طراحی و تحلیل الگوریتم

استاد:

دکتر زاهد رحمتی

تدریسیاران:

اشکان ودادی

داریوش کاظمی

ترم دوم ۱۴۰۰

جلسه چهارم

برنامه نویسی پویا (DP) – قسمت دوم

روش DP:

در روش پویا ابتدا نمونه‌های کوچکتر را حل کرده و نتایج را **ذخیره** می‌کنیم. بعدا هر زمانی به آن‌ها نیاز شد به جای دوباره حساب کردن، از داده ذخیره شده استفاده می‌کنیم.

- در DP از جدول یا آرایه استفاده میشود.
- مشابه تقسیم و حل

1- پیدا کردن رابطه بازگشتی
2- حل نمونه از مسئله به شیوه پایین به بالا با حل نمونه‌های کوچکتر (یا از پایین به بالا)

هر مسئله بهینه‌سازی را نمی‌توان با استفاده از برنامه نویسی پویا حل کرد. اصل بهینگی باید در مسئله صدق کند.
گفته می‌شود اصل بهینگی در یک مسئله صدق می‌کند اگر یک حل بهینه برای نمونه‌ای مسئله، همواره حاوی حل بهینه برای همه زیرنمونه‌ها باشد.

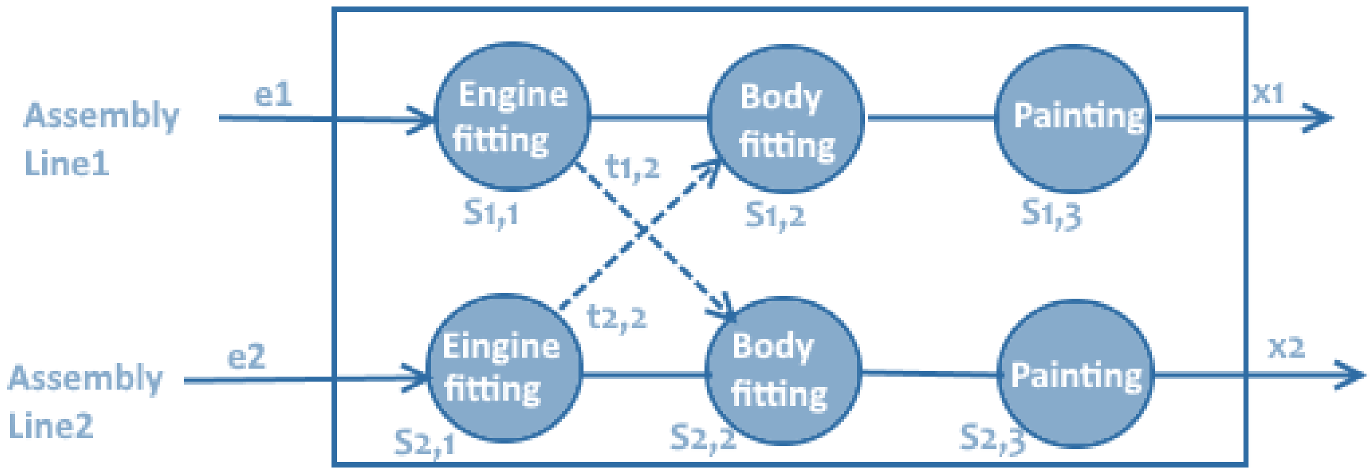
روش پیاده سازی DP:

مرحله اول: پیدا کردن رابطه بازگشتی برای مسئله
مرحله دوم: مشخص کردن ابعاد و اندازه جدول (آرایه، لیست)
مرحله سوم: تعریف هر خانه جدول با کمک رابطه بازگشتی
مرحله چهارم: مقداردهی اولیه و شرایط مرزی
مرحله پنجم: نحوه بروزرسانی خانه‌ها
مرحله ششم: مشخص کردن جواب‌ها

