دانشکده مهندسی کامپیوتر

جزوه درس

ساختمان‌های داده

استاد درس: سید صالح اعتمادی

پاییز ۱۳۹۸

جلسه ۱۱

برنامه ریزی پویا

سهند نظرزاده - ۱۳۹۸/۸/۵

۱.۱۱ مقدمه

برنامه ریزی پویا [۱] (این روش با نام برنامه‌نویسی پویا نیز شناخته می‌شود . منظور از برنامه‌نویسی، روشی خطی برای حل مسئله است، نه نوشتن کد در رایانه) همانند الگوریتم تقسیم و حل [۲] ، مسئله با ترکیب زیرمسئله ها حل می شود با این تفاوت که در الگوریتم تقسیم و حل زیر مسئله ها مستقل از هم بودند اما در برنامه ریزی پویا زیر مسئله ها کاملا مستقل نیستند و زیر مسئله ها ، زیر مسئله های مشترکی (زیر مسئله های هم پوشان) دارند یا زیرمسئله های یکسانی از مسیر های مختلفی حاصل می شوند که با این حال جوابشان یکسان است و در هر مرحله نیازی به محاسبه ی مجدد یک مسئله ی تکراری نیست و فقط یک بار آن را حل کرده و ذخیره می کنیم . معمولا برای حل مسائل بهینه سازی و شمارشی از برنامه ریزی پویا استفاده می شود که مسئله جواب یکتا ندارند اما فقط یک جواب یا اندازه ی بیشینه بودن یا کمینه بودن اهمیت دارد . قسمت مهم حل مسائل با این الگوریتم ترتیب حل زیر مسئله ها است که باید به شکلی صورت گیرد که همه زیر مسئله های یک مسئله قبل از آن حل شده باشند . چارچوب استانداردی برای حل عمومی مسائل برنامه ریزی پویا وجود ندارد . بلکه برنامه ریزی پویا فقط یک روش برخورد کلی (یا یک دید کلی) جهت حل دسته‌ای از مسائل ارائه می‌دهد.

[۱] Programming Dynamic
[۲] Conquer and Divide

۲.۱۱ مسائل بررسی شده

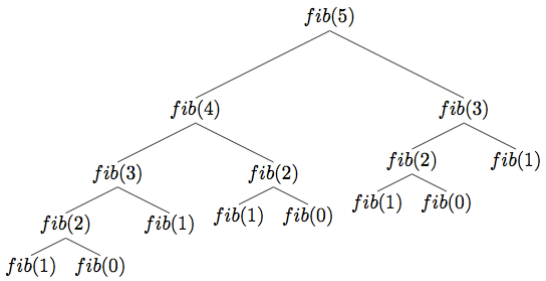
Fibonacci ۱.۲.۱۱

مسئله. الگوریتمی طراحی کنید که با الگوریتم برنامه ریزی پویا n امین عدد دنباله ی فیبوناچی را محاسبه کند.

حل. با توجه به تعریف دنباله فیبوناچی داریم

$\mathrm{Fib(i)} = \mathrm{Fib(i - 1)} + \mathrm{Fib(i - 2)}$, $\mathrm{Fib(0)} = 0$, $\mathrm{Fib(1)} = 1$

برای $\mathrm{n} = 5$ زیر مسئله ها به ترتیب شکل ۱.۱۱ صدا زده می شود



شکل ۱.۱۱: درخت زیر مسئله ها

با توجه به تکرار زیر مسئله ها (به عنوان مثال زیر مسئله $\mathrm{Fib}(2)$ سه بار تکرار شده است) الگوریتم برنامه ریزی پویا نسبت به الگوریتم تقسیم و حل بهینه تر است به همین دلیل هر زیر مسئله را یک بار حل و آن را ذخیره می کنیم. شبه کد ۱ آن به صورت زیر است

input : n

Declare Fib as an array

$Fib_0 = 0$

$Fib_1 = 1$

for $i \leftarrow 0$ ***to*** $n - 1$ ***by*** 1 **do**
| $Fib_i = Fib_{i-1} + Fib_{i-2}$

end

output: Fib_{n-1}

Algorithm 1: DP solution

Change Money ۲.۲.۱۱

مسئله. الگوریتمی طراحی کنید که با کمترین تعداد از سکه های $c_1, c_2, c_3, ..., c_m$ سنتی، money سنت را خرد کند.

