۸. مساله کوتاهترین مسیر بین هر دو راس

پیدا کردن همه‌ی کوتاه‌ترین مسیرها: هدف در این مسئله این است که کوتاه‌ترین مسیر بین هر دو راسی در گراف پیدا شود.

اوّلین و سادهترین راه‌حلی که به ذهن می‌رسد، این است که از همه‌ی رئوس، کمترین مسیر تا بقیّه را پیدا کنیم(به کمک الگوریتم پیدا کردن همه‌ی کوتاه‌ترین مسیرها از مبدا مشترک). که شبه‌کد آن به صورت زیر می‌شود:

```
for s in V:
        sssp(s)
```

یادآوری: *الگوریتم یافتن کوتاه‌ترین مسیرها از مبدا مشترک را می‌توان به کمک دو ماتریس که یکی از آن‌ها ماتریس اجداد و دیگری ماتریس فاصله‌ها است، پیاده‌سازی کرد.*

برای حل مسئله‌ی بالا، راه‌حل‌های دیگری نیز پیشنهاد می‌شود که تکیه بر برنامه‌نویسی پویا دارند.
(برنامه‌نویسی پویا به نوعی از حل مسئله گفته می‌شود که در آن تعریف زیرمس‌سله بهینه است و نتایج هر زیر مسئله برای استفاده‌های بعدی در یک حافظه ذخیره می‌شود.)

حال مسئله را به این صورت تغییر می‌دهیم که به دنبال پیدا کردن طول کوتاه‌ترین مسیر از راس‌ها به هم هستیم به طوری که تعداد یال‌های مسیر محدودیت داشته باشد. به این وسیله ماتریس زیر را تعریف می‌کنیم.

$L_{ij}(k)$ = Distance of the shortest path from vertex i to vertex j that has maximum number of k edges

می‌دانیم جواب نهایی که ما به دنبال آن هستیم باید حدّاکثر به تعداد راس‌های گراف منهای یک یال داشته باشد (حداکثر به اندازه‌ی تعداد یال‌های درخت پوشای کمینه.) پس جواب نهایی ما برابر با ماتریس بالا به ازای تعداد راس منهای یک است.

$D = L\ (n-1)$

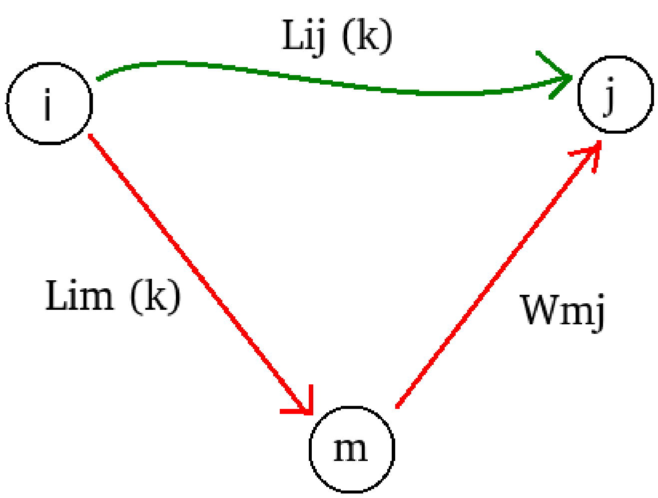
این ماتریس به ازای ۱، برابر همان ماتریس مجاورت در گراف وزندار است که به صورت زیر تعریف می‌شود.

$$Lij\ (1) = Wij = \begin{cases} Wij & (i, j) \text{ in } E \\ \infty & (i, j) \text{ not in } E \\ 0 & (I = j) \end{cases}$$

حال می‌توان از ماتریس به ازای ۱، به ماتریس به ازای ۲، به ... و نهایتا به ماتریس به ازای تعداد راس منهای یک رسید.

$Lij\ (k) \rightarrow Lij\ (k + 1)$

$Lij\ (k) = \text{Min} \{ Lij\ (k), \text{for each } m \text{ in } V: (Lim\ (k) + Wmj) \}$



این عمل برای هر خانه‌ی ماتریس تکرار می‌شود و برای پیدا کردن هر ماتریس باید ماتریس قبلی ان را نیز داشت. پس داریم:

$O((n * (n ^ 2)) * (n – 2)) = O(n ^ 4)$

برای تسهیل فرآیند به دست آوردن این ماتریس‌ها، می‌توان نوع خاصی از ضرب ماتریس‌ها را تعریف کرد. در حالت کلّی ضرب ماتری به صورت زیر تعریف می‌شود:

$A * B = C \quad Cij = \sum Aim * Bmj$

حال ما با تغییر مجموع به کمینه و ضرب به جمع داریم:

$Cij = Min (Aim + Bmj)$

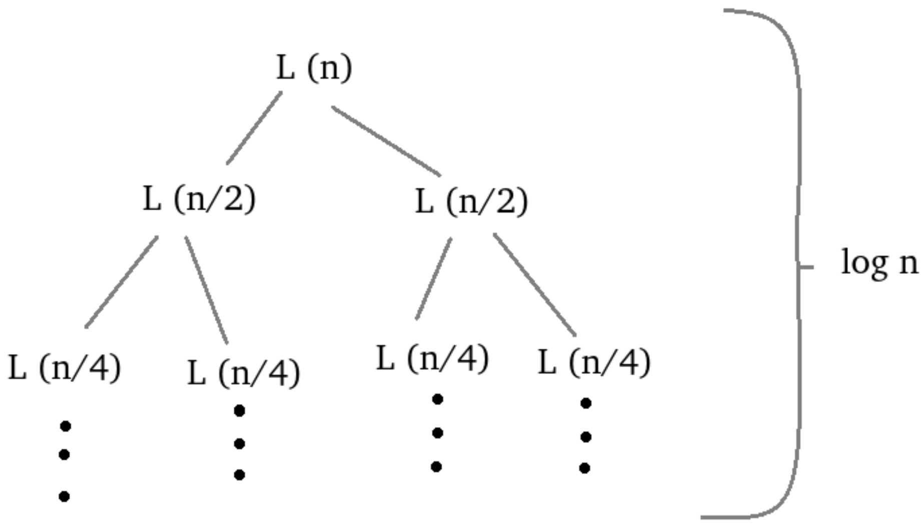
به کمک این نوع ضرب می‌توانیم که ماتریس خود را به دست بیاوریم. یعنی

$Lij (k + 1) = Lij (k) * Wij$

حال برای سبک‌تر کردن الگوریتم، می‌توانیم محاسبات زیر را لحاظ کنیم:

$(a ^ n) = a * a * … * a \quad (n \text{ times } a) = (a ^ (n/2)) * (a ^ (n/2)) = ((a ^ (n/4)) * (a ^ (n/4))) * ((a ^ (n/4)) * (a ^ (n/4))) = …$

پس به کمک یک درخت دو دویی می‌توان محاسبات را سبک‌تر کرد.



$$O((n * (n \wedge 2)) * \log n) = O((n \wedge 3) \log n)$$

راه حل دیگری که برای مسئله عنوان می‌شود، الگوریتم فلوید-وارشال است. این الگوریتم نیز بر پایه‌ی برنامه‌نویسی پویا برقرار است.

ابتدا ماتریسی به این صورت تعریف می‌کنیم که در آن کوتاه‌ترین مسیرها را با گذر از گره‌های خاصی به دست می‌آوریم.

$L_{ij}(k)$ = Distance of the shortest path from vertex i to vertex j that can only include vertexs 1 to k

با این تعریف داریم:

$$L_{ij}(0) = W_{ij} \quad \rightarrow \quad L(0) = W$$

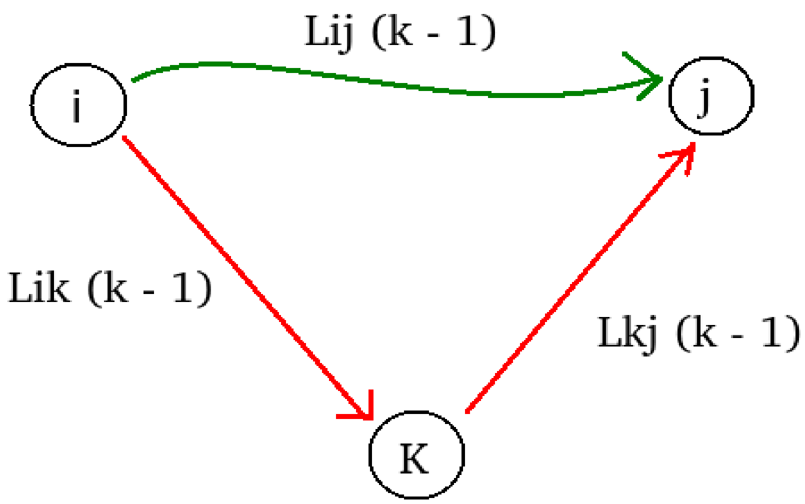
$$L_{ij}(n) = \delta(i, j) \quad \rightarrow \quad L(n) = D$$

که این همان جواب مسئله‌ی ماست.

حال داریم:

Lij (k) = {Passing vertex k} or {Not passing vertex k} = { Lik (k – 1) + Lkj (k – 1) } or {Lij (k – 1)}

Lij (k) = Min {Lij (k-1), (Lik (k – 1) + Lkj (k – 1))}



محاسبه‌ی هر خانه با پیچیدگی زمانی ۱ امکان‌پذیر است و این کار به تعداد رئوس برای خانه‌های ماتریس انجام می‌شود.