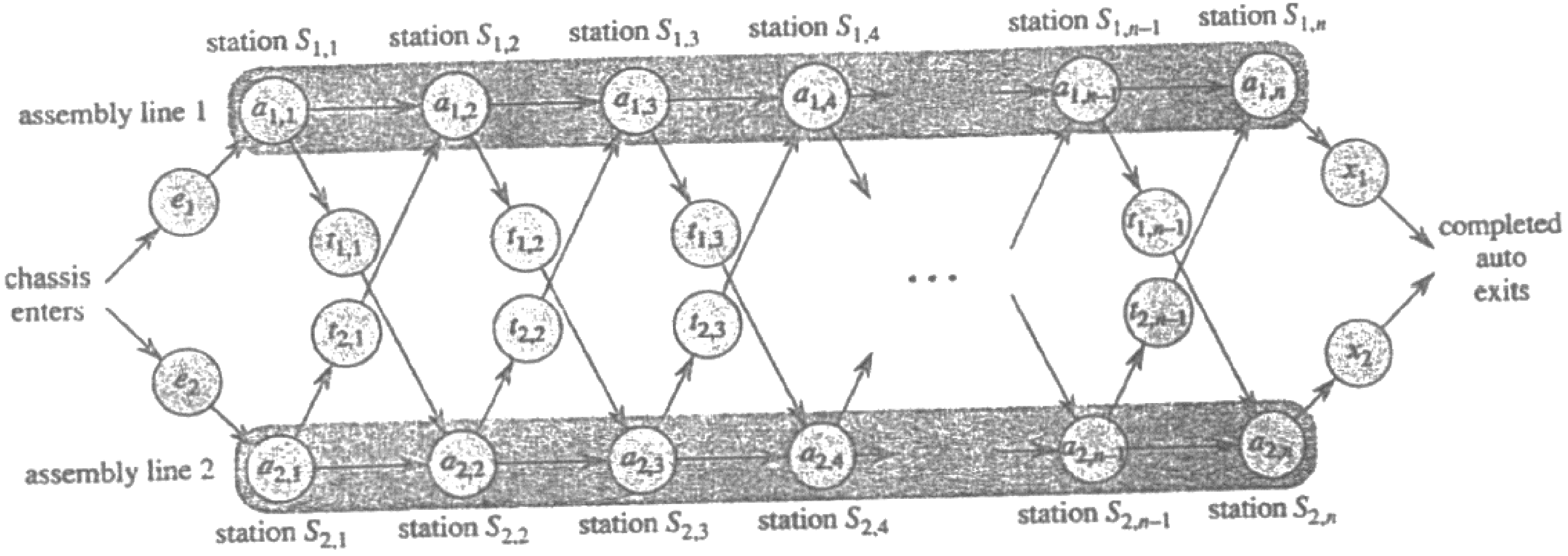
مسئله خط تولید (زمان بندی خطوط مونتاژ):

یک شرکت تولید ماشین دو مسیر تولید ماشین دارد که هرکدام n ایستگاه دارد. هر ایستگاه را با S_{ij} نمایش میدهیم که i خط مسیر مونتاژ را نشان میدهد (1 یا 2) و j شماره ایستگاه در آن خط را نشان می‌دهد. زمان مونتاژ در ایستگاه S_{ij} را با a_{ij} نشان می‌دهیم. زمان انتقال از یک ایستگاه به یک ایستگاه دیگر درون یک خط ناچیز است ولی زمان انتقال به یک خط دیگر مقدار t_{ij} تعریف میکنیم. مقدار e1 و e2 زمان ورود است و مقدار x1 و x2 زمان خروج است.

هدف مسئله تولید ماشین در سریع ترین زمان ممکن است.



مسئله خط تولید (زمان بندی خطوط مونتاژ):



$$f_1[j] = \begin{cases} e_1 + a_{1,1} & \text{if } j = 1\,, \\ \min(f_1[j-1] + a_{1,j},\ f_2[j-1] + t_{2,j-1} + a_{1,j}) & \text{if } j \geq 2 \end{cases}$$