حل.

الگوریتم حریصانه

با توجه به مثال (شکل ۲.۱۱) زیر جواب بدست آمده از الگوریتم حریصانه درست نیست زیرا برای خرد کردن ۴۰ سنت با سکه های ۵، ۱۰، ۲۰ و ۲۵ سنتی ، دو سکه ی ۲۰ سنتی بهینه تر از سکه های ۵ ، ۱۰ و ۲۵ سنتی است به همین دلیل الگوریتم حریصانه برای این مسئله کار آمد نیست.

$$40 \text{ cents} = \underbrace{25 + 10 + 5}_{\text{Greedy}} \quad = \quad \underbrace{20 + 20}_{\text{Optimal}}$$

is not

شکل ۲.۱۱: جواب حریصانه

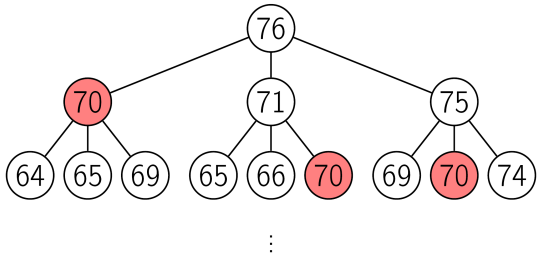
الگوریتم بازگشتی

ابتدا باید رابطه ی بازگشتی مسئله را با کمک گرفتن از الگوریتم تقسیم و حل پیدا کنیم. برای این کار باید محیط مسئله اصلی را به نحوی کوچک تر کنیم تا به زیر مسئله هایی مشابه با مسئله ی اصلی برسیم. در این مسئله بر روی انتخاب شدن یا نشدن سکه i ام حالت بندی می کنیم.

در صورت انتخاب شدن سکه ی i ام که i = 1, 2, 3, ... , m محیط مسئله از money به money - c_i تبدیل می شود و در صورت انتخاب نشدن سکه ی i ام در این مرحله به سراغ سکه های بعدی می رویم و بر روی آن ها حالت بندی می کنیم و در آخر زیر مسئله ای که با کمترین تعداد سکه حل شده است را به عنوان بهینه ترین زیر مسئله انتخاب می کنیم . این کار را تا زمانی که محیط مسئله به اندازه کافی ساده نشده ادامه می دهیم. رابطه بازگشتی به صورت زیر می شود.

$$MinNumCions(money) = Min \begin{cases} MinNumCions(money - c_1) + 1 \\ MinNumCions(money - c_2) + 1 \\ MinNumCions(money - c_3) + 1 \\ ... \\ MinNumCions(money - c_m) + 1 \end{cases} \quad (۱.۱۱)$$

طبق شکل ۳.۱۱ برای $\text{money} = 76$ و با سکه های $c_1 = 1, c_2 = 5, c_3 = 6$ سنتی، به دلیل محاسبات تکراری (در همین ابتدای کار زیر مسئله ی $\text{money} = 70$ سه بار تکرار شده است) الگوریتم بازگشتی نیز از لحاظ زمانی کار آمد نیست.



شکل ۳.۱۱: درخت زیر مسئله ها

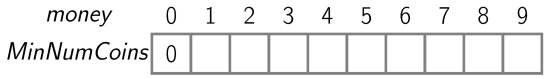
شبه کد ۲ آن به صورت زیر است.

```
RecursiveChange(money, coin) if money = 0 then
    return 0
end
MinNumCoins ← ∞
for i ← 0 to |cions| by 1 do
    if coin_i ≤ money then
        NumCoins ← RecursiveChange money − coin_i, coin
        if NumCoins + 1 < MinNumCoins then
            MinNumCoins ← NumCoins + 1
        end
    end
end
return MinNumCoins
```

