آرایه dp را تعریف می‌کنیم به اینگونه که $dp[i]$ برابر کمترین تعداد اسکناس لازم برای پرداخت مقدار i باشد. $dp[0]$ برابر صفر است و سایر خانه‌های آرایه را برابر ∞ قرار می‌دهیم. به ازای هر i از ۱ تا n، مینیمم $1 + dp[i - d_j]$ را به ازای همه‌ی jهای از ۱ تا m به طوری که $i \geq d_j$ باشد حساب می‌کنیم و $dp[i]$ را برابر این مقدار مینیمم قرار می‌دهیم. به شبه کد زیر توجه کنید.

```
for i in 1:n:
    for j in 1:m:
        if i ≥ dj and dp[i] > 1+dp[i-dj]
            dp[i] = 1 + dp[i - dj]
```

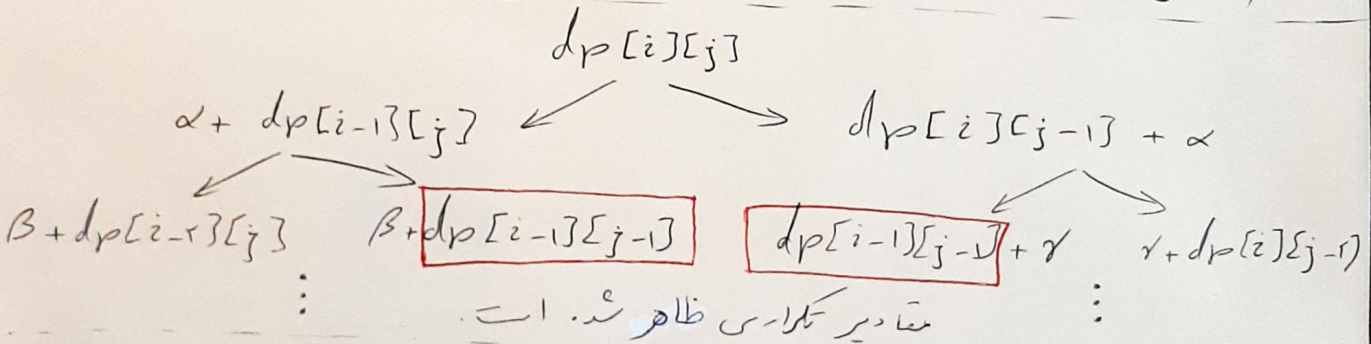
پیچیدگی زمانی این الگوریتم $O(mn)$ است.

۲- $dp[i][j]$ را تعریف می‌کنیم حداکثر تعداد سکه‌ای که تا خانه‌ی سطر i و ستون j می‌توان جمع کرد. همچنین $V[i][j]$ برابر وجود یا عدم وجود سکه در خانه‌ سطر i و ستون j است.

با توجه به حرکات مجاز در مسئله، مهره‌ای که در خانه‌ سطر i و ستون j قرار داشته باشد، یا از خانه‌ بالایی آن (یعنی سطر $i-1$ و ستون j) و یا از خانه‌ سمت چپ آن (یعنی سطر i و ستون $j-1$) به خانه‌ فعلی آمده است. پس می‌توان گفت:

$$dp[i][j] = V[i][j] + \max\{dp[i][j-1], dp[i-1][j]\}$$

$$dp[1][1] = V[1][1]$$



برای پیاده‌سازی، تمامی مقادیر در آرایه‌ dp (به جز $dp[1][1]$) را برابر -1 قرار می‌دهیم و در هر مرحله در صورتی که مقدار آن -1 ماند آن را محاسبه می‌کنیم.

۳- $dp[i][j]$ را تعریف می‌کنیم کمترین هزینه برای رسیدن به v_i با حداکثر فاصله j از پایگاه قبلی. $dp[1][i]$ به ازای هر i برابر صفر است. به ازای i از ۱ تا n و j از ۰ تا D دو حالت را در نظر می‌گیریم:

اگر $j + d(v_i, v_{i+1}) > D$ یا $j < d(v_i, v_{i+1})$ باشد $dp[i][j]$ برابر می‌شود با $dp[i-1][D] + c(v_i)$.