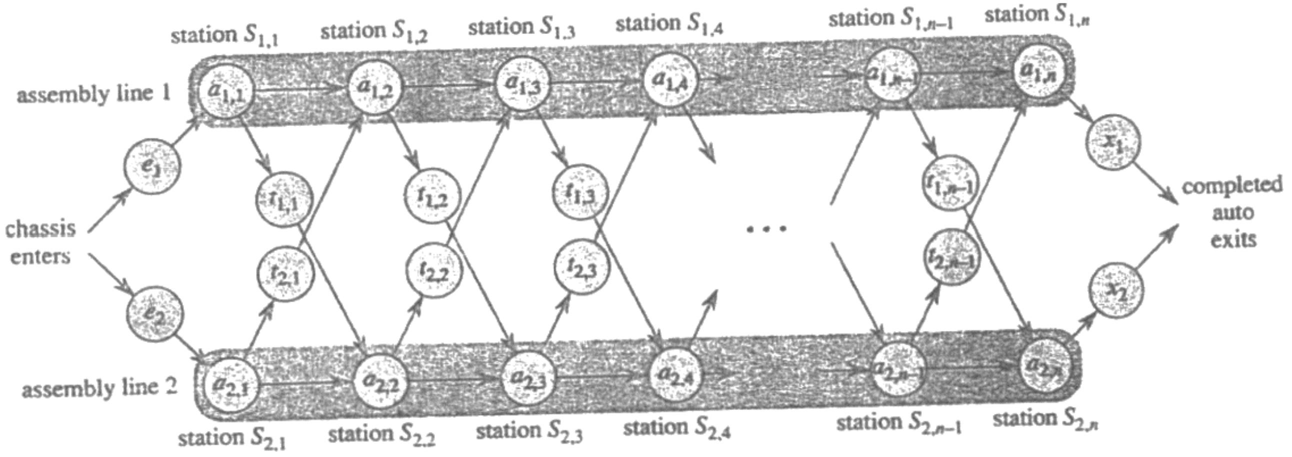
$$f_2[j] = \begin{cases} e_2 + a_{2,1} & \text{if } j = 1\,, \\ \min(f_2[j-1] + a_{2,j},\ f_1[j-1] + t_{1,j-1} + a_{2,j}) & \text{if } j \geq 2 \end{cases}$$

مسئله خط تولید (زمان بندی خطوط مونتاژ):

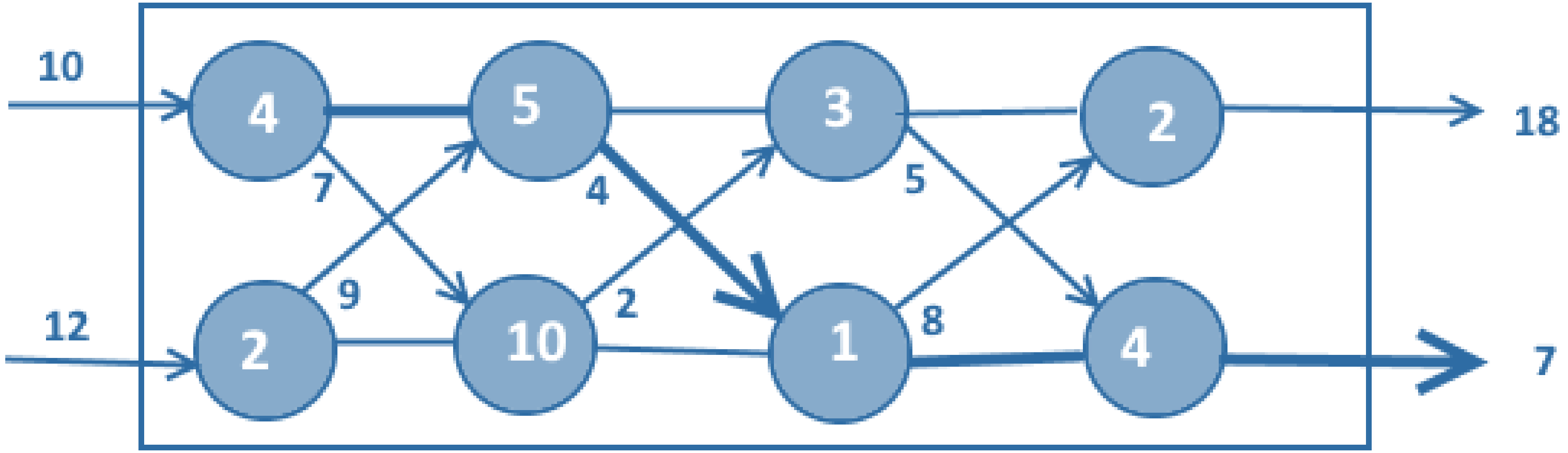
$$f_1[j] = \begin{cases} e_1 + a_{1,1} & \text{if } j = 1\,, \\ \min(f_1[j-1] + a_{1,j},\ f_2[j-1] + t_{2,j-1} + a_{1,j}) & \text{if } j \geq 2 \end{cases}$$