Algorithm 2: Recursive solution

الگوریتم برنامه ریزی پویا

ابتدا باید رابطه بین مسئله و زیر مسئله ها را پیدا کنیم که همان رابطه ی بازگشتی ۱.۱۱ است.
در ادامه باید زیر مسئله ها را به ترتیبی حل و ذخیره کنیم که قبل از حل هر مسئله، زیر مسئله های آن حل و ذخیره شده باشد. طبق رابطه ی بازگشتی برای حل مسئله money = k باید مسائل money = k - c_i که $i = 1, 2, 3, \ldots, m$ قبل از آن حل شده باشد. به همین دلیل از مسئله $money = 1$ شروع کرده و به ترتیب مسائل $money = 2$ و $money = 3$ الی $money = k$ را حل و ذخیره می کنیم. برای ذخیره جواب زیر مسائل ، آرایه ی $MinNumCoins$ و به طول $k + 1$ را تعریف می کنیم. برای راحتی کار جواب مسئله ی $money = 0$ را صفر فرض می کنیم $MinNumCoins_0 = 0$ (صفر سنت پول با صفر سکه خرد می شود).
برای درک بهتر برنامه ریزی پویا ، مسئله ی $money = 9$ و با سکه های $c_1 = 1, c_2 = 5, c_3 = 6$ سنتی را بررسی می کنیم. در ابتدا آرایه $MinNumCoins$ به شکل ۴.۱۱ زیر است.

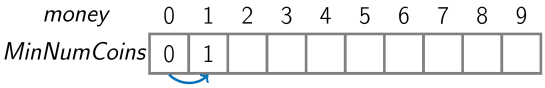


شکل ۴.۱۱: ابتدای کار

برای حل مسئله ی $money = 1$ باید جواب زیر مسئله های $money = 1 - 1 = 0$ ، $money = 1 - 5 = -4$ و $money = 1 - 6 = -5$ را بدانیم. دو زیر مسئله ی آخر به دلیل منفی بودن $(money - c_i < 0)$ ، قابل قبول نیستند. پس جواب این مسئله به صورت زیر می شود.

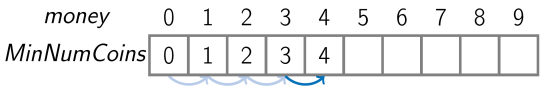
$$MinNumcoins_1 = MinNumCoins_0 + 1$$

و آرایه به شکل ۵.۱۱ زیر تغییر میکند.



شکل ۵.۱۱: مرحله ی اول

مسئله های $money = 2$ ، $money = 3$ و $money = 4$ با روش مشابه ای حل می شوند. و آرایه به شکل ۶.۱۱ زیر تغییر می کند.



شکل ۶.۱۱: مرحله ی چهارم

مسئله ی $money = 5$ با مسائل قبلی کمی متفاوت است. رابطه ی بازگشتی را برای آن می نویسیم تا رابطه آن با زیر مسئله مشخص شود.

$$MinNumCions_5 = min \begin{cases} MinNumCions_{5-1} + 1 = MinNumCoins_4 + 1 \\ MinNumCions_{5-5} + 1 = MinNumCoins_0 + 1 \\ MinNumCions_{5-6} + 1 \end{cases}$$

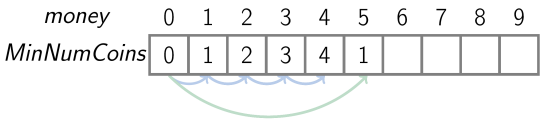
زیر مسئله آخر به دلیل منفی بودن قابل قبول نیست $(money \text{ - } c_i < 0)$.
حال باید با توجه به جواب زیر مسئله ها $MinNumCoins_0 = 0$ و $MinNumCoins_4 = 4$ از بین آن ها بهینه ترین جواب را انتخاب کرد.

$$MinNumCions_5 = min \begin{cases} MinNumCions_4 + 1 = 4 + 1 = 5 \\ MinNumCions_0 + 1 = 0 + 1 = 1 \end{cases}$$

پس جواب این مسئله به صورت زیر می شود.

$$MinNumCoins_5 = \min(5,\, 1) = 1$$

و آرایه به شکل ۷.۱۱ زیر تغییر می کند.



