Dynamic Programming

- یک روش موثر و کارآمد برای حل مسائل بهینه سازی و جستجو است.
- دو ویژگی زیرمسئله های هم پوشان و زیرساخت های بهینه.
- چارچوب استانداری برای فرموله کردن مسئله وجود ندارد.
- شبیه روش تقسیم و حل، جواب مسئله را با ترکیب کردن جواب زیر مسئله ها بدست می آورد.
- برنامه ریزی پویا بهتر است در مسائلی استفاده کنیم که زیرمسائل آنها مستقل نیست.
- الگوریتم برنامه ریزی پویا زیرمسئله ها را یکبار حل و جواب آنها را در یک جدول ذخیره می کند و با این کار از تکرار اجرای زیرمسائل در زمانی که مجددا به جواب آن احتیاج داریم جلوگیری می شود.
- جواب یک مسئله بهینه ساز را بتوان از ترکیب جوابهای بهینه زیرمسئله های آن بدست آورد
- زیرساختار بهینه، یعنی جواب هر زیرمسئله از مسئله اصلی خود بهینه باشد.
- اگر زیرمسئله ها هم پوشان نباشد به آن تقسیم و حل می گویند. مرتب سازی ادغامی و سریع هر دو تقسیم و حل هستند.
- زیرمسئله های هم پوشان به معنای کوچک بودن فضای زیرمسئله هاست، به این معنا که هر الگوریتم بازگشتی برای حل این مسئله باید به جای ایجاد زیرمسئله های جدید، زیرمسئله های تکراری را بارها و بارها حل کند برای مثال فرمول بازگشتی دنباله فیبوناچی

$$F_i = F_{i-1} + F_{i-2}$$

یک الگوریتم Dynamic Programming مشخص نیست بلکه یک تکنیک مانند divide and conquer است.

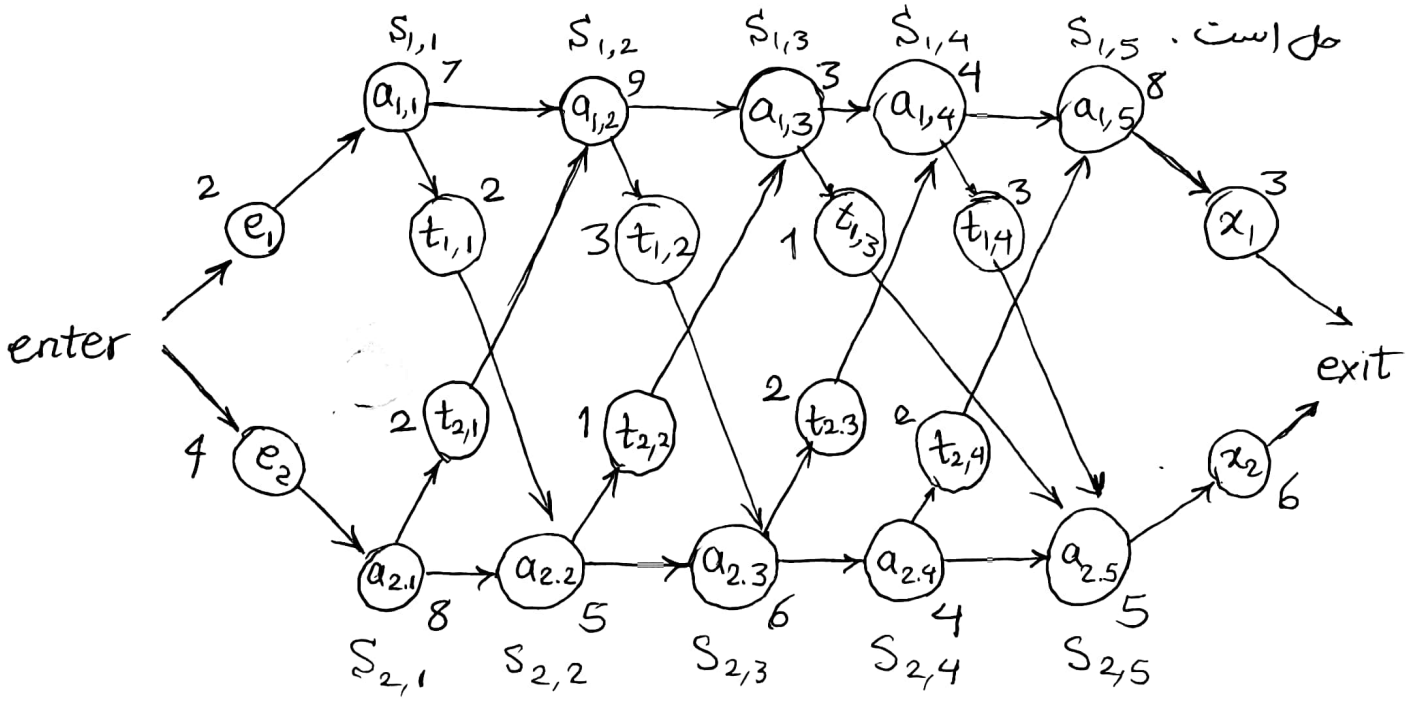
- برای مسائل بهینه سازی استفاده می‌شود.
  - یک جواب با مقدار بهینه را می‌یابد.
  - Minimization یا Maximization