$$f^* = \min\ (f_1\ [n] + x_1\ ,\ f_2[n]\ +\ x_2)$$

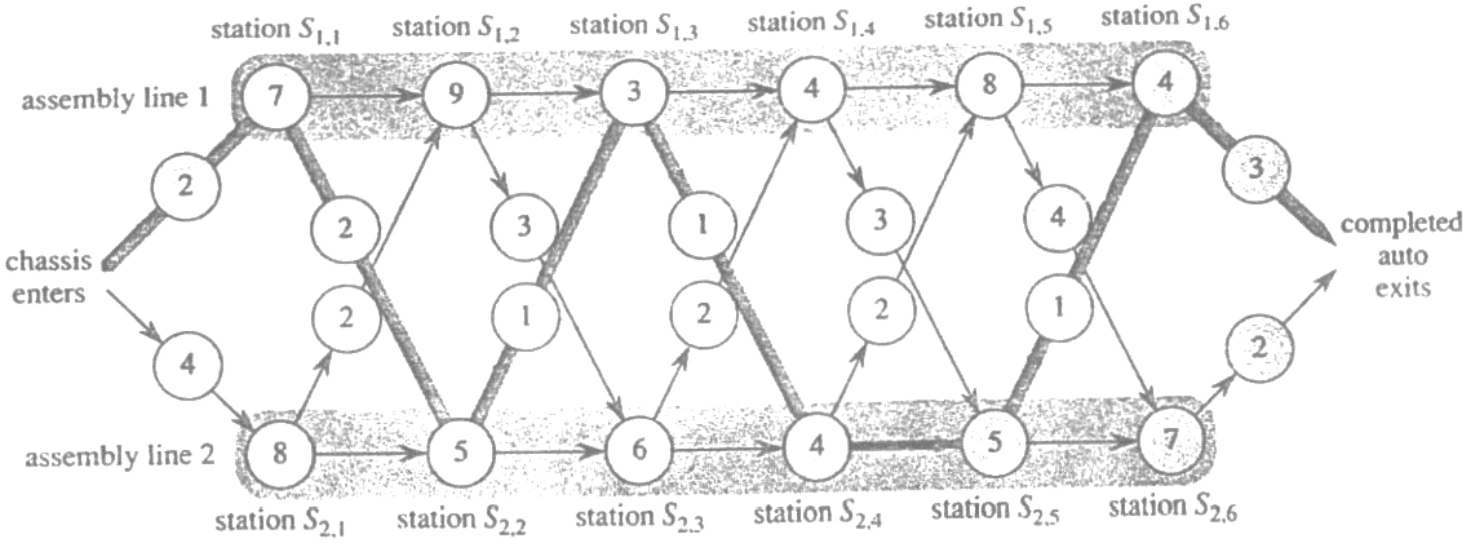
$$f_2[j] = \begin{cases} e_2 + a_{2,1} & \text{if } j = 1\,, \\ \min(f_2[j-1] + a_{2,j},\ f_1[j-1] + t_{1,j-1} + a_{2,j}) & \text{if } j \geq 2 \end{cases}$$



مسئله خط تولید (مثال):



مسئله خط تولید (مثال):



(a)

j	1	2	3	4	5	6
$f_1[j]$	9	18	20	24	32	35
$f_2[j]$	12	16	22	25	30	37

$f^* = 38$

j	2	3	4	5	6
$l_1[j]$	1	2	1	1	2
$l_2[j]$	1	2	1	2	2

$l^* = 1$

(b)

شکل ۱۵.۲ (a) نمونه‌ای از مسئله خطوط مونتاژ با هزینه‌های مشخص شده e_i، $a_{i,j}$، $t_{i,j}$ و x_i. مسیر پر رنگ سریعترین مسیر در کارخانه را نشان می‌دهد. (b) مقادیر $f_i[j]$، f^*، $l_i[j]$ و l^* برای قسمت (a).