شکل ۷.۱۱: مرحله ی پنجم

مسئله $money = 6$ را با روشی مشابه با مسئله ی $money = 5$ بررسی می کنیم. برای این کار به کمک رابطه ی بازگشتی ، رابطه ی میان مسئله و زیر مسائل را میابیم.

$$MinNumCions_6 = min \begin{cases} MinNumCions_{6-1} + 1 = MinNumCoins_5 + 1 \\ MinNumCions_{6-5} + 1 = MinNumCoins_1 + 1 \\ MinNumCions_{6-6} + 1 = MinNumCoins_0 + 1 \end{cases}$$

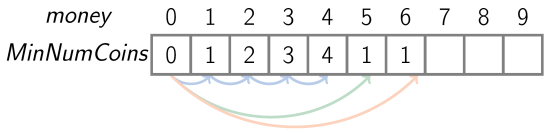
با توجه به جواب زیر مسائل داریم.

$$MinNumCions_6 = min \begin{cases} MinNumCions_5 + 1 = 1 + 1 = 2 \\ MinNumCions_1 + 1 = 1 + 1 = 2 \\ MinNumCions_0 + 1 = 0 + 1 = 1 \end{cases}$$

جواب این مسئله به صورت زیر بدست می آید.

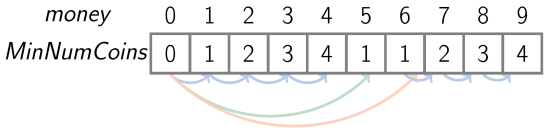
$$MinNumCoins_6 = \min(2,\ 2,\ 1) = 1$$

آرایه به شکل ۸.۱۱ تغییر می کند.



شکل ۸.۱۱: مرحله ی ششم

سایر مسایل با روشی مشابه ای حل می شوند. در آخر آرایه به شکل ۹.۱۱ زیر تبدیل می شود و پاسخ مسئله ی اصلی برابر با $MinNumCoins_{money} = MinNumCoins_9 = 4$ است.



شکل ۹.۱۱: انتهای کار

شبه کد ۳ به در زیر آمده است.

`DynamicChange`($money$, $coin$)

Declare $MinNumCoins$ as an array

$MinNumCoins_0 = 0$

for $i \leftarrow 1$ ***to*** $money$ ***by*** 1 **do**

$MinNumCoins_i = \min(MinNumsCoins_{i-c_1}, MinNumCoins_{i-c_2},$

$MinNumCoins_{i-c_3}, \ldots, MinNumCoins_{i-c_m}) + 1$

end

return $MinNumCoins_{money}$

Algorithm 3: Dynamic solution

Distance Edit ۳.۲.۱۱

مسئله. الگوریتمی طراحی کنید که با کمترین تعداد از عملیات های اضافی کردن یک کاراکتر [۳] ، پاک کردن یک کاراکتر [۴] و تعویض یک کاراکتر با کاراکتری دیگر [۵] رشته ی A را به رشته ی B تبدیل کند.

حل. همانند مسئله ی قبل ابتدا رابطه میان مسئله و زیر مسائل را پیدا می کنیم. برای این کار رو آخرین کاراکتر هر دو رشته حالت بندی می کنیم.

$$\mathrm{A} = a_1, a_2, a_3, \ldots, a_n$$
$$\mathrm{B} = b_1, b_2, b_3, \ldots, b_m$$

بر روی a_n و b_m حالت بندی می کنیم:

حالت اول ($a_n = b_m$)

در این حالت به دلیل این که هر دو کاراکتر برابر هستند، برای تبدیل رشته ی اول به رشته ی دوم کافی است فرض کنیم که دو کاراکتر آخر وجود ندارند و سپس زیر مسئله ای که رشته های آن

$$\mathrm{A} = a_1, a_2, a_3, \ldots, a_{n-1}$$
$$\mathrm{B} = b_1, b_2, b_3, \ldots, b_{m-1}$$

هستند را حل کنیم و سپس کاراکتر ها را در انتها رشته ها قرار می دهیم (برای اضافی کردن کاراکتر ها به انتهای دو رشته نیازی به استفاده از عملگر ها نیست زیرا قبل از تبدیل مسئله به زیر مسئله در انتهای دو رشته وجود داشته اند) تا رشته ی اول به رشته دوم تبدیل شود. به دلیل این که در مسیر تبدیل رشته ی بدست آمده از حل زیر مسئله به رشته ی مسئله اصلی از عملگر ها استفاده نکردیم؛ جواب مسئله همان جواب زیر مسئله است.