در غیر این صورت $dp[i][j]$ برابر با کمینه $dp[i-1][D] + c(v_i)$ و $dp[i-1][j - d(v_i)]$.

$dp[n][D]$ پاسخ مسئله است. پیچیدگی الگوریتم برابر است با $O(nD)$.

i	C_i^1	C_i^2	C_i^3	C_i^4	C_i^5	Shortest Path Arcs
1	0	0	0	0	0	بدون گره، از هیچ یالی در راه نیست.
2	4	4	4	4	4	(2) $\xrightarrow{4}$ (1)
3	5	5	5	5	5	(3) $\xrightarrow{5}$ (1)
4	∞	7	7	7	7	(4) $\xrightarrow{3}$ (2) $\xrightarrow{4}$ (1)
5	∞	14	13	12	12	(5) $\xrightarrow{2}$ (6) $\xrightarrow{3}$ (4) $\xrightarrow{3}$ (2) $\xrightarrow{4}$ (1)
6	∞	14	10	10	10	(6) $\xrightarrow{3}$ (4) $\xrightarrow{3}$ (2) $\xrightarrow{4}$ (1)
7	∞	∞	16	12	12	(7) $\xrightarrow{2}$ (6) $\xrightarrow{3}$ (4) $\xrightarrow{3}$ (2) $\xrightarrow{4}$ (1)

C_j^i برابر است با طول کوتاه‌ترین مسیر از j به ۱ با گذر از حداکثر i یال.

$$C_j^i = \min\left\{ C_j^{i-1}, \min_{(k,j)\in E} \{ C_k^{i-1} + l_{kj} \} \right\}$$

j اسم راس می‌باشد.

۵- $dp[i][j]$ را تعریف می کنیم کم ترین هزینه ای که سیاوه عماد برای رسیدن از ماجاجی i به ماجاجی j (با واسطه یا بدون واسطه)

واضح است که $dp[i][i]$ به ازای هر i از ۱ تا n برابر صفر است.

طبق فرض C_{ij} هزینه انتقال سیاوه عماد از ماجاجی i به ماجاجی j (بدون واسطه) است.

$$dp[i][j] = \min_{k=i,i+1,\dots,j} \{ dp[i][k] + C_{kj} \}$$

آخرین ماجاجی که سیاوه عماد را به ماجاجی j تحویل داد است.

$dp[1][n]$ پاسخ مسئله است و پیچیدگی زمانی محاسبه آن $O(n^3)$ می باشد.

۶- dp[i][j] را تعریف می‌کنیم کوتاه‌ترین مسیر از راس i به j.
کوتاه‌ترین مسیر از یک راس به خودش برابر صفر است. همچنین اگر از یک
راس به راس دیگر یال نداشته باشیم مقدار ∞ را قرار می‌دهیم.
در گام اول dp[i][j] را با توجه به وزن یال‌های گراف ورودی
مقداردهی می‌کنیم. در گام بعد آرایه را به ازای هر راس آپدیت
می‌کنیم. به این شکل که در هر مرحله یک راس را به عنوان راس میانی
در نظر می‌گیریم و همهٔ زوجی را که راس انتخاب شده راس میانی در
کوتاه‌ترین مسیرشان است را آپدیت می‌کنیم. دو حالت زیر را برای
راس در نظر می‌گیریم:

اگر k راس میانی برای راس‌های i و j نباشد، مقدارش
آپدیت نمی‌شود.
در غیر اینصورت اگر dp[i][j] > dp[i][k]+dp[k][j] باشد
مقدار dp[i][j] را برابر با dp[i][k]+dp[k][j] قرار می‌دهیم. به شبه کد
زیر توجه کنید.

```
for k in [1:n]:
    for i in [1:n]:
        for j in [1:n]:
            if dp[i][k] != ∞ & dp[k][j] != ∞):
                dp[i][j] = (dp[i][j] <= dp[i][k]+dp[k][j])*dp[i][j] + (dp[i][j] > dp[i][k]+dp[k][j])*(dp[i][k]+dp[k][j])
```