مسئله خط تولید (کد):

FASTEST-WAY(a, t, e, x, n)

$f_1[1] \leftarrow e_1 + a_{1,1}$
$f_2[1] \leftarrow e_2 + a_{2,1}$
for $j \leftarrow 2$ **to** n
  do if $f_1[j-1] + a_{1,j} \leq f_2[j-1] + t_{2,j-1} + a_{1,j}$
    then $f_1[j] \leftarrow f_1[j-1] + a_{1,j}$
      $l_1[j] \leftarrow 1$
    else $f_1[j] \leftarrow f_2[j-1] + t_{2,j-1} + a_{1,j}$
      $l_1[j] \leftarrow 2$
  if $f_2[j-1] + a_{2,j} \leq f_1[j-1] + t_{1,j-1} + a_{2,j}$
    then $f_2[j] \leftarrow f_2[j-1] + a_{2,j}$
      $l_2[j] \leftarrow 2$
    else $f_2[j] \leftarrow f_1[j-1] + t_{1,j-1} + a_{2,j}$
      $l_2[j] \leftarrow 1$
if $f_1[n] + x_1 \leq f_2[n] + x_2$
  then $f^* = f_1[n] + x_1$
    $l^* = 1$
  else $f^* = f_2[n] + x_2$
    $l^* = 2$

Time + Space: O(n)

Edit Distance Code and Question

دو رشته STR1 و STR2 به شما داده می‌شود. 3 عملیات زیر داده شده است. حداقل تغییراتی که لازم است تا با استفاده از عملیات‌های زیر رشته اول را به رشته دوم تبدیل کند.

1. اضافه کردن حرف
2. حذف حرف
3. جایگزین کردن حرف

هزینه همه‌ی عملیات‌ها یکسان است.

Edit Distance Code and Question

مثال)

Input: str1 = "geek", str2 = "gesek"
Output: 1
We can convert str1 into str2 by inserting a 's'.

Input: str1 = "cat", str2 = "cut"
Output: 1
We can convert str1 into str2 by replacing 'a' with 'u'.

Input: str1 = "sunday", str2 = "saturday"
Output: 3
Last three and first characters are same.
We basically need to convert "un" to "atur".
This can be done using below three operations. Replace 'n' with 'r', insert t, insert a

Edit Distance Code and Question

دو رشته STR1 و STR2 به شما داده می‌شود:
اضافه کردن حرف + حذف حرف + جایگزین کردن حرف

آخرین حرف دو رشته را مقایسه میکنیم. بعد به صورت بازگشتی حرف به حرف عقب میرویم.
الگوریتم:

1. اگر m=0 بود،n را برگدان و اگر n = 0بود m را برگردان.
2. اگر آخرین حرف دو رشته برابر بود، اتفاقی نمی افتد و مقدار dp[m][n] برابر با مقدار dp[m-1][n-1] میشود.
3. اگر آخرین حرف دو رشته برابر نبود، هر سه حالت ممکن را درنظر میگیرم و برنامه رو بازگشتی اعمال میکنیم و مینیوم سه حالت ممکن را حساب میکنیم. (dp[m][n]={1,2,3})
    1. درج: بازگشتی برای m و n-1
    2. حذف: بازگشتی برای m-1 و n
    3. جایگزینی: بازگشتی برای m-1 و n-1

Edit Distance Code and Question

1. اگر m=0 بود، n را برگردان و اگر n = 0 بود m را برگردان.
2. اگر آخرین حرف دو رشته برابر نبود، هر سه حالت ممکن را درنظر میگیرم و برنامه رو بازگشتی اعمال میکنیم و مینیوم سه حالت ممکن را حساب میکنیم.
   1. درج: بازگشتی برای m و n−1
   2. حذف: بازگشتی برای m−1 و n
   3. جایگزینی: بازگشتی برای m−1 و n−1

```
int dp[m + 1][n + 1];
for (int i = 0; i <= m; i++) {
    for (int j = 0; j <= n; j++) {
        if (i == 0)
            dp[i][j] = j; // Min. operations = j
        else if (j == 0)
            dp[i][j] = i; // Min. operations = i
        else if (str1[i - 1] == str2[j - 1])
            dp[i][j] = dp[i - 1][j - 1];
        else
            dp[i][j] = 1 + min(dp[i][j - 1], // Insert
                                dp[i - 1][j], // Remove
                                dp[i - 1][j - 1]); // Replace

    }
}
return dp[m][n];
```