به عنوان مثال برای مسئله ای که رشته ی اول آن $\mathrm{A} = \text{"insert"}$ و رشته ی دوم آن $\mathrm{B} = \text{"edit"}$ هستند؛ به دلیل برابر بودن کاراکتر آخر هر دو رشته ($a_6 = \mathrm{t}$ و $b_4 = \mathrm{t}$) کاراکتر ها را از انتهای دو رشته حذف می کنیم و زیر مسئله (کمترین تعداد از عملیات ها برای تبدیل رشته ی $\mathrm{A} = \text{"inser"}$ به $\mathrm{B} = \text{"edi"}$) را حل می کنیم و سپس در کاراکتر ها را در انتهای دو رشته قرار می دهیم. به دلیل این که در مسیر تبدیل جواب زیر به جواب مسئله از عملگر ها استفاده نکردیم؛ جواب مسئله همان جواب زیر مسئله است.

---

[۳] Insertions
[۴] deletions
[۵] subtitutions

حالت دوم ($a_n \neq b_m$)

برای تبدیل کاراکتر a_n به کاراکتر b_n باید از عملگر ها استفاده کنیم. هر عملگر مسئله را به یک زیر مسئله مرتبط می کند.
بنابراین روی عملگر ها حالت بندی می کنیم:

عملگر تعویض کردن یک کاراکتر
برای تبدیل رشته ی اول به رشته ی دوم کافی است فرض کنیم که دو کاراکتر آخر وجود ندارند و سپس زیر مسئله ای که رشته های آن

$$\mathrm{A} = a_1, a_2, a_3, ..., a_{n-1}$$
$$\mathrm{B} = b_1, b_2, b_3, ..., b_{m-1}$$

هستند را حل کنیم و سپس کاراکتر آخر رشته ی اول a_n را با عملگر تعویض و با هزینه ی یک به کاراکتر آخر رشته ی دوم b_m تبدیل می کنیم و در انتهای رشته اول قرار می دهیم و کاراکتر آخر رشته ی دوم را انتهای همان رشته قرار می دهیم (برای اضافی کردن کاراکتر به انتهای رشته ی دوم نیازی به استفاده از عملگر ها نیست زیرا قبل از تبدیل مسئله به زیر مسئله در انتهای آن رشته وجود داشته است) تا رشته ی اول به رشته دوم تبدیل شود. به دلیل این که در مسیر تبدیل رشته ی بدست آمده از حل زیر مسئله به رشته ی مسئله اصلی، یک بار از عملگر تعویض استفاده شده است؛ جواب مسئله اصلی یک واحد بیشتر از زیر مسئله است.
به عنوان مثال برای مسئله ای که رشته ی اول آن $\mathrm{A} = \text{"edit"}$ و رشته ی دوم آن $\mathrm{B} = \text{"distance"}$ هستند؛ کاراکتر های آخر را از انتهای دو رشته حذف می کنیم و زیر مسئله (کمترین تعداد از عملیات ها برای تبدیل رشته ی $\mathrm{A} = \text{"edi"}$ به $\mathrm{B} = \text{"distanc"}$) را حل می کنیم؛ سپس کاراکتر آخر رشته ی اول را با عملگر تعویض و با هزینه یک به کاراکتر آخر رشته ی دوم تبدیل می کنیم و در انتهای رشته ی اول قرار می دهیم و کاراکتر آخر رشته ی دوم را انتهای همان رشته قرار می دهیم (برای اضافی کردن کاراکتر به انتهای رشته ی دوم نیازی به استفاده از عملگر ها نیست زیرا قبل از تبدیل مسئله به زیر مسئله در انتهای آن رشته وجود داشته است) تا رشته ی اول به رشته دوم تبدیل شود.به دلیل این که در مسیر تبدیل رشته ی بدست آمده از حل زیر مسئله به رشته ی مسئله اصلی، یک بار از عملگر تعویض استفاده شده است؛ جواب مسئله اصلی یک واحد بیشتر از زیر مسئله است.

