,v) = w(u,v) + h(u) - h(v)$

$h(u) = \delta(s, u)$

$\because \forall x \in V$, $w(u,x) \geqslant 0$, 且 $w(s,u) = 0$

故 $h(u) = \delta(s,u) = w(s,u) = 0$

同理, $h(v) = \delta(s,v) = w(s,v) = 0$

故 $\forall (u,v) \in E$, $\hat{w}(u,v) = w(u,v) + h(u) - h(v) = w(u,v)$

25.3-5 假定在一个权重函数为 w 的有向图 G 上运行 Johnson 算法。证明：如果图 G 包含一条权重为 0 的环路 c，那么对于环路 c 上的每条边 (u, v)，$\hat{w}(u, v) = 0$。

环路 c 上的每条边 (u,v), 有 $h(u) = \delta(s,u) = \delta(s,v) = h(v)$

$\hat{w}(u,v) = w(u,v) + h(u) - h(v) = w(u,v)$

故 $\sum_{(u,v) \in c} \hat{w}(u,v) = \sum_{(u,v) \in c} w(u,v) = 0$

又 $\forall (u,v) \in E$, $\hat{w}(u,v) \geqslant 0$, 故对环路 c 上的每条边 (u,v), $\hat{w}(u,v) = 0$

32.1-2 假设在模式 P 中所有字符都不相同。试说明如何对一段 n 个字符的文本 T 加速过程 NAIVE-STRING-MATCHER 的执行速度，使其运行时间达到 $O(n)$。

```
NAIVE-STRING-MATCHER(T,P)
1  n=T.length
2  m=P.length
3  for s = 0 to n−m
4      if P[1..m] == T[s+1..s+m]
5          print "Pattern occurs with shift"s
```

```
NAIVE - STRING - MATCHER (T, P)
n = T. length
m = P. length
s = 0
while  s <= n-m :
        j = 1
        while T[s+j] == P[j]:
                if j == m :
                    print " Pattern occurs with shift" s
                    break
                j += 1
        if j==1 :
                s += 1
        else:
                s = s+j
```

该算法只扫描 T 的每个字符 1 次，时间复杂度为 $O(n)$.

32.2-2 如何扩展 Rabin-Karp 算法，使其能解决如下问题：如何在文本字符串中搜寻出给定的 k 个模式中的任何一个出现？起初假设所有 k 个模式都是等长的，然后扩展你的算法以适用于不同长度的模式。

```
RABIN-KARP-MATCHER(T,P,d,q)
 1  n = T.length
 2  m = P.length
 3  h = d^(m-1) mod q
 4  p = 0
 5  t_0 = 0
 6  for i = 1 to m                    // preprocessing
 7      p = (dp+P[i]) mod q
 8      t_0 = (dt_0+T[i]) mod q
 9  for s = 0 to n-m                  // matching
10      if p == t_s
11          if P[1..m] == T[s+1..s+m]
12              print"Pattern occurs with shift"s
13      if s<n-m
14          t_(s+1) = (d(t_s-T[s+1]h)+T[s+m+1]) mod q
```

k 个模式等长：

```
RABIN-KARP-MATCHER (T, P, d, q)
n = T.length          // T[1...n]
m = P[0].length       // P[1...k][1...m]
h = d^(m-1) mod q
for i=1 to k:
    p[i] = 0
t_0 = 0
for i=1 to m:
    t_0 = (d t_0 + T[i]) mod q
    for j=1 to k:
        p[j] = d p[j] + P[j][i] mod q
for s=0 to n-m:
    for j=1 to k:
        if p[j] = t_s
            if P[j][1...m] = T[s+1...s+m]
                print "Pattern occurs with shift" s
    if s < n-m:
        t_(s+1) = (d(t_s - T[s+1]h) + T[s+m+1]) mod q
```

时间复杂度为 $O(k) + O(mk) + O(km(n-m+1))$

$= O(km(n-m+1))$

k 个模式不等长：

```
RABIN-KARP-MATCHER (T, P, d, q)
n = T.length
for i=1 to k:
    m[i] = P[i].length
    h[i] = d^(m[i]-1) mod q
    p[i] = 0
    t[j] = 0
for i=1 to k:
    for j=1 to m[i]:
        t_0[i] = (d t_0[i] + T[j]) mod q
        p[i] = (d p[i] + P[i][j]) mod q
for s=0 to n-m:
    for j=1 to k:
        if p[j] == t_s[j]:
            if P[j][1...m[j]] == T[s+1...s+m[j]]
                print "Pattern occurs with shift" s
    for i=1 to k:
        if s < n-m[i]
            t_(s+1)[i] = (d(t_s[i] - T[s+1]h[i]) + T[s+m+1]) mod q
```

时间复杂度为 $O(k) + O(k \cdot \max\{P[1\cdots k].length\}) + O(k \cdot \max\{P[1\cdots k].length\} \cdot (n-m+1))$

$= O(k \cdot \max\{P[1\cdots k].length\} \cdot (n-m+1))$

32.2-3 试说明如何扩展 Rabin-Karp 算法用于处理以下问题：在一个 $n \times n$ 的二维字符数组中搜索一个给定的 $m \times m$ 的模式。（该模式可以在水平方向和垂直方向移动，但是不可以旋转。）

```
RABIN-KARP-MATCHER(T, P, d, q)
for k = 1 to n:
    l_T = T[k].length
    for j = 1 to m:
        l_p = P[j].length
        h = d^(m-1) mod q
        p = 0
        t_0 = 0
        for i = 1 to m:
            p = (dp + P[i]) mod q
            t_0 = (dt_0 + T[i]) mod q
        for s = 0 to n-m:
            if p == t_s:
                if P[j][1...m] = T[k][s+1...s+m]:
                    print("Pattern occurs with shift", s)
            if s < n-m:
                t_{s+1} = (d(t_s - T[s+1]h) + T[s+m+1]) mod q
```

32.4-3 试说明如何通过检查字符串 PT（由 P 和 T 连结形成的长度为 $m+n$ 的字符串）的 π 函数来确定模式 P 在文本 T 中的出现位置。

设模式 P 在文本 T 中出现的位置集合为 M，q 为 PT 的下标

则 $M = \{q - m \mid m \in \Pi^*(q) \text{ 且 } q \geq 2m\}$

32.4-7 写出一个线性时间的算法，以确定文本 T 是否是另一个字符串 T' 的循环旋转。例如 arc 和 car 是彼此的循环旋转。

若 T 与 T' 的长度不相等，则 T 与 T' 一定不是彼此的循环旋转。
若长度相等，令文本串为 TT (两个 T 拼接)，模式串为 T'，使用 KMP 算法判断 T' 是否为 TT 的子串，
若 T' 是 TT 的子串，则 T 是 T' 的循环旋转，若否，则 T 不是 T' 的循环旋转。

```
Cyclic-Rotation (T, T')
    if T.length != T'.length
        return FALSE
    else:
        TT = T+T
        if KMP(TT, T') = TRUE    #使用KMP算法判断T'是否为TT的子串.
            return TRUE
        else:
            return FALSE
```