Edit Distance Code and Question

1. اگر m=0 بود، n را برگردان و اگر n = 0 بود m را برگردان.
2. اگر آخرین حرف دو رشته برابر نبود، هر سه حالت ممکن را درنظر میگیرم و برنامه رو بازگشتی اعمال میکنیم و مینیوم سه حالت ممکن را حساب میکنیم.
   1. درج: بازگشتی برای m و n-1
   2. حذف: بازگشتی برای m-1 و n
   3. جایگزینی: بازگشتی برای m-1 و n-1

```
int dp[m + 1][n + 1];
for (int i = 0; i <= m; i++) {
    for (int j = 0; j <= n; j++) {
        if (i == 0)
            dp[i][j] = j; // Min. operations = j
        else if (j == 0)
            dp[i][j] = i; // Min. operations = i
        else if (str1[i - 1] == str2[j - 1])
            dp[i][j] = dp[i - 1][j - 1];
        else
            dp[i][j] = 1 + min(dp[i][j - 1], // Insert
                               dp[i - 1][j], // Remove
                               dp[i - 1][j - 1]); // Replace

    }
}
return dp[m][n];
```

پیچیدگی زمانی: O(mn)

Edit Distance Code and Question

مثال) CART را به MARCH میخواهیم تبدیل کنیم.

- تبدیل C به M
- تبدیل T به C
- اضافه کردن h

editDist	Ø	M	A	R	C	H
Ø						
C						
A						
R						
T						

Edit Distance Code and Question

مثال) CART را به MARCH میخواهیم تبدیل کنیم.

- تبدیل C به M
- تبدیل T به C
- اضافه کردن h

editDist	Ø	M	A	R	C	H
Ø	0	1	2	3	4	5
C	1					
A	2					
R	3					
T	4					

Edit Distance Code and Question

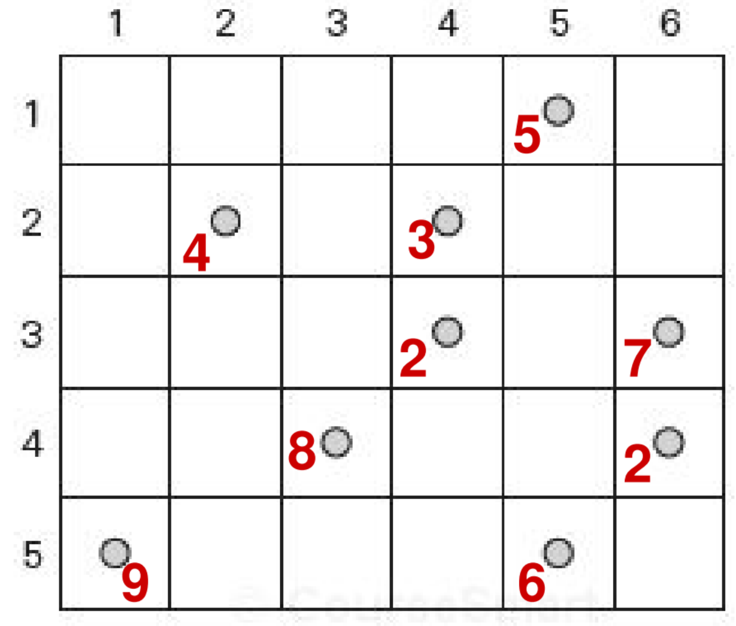
مثال) CART را به MARCH میخواهیم تبدیل کنیم.

- تبدیل C به M
- تبدیل T به C
- اضافه کردن h

editDist	Ø	M	A	R	C	H
Ø	0	1	2	3	4	5
C	1	1	2	3	3	4
A	2	2	1	2	3	4
R	3	3	2	1	2	3
T	4	4	3	2	2	3

ربات سکه جمع‌کن:

تعدادی از سکه‌ها در یک جدول n در m قرار داده شده است. در هر خانه جدول ماکسیموم یک سکه قرار دارد. یک ربات که در بالا چپ جدول قرار دارد، می‌خواهد بیشترین تعداد سکه ممکن را جمع کند و به خانه پایین راست برسد. (ربات فقط در دو جهت پایین راست میتواند حرکت کند)