عملگر اضافه کردن یک کاراکتر

برای تبدیل رشته ی اول به رشته ی دوم کافی است فرض کنیم که کاراکتر آخر رشته دوم وجود ندارد و زیر مسئله ای که رشته های آن

$\mathrm{A} = a_1, a_2, a_3, ..., a_n$

$\mathrm{B} = b_1, b_2, b_3, ..., b_{m-1}$

هستند را حل کنیم؛ سپس کاراکتر آخر رشته ی دوم b_m را با عملگر اضافه کردن و با هزینه ی یک، به رشته ی بدست آمده از حل زیر مسئله اضافه می کنیم تا رشته ی اول به رشته دوم تبدیل شود. به دلیل این که در مسیر تبدیل رشته ی بدست آمده از حل زیر مسئله به رشته ی مسئله اصلی، یک بار از عملگر اضافه کردن استفاده شده است؛ جواب مسئله اصلی یک واحد بیشتر از زیر مسئله است.

به عنوان مثال برای مسئله ای که رشته ی اول آن $\mathrm{A} =$ "edit" و رشته ی دوم آن $\mathrm{B} =$ "distance" هستند؛ کاراکتر آخر رشته ی دوم $b_8 = \mathrm{e}$ را از انتهای رشته دوم حذف می کنیم و زیر مسئله (کمترین تعداد از عملیات ها برای تبدیل رشته ی $\mathrm{A} =$ "edit" به $\mathrm{B} =$ "distanc") را حل می کنیم؛سپس کاراکتر آخر رشته ی دوم $b_8 = \mathrm{e}$ را با عملگر اضافه کردن و با هزینه ی یک، به رشته ی بدست آمده از حل زیر مسئله اضافه می کنیم تا رشته ی اول به رشته دوم تبدیل شود. به دلیل این که در مسیر تبدیل رشته ی بدست آمده از حل زیر مسئله به رشته ی مسئله اصلی، یک بار از عملگر اضافه کردن استفاده شده است؛ جواب مسئله اصلی یک واحد بیشتر از زیر مسئله است.

عملگر پاک کردن یک کاراکتر

برای تبدیل رشته ی اول به رشته ی دوم کافی است کاراکتر آخر رشته اول a_n را با عملگر پاک کردن و با هزینه ی یک از آخر رشته اول پاک کنیم و زیر مسئله ای که رشته های آن

$\mathrm{A} = a_1, a_2, a_3, ..., a_{n-1}$

$\mathrm{B} = b_1, b_2, b_3, ..., b_m$

هستند را حل کنیم؛ به دلیل این که در مسیر تبدیل رشته ی مسئله اصلی به رشته ی بدست آمده از حل زیر مسئله، یک بار از عملگر پاک کردن استفاده شده است؛ جواب مسئله اصلی یک واحد بیشتر از زیر مسئله است.

به عنوان مثال برای مسئله ای که رشته ی اول آن $\mathrm{A} =$ "edit" و رشته ی دوم آن $\mathrm{B} =$ "distance" هستند؛ کاراکتر آخر رشته ی اول $a_4 = \mathrm{e}$ را با عملگر پاک کردن و با هزینه ی یک از آخر رشته اول پاک کنیم و زیر مسئله (کمترین تعداد از عملیات ها برای تبدیل رشته ی $\mathrm{A} =$ "edi" به $\mathrm{B} =$ "distance") را حل می کنیم؛ به دلیل این که در مسیر تبدیل رشته ی مسئله اصلی به رشته ی بدست آمده از حل زیر مسئله، یک بار از عملگر پاک کردن استفاده شده است؛ جواب مسئله اصلی یک واحد بیشتر از زیر مسئله است.