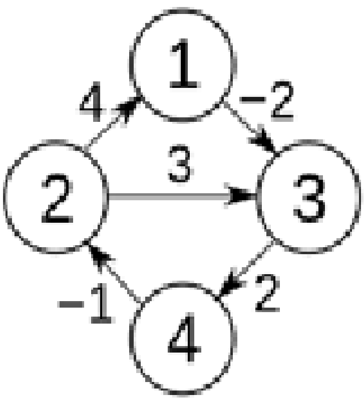
O (1 * (n ^ 2) * n) = O(n ^ 3)

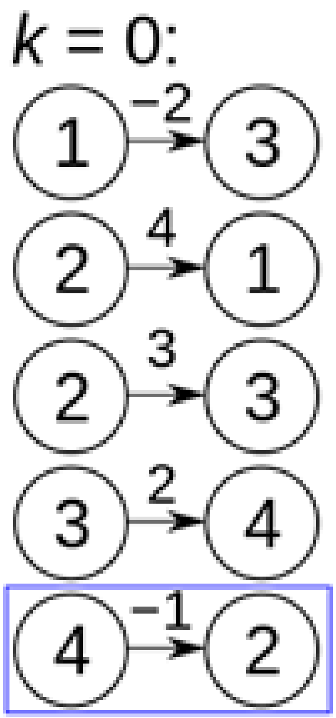
نکته: *از این الگوریتم برای تشخیص دورهای منفی هم می‌توان استفاده کرد. به این صورت که می‌دانیم مقدار اوّلیّه روی قطر اصلی ماتریس (برابر با فاصله‌ی یک راس تا خودش)، صفر است. اگر پس از اجرای الگوریتم مقدار منفی‌ای روی قطر اصلی ظاهر شود، نشان از این دارد که در گراف دور منفی وجود داشته است.*

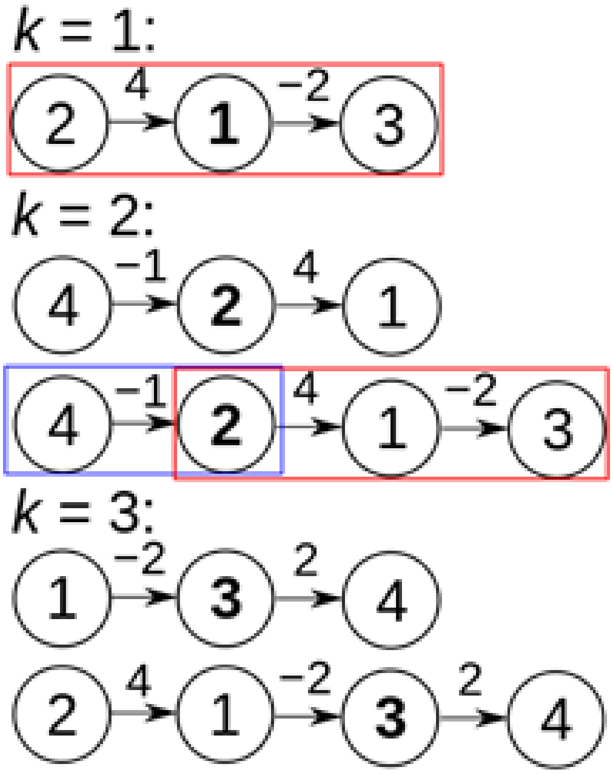
یادآوری: *دور منفی دوری است که مجموع وزن یال‌های آن منفی شود.*

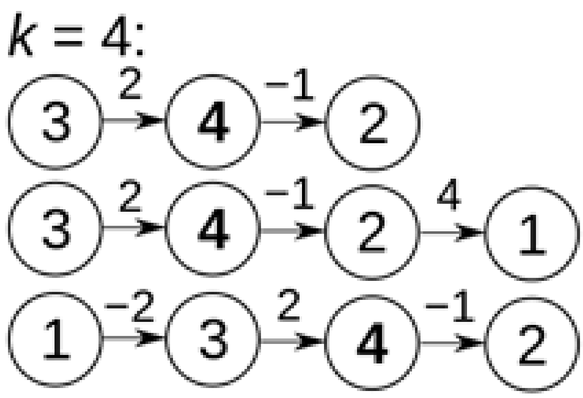
نکته: *با توجّه به راه‌حل‌های عنوان شده می‌توان مفهوم جدیدی به نام راس مرکزی گراف را معرفی کرد. راس مرکزی گراف راسی است که بیشینه‌ی کوتاه‌ترین مسیرها از سایر رئوس به آن راس کمترین باشد.*

در نهایت به شبه کد و مثال دیداری زیر برای الگوریتم فلوید-وارشال دقّت کنید.






k = 1:
2 4 1 −2 3
k = 2:
4 −1 2 4 1
4 −1 2 4 1 −2 3
k = 3:
1 −2 3 2 4
2 4 1 −2 3 2 4


k = 4:
3 2 4 −1 2
3 2 4 −1 2 4 1
1 −2 3 2 4 −1 2

```
# Assume a function edgeCost(i,j) which returns the cost of the edge from i to j
#  (infinity if there is none).
#    Also assume that n is the number of vertices and edgeCost(i,i)=0
#

int path[][];
# A 2-dimensional matrix. At each step in the algorithm, path[i][j] is the shortest path
#  from i to j using intermediate vertices (1..k-1).  Each path[i][j] is initialized to
#   edgeCost(i,j) or infinity if there is no edge between i and j.

def FloydWarshall():
    for k in range(1, n):
       for (i, j) in {1,...,n} ^ 2:
          path[i][j] = min(path[i][j], path[i][k] + path[k][j]);
```