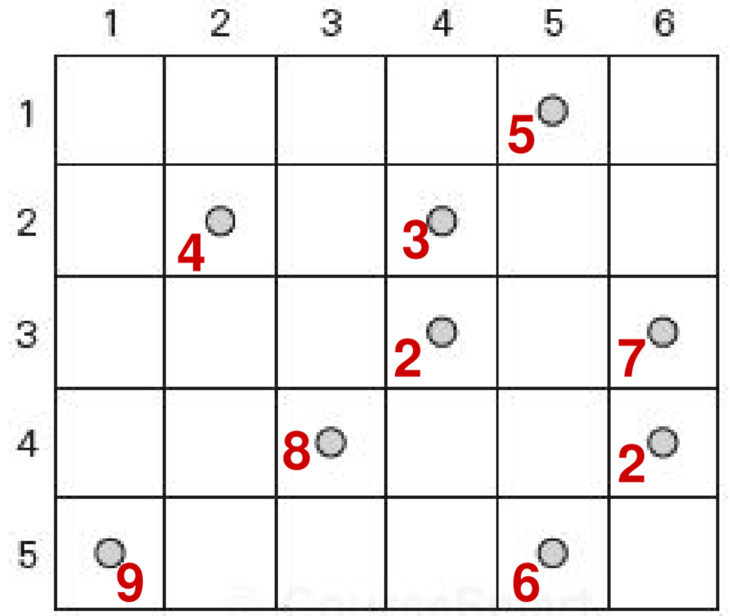
ربات سکه جمع‌کن:

تعدادی از سکه‌ها در یک جدول n در m قرار داده شده است. در هر خانه جدول ماکسیموم یک سکه قرار دارد. یک ربات که در بالا چپ جدول قرار دارد، می‌خواهد بیشترین تعداد سکه ممکن را جمع کند و به خانه پایین راست برسد. (ربات فقط در دو جهت پایین راست میتواند حرکت کند)

F[i, j] = max{F[i − 1, j], F[i, j − 1]} + coins[i,j]

F[0, j] = 0 for 1≤ j ≤ m

F[i, 0] = 0 for 1≤ i ≤ n



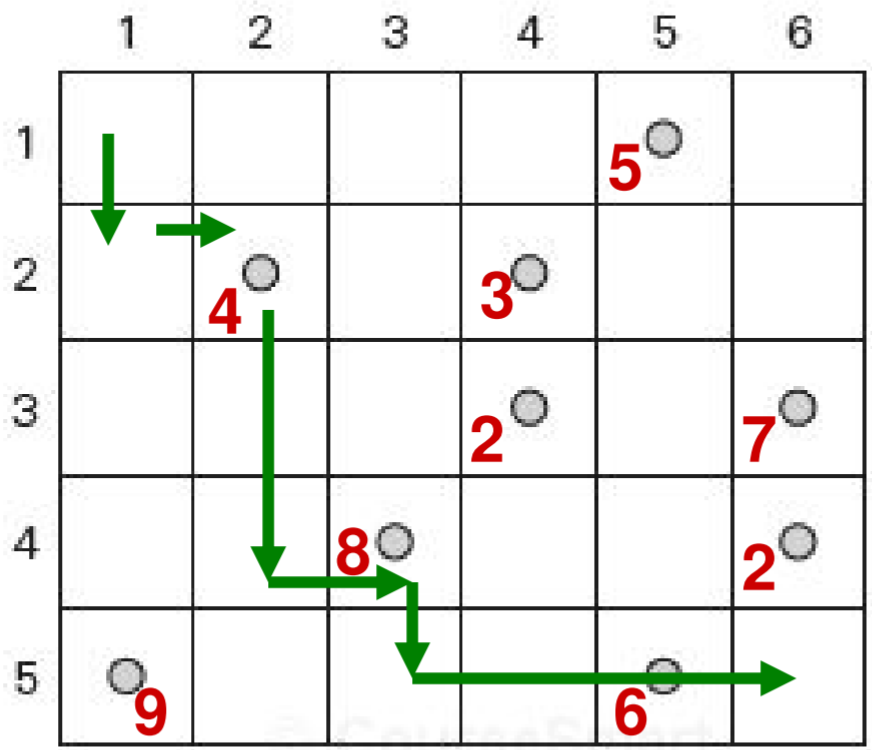
ربات سکه جمع‌کن (مثال):

$$F[i, j] = \max\{F[i-1, j], F[i, j-1]\} + coins[i,j]$$

$$F[0, j] = 0 \text{ for } 1 \le j \le m$$

$$F[i, 0] = 0 \text{ for } 1 \le i \le n$$

	1	2	3	4	5	6
1	0	0	0	0	5	5
2	0	4	4	7	7	7
3	0	4	4	9	9	16
4	0	4	12	12	12	18
5	9	9	12	12	18	18