برای باز نویسی رابطه ی میان مسئله و زیر مسائل آرایه دو بعدی D را به نحوی تعریف می کنیم که مقدار خانه ی $D_{i,j}$ برابر با کمترین تعداد عملیات هایی باشد که برای تبدیل رشته

$$\mathrm{A} = a_1, a_2, a_3, ..., a_i$$

به رشته ی

$$\mathrm{B} = b_1, b_2, b_3, ..., b_j$$

لازم است. رابطه ی میان مسئله و زیر مسائل به صورت زیر خلاصه می شود.

$$D_{i,j} = min \begin{cases} D_{i-1,j-1} & a_i = b_j \\ D_{i-1,j-1} + 1 & a_i \neq b_j \\ D_{i,j-1} + 1 & a_i \neq b_j \\ D_{i-1,j} + 1 & a_i \neq b_j \end{cases}$$

حال با توجه به رابطه ی بدست آمده باید زیر مسائل را به ترتیبی حلو ذخیره کنیم که قبل از حل یک مسئله، زیر مسئله های آن مسئله حل و ذخیره شده باشد. طبق رابطه میان مسئله و زیر مسائل باید برای حل مسئله $D_{i,j}$ باید مسائل $D_{i-1,j-1}$ ، $D_{i,j-1}$ و $D_{i-1,j}$ قبل از آن حل و ذخیره شده باشند. بنابراین بر روی آرایه از بالا به پایین و از چپ به راست حرکت می کنیم.(در این مسئله حرکت در راستاهای افقی و عمودی نسبت به هم ارجعیت ندارند؛ مسائلی وجود دارند که تریتب راستای حرکت حلقه ی اول و راستای حلقه ی دوم مهم است.) حال باید جواب زیر مسائلی را که به اندازه کافی کوچک هستند را در آرایه D ذخیره کنیم.(معمولا ردیف اول و ستون اول زیر مسائلی هستند که به اندازه کافی کوچک هستند.) اگر یکی از رشته ها هیچ کاراکتری نداشته باشد؛ زیر مسئله به اندازه ی کافی کوچک شده است که جواب آن را بدانیم.

حالت اول این است که رشته ی اول کاراکتری نداشته باشد؛ برای تبدیل رشته اول به رشته دوم با j کاراکتر باید j بار عملگر اضافه کردن یک حرف را روی رشته ی اول استفاده کنیم؛ تا رشته ی اول به رشته ی دوم تبدیل شود.

حالت دوم این است که رشته ی دوم کاراکتری نداشته باشد؛ برای تبدیل رشته اول با i کاراکتر به رشته دوم باید i بار عملگر پاک کردن یک حرف را روی رشته ی اول استفاده کنیم؛ تا رشته ی اول به رشته ی دوم تبدیل شود. روابط بالا را می توان به صورت زیر خلاصه کرد.

$$D_{0,j} = j \qquad j = 0, 1, 2, 3, ..., m$$

$$D_{i,0} = i \qquad i = 0, 1, 2, 3, ..., n$$

حال با توجه به رابطه ی به دست آمده و زیر مسائل حل شده به سراغ حل و ذخیره زیر مسائل پیچیده تر می رویم.

برای درک بهتر برنامه ریزی پویا ، مسئله ای با رشته های $A = \text{"distance"}$ و $B = \text{"editing"}$ را بررسی می کنیم. ابتدا زیر مسائلی که به اندازه کافی کوچک هستند را حل و در آرایه ذخیره می کنیم.

زمانی که هر دو رشته کاراکتری ندارند با صفر عملیات رشته ی اول به رشته دوم تبدیل می شود.

$$D_{0,0} = 0$$

زمانی که رشته ی اول کاراکتری ندارد و رشته ی دوم $B = \text{"e"}$ است با اضافه کردن کاراکتر "e" به رشته ی اول، رشته ی اول به رشته دوم تبدیل می شود.

$$D_{0,1} = 1$$

زمانی که رشته ی اول کاراکتری ندارد و رشته ی دوم $B = \text{"ed"}$ است با اضافه کردن کاراکتر "e" و "d" به رشته ی اول، رشته ی اول به رشته دوم تبدیل می شود.

$$D_{0,2} = 2$$

زیر مسائل $D_{0,j}$ که $j = 0, 1, 2, ..., m$ است؛ با منطقی مشابه حل می شوند.

$$D_{0,j} = j$$

زمانی که رشته اول $A = \text{"d"}$ است و رشته دوم کاراکتری ندارد با پاک کردن کاراکتر "d" از رشته ی اول، رشته ی اول به رشته دوم تبدیل می شود.

