(Dynamic Programing) برنامەنویسی پویا

برنامه نویسی پویا (dynamic programming)

- برنامه نویسی پویا روشی است که برای کاهش زمان اجرای الگوریتم‌های بازگشتی مورد استفاده قرار می‌گیرد.
- برای برخی از الگوریتم‌های بازگشتی قابل استفاده است.
- بیشتر این الگوریتم‌ها مربوط به مسائل بهینه‌سازی هستند.
- قبل از اینکه به تعریف آن با جزئیات بیشتر بپردازیم، به یک مثال توجه نمایید.

اولین مثال

- دنباله فیبوناچی:

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597, 2584, 4181, 6765, …

F_0	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}	F_{11}	F_{12}	F_{13}	F_{14}	F_{15}	F_{16}	F_{17}	F_{18}	F_{19}	F_{20}
0	1	1	2	3	5	8	13	21	34	55	89	144	233	377	610	987	1597	2584	4181	6765

- رابطه بازگشتی:

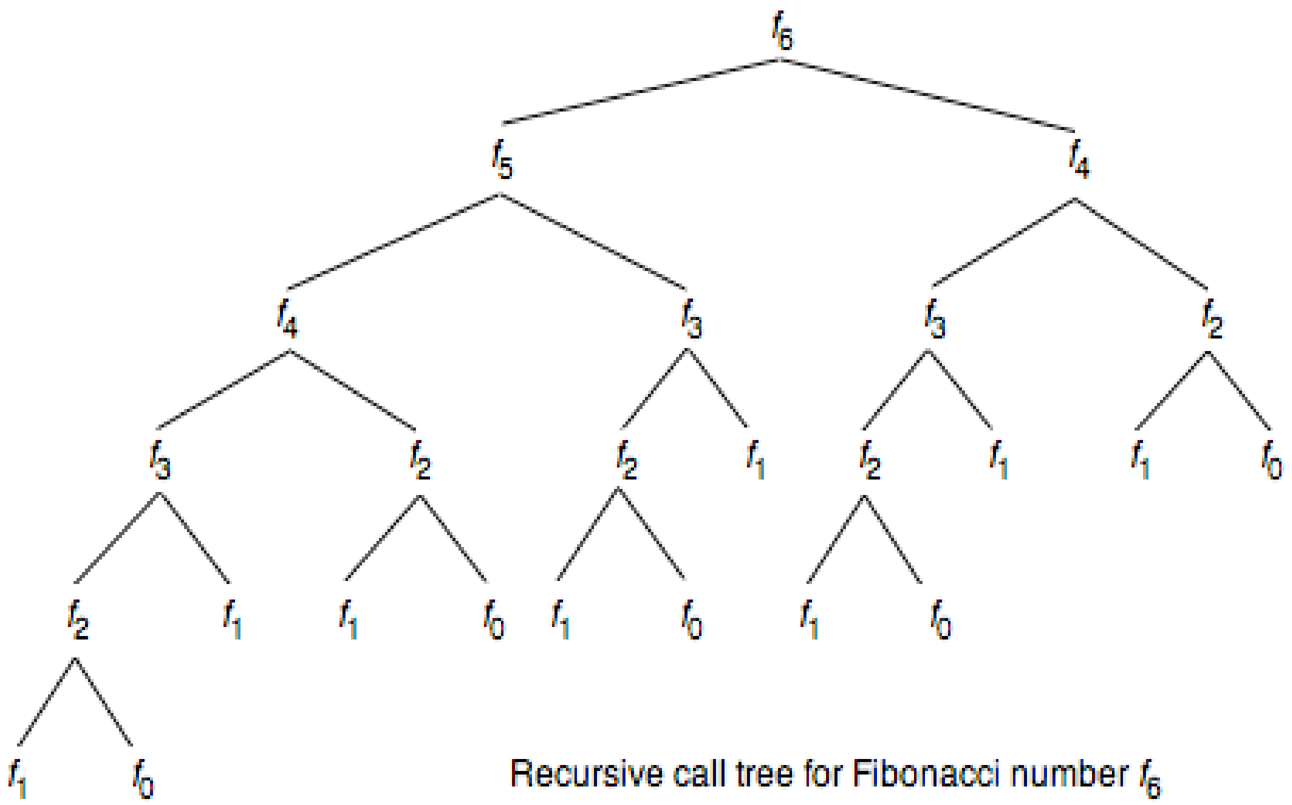
$$f(n) = \begin{cases} 0 & n = 0 \\ 1 & n = 1 \\ f(n-1) + f(n-2) & n > 1 \end{cases}$$