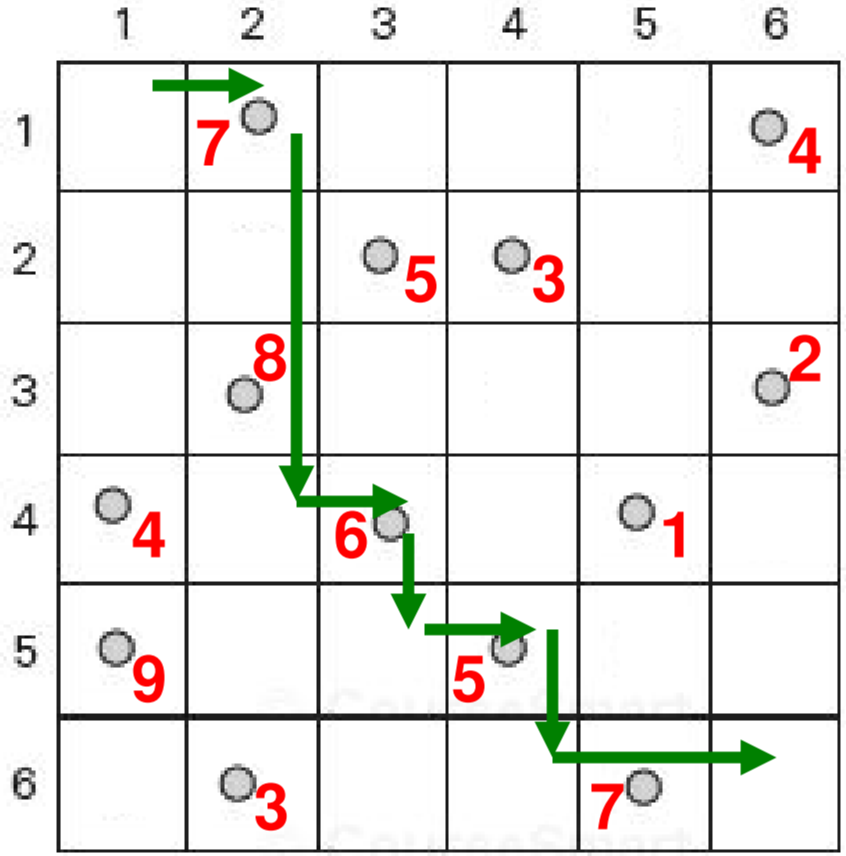
ربات سکه جمع‌کن (مثال):

$$F[i, j] = \max\{F[i-1, j], F[i, j-1]\} + coins[i,j]$$

$$F[0, j] = 0 \text{ for } 1 \leq j \leq m$$

$$F[i, 0] = 0 \text{ for } 1 \leq i \leq n$$

	1	2	3	4	5	6
1	0	7	7	7	7	11
2	0	7	12	15	15	15
3	0	15	15	15	15	17
4	4	15	21	21	22	22
5	13	15	21	26	26	26
6	13	18	21	26	33	33



ربات سکه جمع‌کن

Recurrence

$$F(i, j) = \max\{F(i-1, j),\ F(i, j-1)\} + c_{ij} \quad \text{for } 1 \le i \le n,\ \ 1 \le j \le m$$

$$F(0, j) = 0 \ \text{ for } 1 \le j \le m \quad \text{and} \quad F(i, 0) = 0 \ \text{ for } 1 \le i \le n,$$

where c_{ij} = 1 if there is a coin in cell (i, j) and c_{ij} = 0 otherwise.

ALGORITHM *RobotCoinCollection*$(C[1..n, 1..m])$

//Applies dynamic programming to compute the largest number of
//coins a robot can collect on an $n \times m$ board by starting at (1, 1)
//and moving right and down from upper left to down right corner
//Input: Matrix $C[1..n, 1..m]$ whose elements are equal to 1 and 0
//for cells with and without a coin, respectively
//Output: Largest number of coins the robot can bring to cell (n, m)
$F[1,1] \leftarrow C[1,1]$; **for** $j \leftarrow 2$ **to** m **do** $F[1, j] \leftarrow F[1, j-1] + C[1, j]$
for $i \leftarrow 2$ **to** n **do**
    $F[i, 1] \leftarrow F[i-1, 1] + C[i, 1]$
    for $j \leftarrow 2$ **to** m **do**
        $F[i, j] \leftarrow \max(F[i-1, j], F[i, j-1]) + C[i, j]$
return $F[n, m]$

Time Complexity: Θ(nm)
Space Complexity: Θ(nm)

خسته نباشید!

داریوش کاظمی – اشکان ودادی