یک روش ۴ مرحله ای است:

۱. شناسایی ساختار یک جواب بهینه.
۲. مقدار جواب بهینه را به صورت بازگشتی تعریف می‌کند.
۳. مقدار یک جواب بهینه را به روش bottom-up محاسبه می‌کند.
۴. یک جواب بهینه از اطلاعات محاسبه شده می‌سازد.

Assembly-line scheduling

یک مثال ساده از dynamic Programming است. در واقع با الگوریتمهای گراف نیز قابل حل است.



کارخانه اتومبیل با دو خط تولید (مونتاژ)

- هر خط n تا station دارد: $S_{1,1}, \dots, S_{1,n}$ و $S_{2,1}, \dots, S_{2,n}$
- station های مرتبط $S_{1,j}$ و $S_{2,j}$ عمل یکسانی را انجام می دهند که می تواند زمانهای متفاوت $a_{1,j}$ و $a_{2,j}$ طول بکشد.
- زمانهای ورود e_1 و e_2 هستند.
- زمانهای خروج x_1 و x_2 هستند.
- وقتی که وارد یک station می شویم بعدش:
  - می توانیم در همان خط تولید بمانیم (بدون هزینه)
  - به خط دیگر منتقل شویم که هزینه بعد از $S_{i,j}$ برابر $t_{i,j}$ است

مسئله: همه این cost ها داده شده است (time=cost). چه station هایی از میان خط ۱ و خط ۲ باید به عنوان سریعترین مسیر کارخانه انتخاب شوند.

آیا همه امکان ها باید بررسی شوند؟

- 2^n حالت وجود دارد.
- اگر n بزرگ باشد، مناسب نیست.

structure of an optimal solution

در مورد سریعترین سیر از ورودی و از سیان $S_{1,j}$ فکر کنید.

- اگر $j=1$ ساده است: فقط تشخیص می دهیم که چه زمانی طول می کشد که از $a_{1,1}$ بگذریم
- اگر $j \geq 2$ دو انتخاب وجود دارد که چگونه به $S_{1,j}$ برسیم:

  - از $S_{1,j-1}$ و سپس مستقیم به $S_{1,j}$
  - از $S_{2,j-1}$ و سپس منتقل شویم به $S_{1,j}$

فرض کنید که سریعترین سیر از میان $S_{1,j-1}$ می گذرد.

Key observation: ما باید انتخاب کنیم سریعترین سیری که از ورودی به $S_{1,j-1}$ است در این راه حل. اگر یک سیر سریعتر به $S_{1,j-1}$ وجود داشت، ما باید از آن استفاده کنیم به جای اینکه از سیر سریعترکه از $S_{1,j-1}$ می گذرد.

حال فرض کنید که سریعترین سیر از $S_{2,j-1}$ می گذرد، دوباره، ما باید سریعترین سیر که از $S_{2,j-1}$ می گذرد را بدست آوریم. درغیراینصورت سیر سریعتری که از $S_{2,j-1}$ می گذرد را استفاده می کنیم که یک سیر سریعتر از $S_{1,j}$ می دهیم.