$$D_{1,0} = 1$$

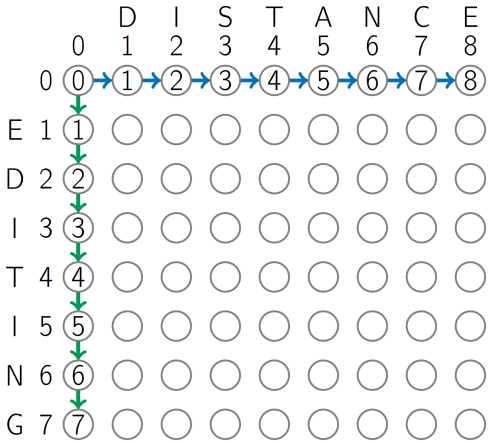
زمانی که رشته اول $A = \text{"di"}$ است و رشته دوم کاراکتری ندارد با پاک کردن کاراکتر "d" و "i" از رشته ی اول، رشته ی اول به رشته دوم تبدیل می شود.

$$D_{2,0} = 2$$

زیر مسائل $D_{i,0}$ که $i = 0, 1, 2, ..., n$ است؛ با منطقی مشابه حل می شوند.

$$D_{i,0} = i$$

پس از حل و ذخیره مقادیر بالا آرایه به شکل ۱۰.۱۱ زیر در می آید.



شکل ۱۰.۱۱: ابتدای کار

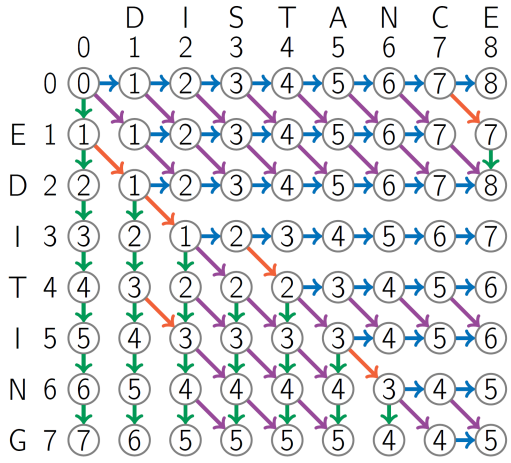
برای حل مسئله ی $D_{1,1}$ به دلیل اینکه کاراکتر های اول دو رشته برابر نیستند و طبق رابطه ی میان مسئله و زیر مسائل باید مقادیر زیر مسئله های $D_{0,0}$ ، $D_{1,0}$ و $D_{0,1}$ را بدانیم و از آن ها استفاده کنیم. جواب این مسئله به صورت زیر می شود.

$$D_{2,1} = min \begin{cases} D_{0,0} + 1 = 0 + 1 = 1 \\ D_{1,0} + 1 = 0 + 1 = 1 \\ D_{0,1} + 1 = 0 + 1 = 1 \end{cases}$$

برای حل مسئله ی $D_{2,1}$ به دلیل اینکه کاراکتر های اول دو رشته برابر هستند و طبق رابطه ی میان مسئله و زیر مسائل باید مقادیر زیر مسئله های $D_{1,0}$ ، $D_{1,1}$ و $D_{2,0}$ را بدانیم و از آن ها استفاده کنیم. جواب این مسئله به صورت زیر می شود.

$$D_{2,1} = min \begin{cases} D_{1,0} = 1 D_{1,0} + 1 = 1 + 1 = 2 \\ D_{1,1} + 1 = 1 + 1 = 2 \\ D_{2,0} + 1 = 2 + 1 = 3 \end{cases}$$

زیر مسائل D_i, j که $= 1, 2, ..., n$ و $j = 1, 2, ..., m$ است؛ با منطقی مشابه حل می شوند. پس از حل و ذخیره مقادیر بالا آرایه به شکل ۱۱.۱۱ زیر در می آید. جواب مسئله اصلی برابر است با مقدار $D_{n,m} = 5$ است.



شکل ۱۱.۱۱: انتهای کار

مراجع :

CLRS

opedia.ir

https://en.wikipedia.org

https://www.youtube.com/watch؟v=MiqoA-yF-۰M