Generally: یک راه حل بهینه برای یک مسئله (سریعترین سیر به $S_{1,j}$) جوابی برای یک زیرمسئله (سریعترین سیر به $S_{1,j-1}$ یا $S_{2,j-1}$) دارد.

این در واقع optimal substructure است.

optimal substructure: استفاده می شود تا یک جواب بهینه برای مسئله از راه حل بهینه به زیرمسئله ها ساخته شود.

سریعترین مسیر به $S_{1,j}$:

- سریعترین مسیر به $S_{1,j-1}$ و سپس مستقیماً به $S_{1,j}$ یا
- سریعترین مسیر به $S_{2,j-1}$ و انتقال از line2 به line1 و رفتن به $S_{1,j}$

به طور متقارن، سریعترین مسیر به $S_{2,j}$:

- سریعترین مسیر به $S_{2,j-1}$ و سپس مستقیماً به $S_{2,j}$ یا
- سریعترین مسیر به $S_{1,j-1}$ و انتقال از line1 به line2 و رفتن به $S_{2,j}$

بنابراین، برای حل مسئله یافتن سریعترین مسیر به $S_{1,j}$ و $S_{2,j}$ حل زیرمسئله یافتن سریعترین مسیرها به $S_{1,j-1}$ و $S_{2,j-1}$ لازم است.

Recursive solution

Let $f_i[j]$ = fastest time to get through $S_{i,j}$

هدف: f^* ، سریعترین زمان برای کل مسیر.

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

$$f_1[1] = e_1 + a_{1,1}$$

$$f_2[1] = e_1 + a_{2,1}$$

for $j = 2, \ldots, n$:

$$f_1[j] = \min\left(f_1[j-1] + a_{1,j},\ f_2[j-1] + t_{2,j-1} + a_{1,j}\right)$$

$$f_2[j] = \min\left(f_2[j-1] + a_{2,j},\ f_1[j-1] + t_{1,j-1} + a_{2,j}\right)$$

$f_i[j]$ جواب بهینه مسئله را می‌دهد.

- $l_i[j]$ = line # (1 or 2) whose station $j-1$ is used in fastest way through $S_{i,j}$.

به عبارت دیگر مثل از $S_{i,j}$ ، $S_{l_i[j],j-1}$ است

l^* = Line # whose station n is used.

For example:

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

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

$f^* = 35$

$l^* = 1$

مسیر بهینه با مقادیر l مشخص می‌شود.

compute an optimal solution

از فرمول بازگشتی بالا می توانیم یک الگوریتم بازگشتی بسازیم.

Let $r_i(j)$ = # of references made to $f_i[j]$

$r_1(n) = r_2(n) = 1$

$r_1(j) = r_2(j) = r_1(j+1) + r_2(j+1)$ for $j = 1, \ldots, n-1$

ادعا می کنیم: $r_i(j) = 2^{n-j}$

Basis: $j = n$, $2^{n-j} = 2^0 = 1 = r_i(n)$

Inductive step: (اثبات با استفاده از استقرای ریاضی)

Assume $r_i(j+1) = 2^{n-(j+1)}$

Then $r_i(j) = r_1(j+1) + r_2(j+1)$

$= 2^{n-(j+1)} + 2^{n-(j+1)}$

$= 2^{n-(j+1)+1}$

$= 2^{n-j}$

فقط $f_i[1]$ خودش به تنهایی 2^{n-1} بار تکرار می شود.

بنابراین روش بالا به پایین روش خوبی برای محاسبه $f_i[j]$ نیست.

مشاهده: $f_i[j]$ وابسته به فقط $f_1[j-1]$, $f_2[j-1]$ برای $2 \leq j$ است.

بنابراین، بهتر است با روش افزایشی آن را محاسبه شود.

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 $f^* = f_2[n] + x_2$
      $l^* = 2$

Go through example.

Constructing an optimal solution

PRINT-STATIONS(l, n)

$i \leftarrow l^*$
print "line " i ", station " n
for $j \leftarrow n$ **downto** 2
    do $i \leftarrow l_i[j]$
        print "line " i ", station " $j - 1$

Go through example.

Time $= \Theta(n)$