اولین مثال

- مرتبه زمانی؟

$$t(n) = t(n-1) + t(n-2) + \theta(1)$$

- با یک حساب سرانگشتی الگوریتم از مرتبه نمایی است.
  - ✓ حد بالا: $O(2^n)$ (فرض اینکه همه زیرمسائل به فرم $f(n-1)$ هستند که درنتیجه ارتفاع درخت: n)
  - ✓ حد پایین: $\Omega\left(2^{\frac{n}{2}}\right)$ (فرض اینکه همه زیرمسائل به فرم $f(n-2)$ هستند، در نتیجه ارتفاع درخت: $\frac{n}{2}$)



Recursive call tree for Fibonacci number f_6

مرتبه زمانی تولید اعداد فیبوناچی

- حد محکم برای این الگوریتم البته $\Theta\left(\left(\frac{1+\sqrt{5}}{2}\right)^n\right)$ است.
- عدد $\frac{1+\sqrt{5}}{2}$ به نسبت طلایی معروف است.
- حتما اسم مستطیل طلایی را قبلا شنیده‌اید. که نسبت طول به عرض آن همین نسبت طلایی است.
- در واقع، هرگاه خطی را به دو قسمت a و b $(a > b)$ تقسیم کنند طوری که $\frac{a+b}{a} = \frac{a}{b}$ نسبت طلایی به وجود آمده است.

$$\frac{a+b}{a} = \frac{a}{b} = \phi = \frac{1+\sqrt{5}}{2} \approx 1.6$$

$$\frac{a}{b} = \frac{f(n)}{f(n-1)}$$

$$\frac{5}{3} = 1.666\ldots$$

$$\frac{8}{5} = 1.6$$

$$\frac{13}{8} = 1.625$$

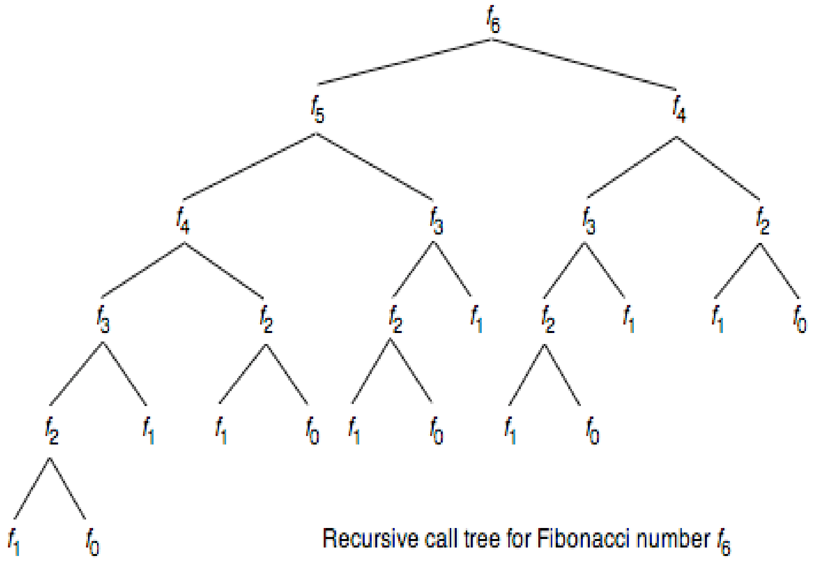
$$\frac{21}{13} = 1.615\ldots$$

$$\frac{34}{21} = 1.619\ldots$$

$$\frac{55}{34} = 1.617$$

...

کاهش مرتبه زمانی تولید اعداد فیبوناچی



Recursive call tree for Fibonacci number f_6

- آیا می‌توان این زمان را کاهش داد؟
  - ✓ به تکرارهای زیرمسائل در درخت بازگشت دقت کنید.
  - ✓ اگر هرکدام را فقط یک بار حل کنیم...
- دو راه برای این کار وجود دارد (اگر امکان‌پذیر باشد):
  - ✓ روش بالا به پایین: Memoaization
    - جواب زیرمسائلی را که حل می‌شوند در یک جدول ذخیره می‌کنیم تا نیاز نباشد آن‌ها را چند بار حل کنیم.
  - ✓ روش پایین به بالا: Dynamic Programming (DP)
    - ساختار بهینه را می‌سازیم و برمبنای آن زیرمسائل را هرکدام یک بار حل می‌کنیم. ممکن است نیاز باشد همه یا بخشی از زیرمسائل حل‌شده را در جدولی ذخیره کنیم.
- برای دنباله فیبوناچی؟

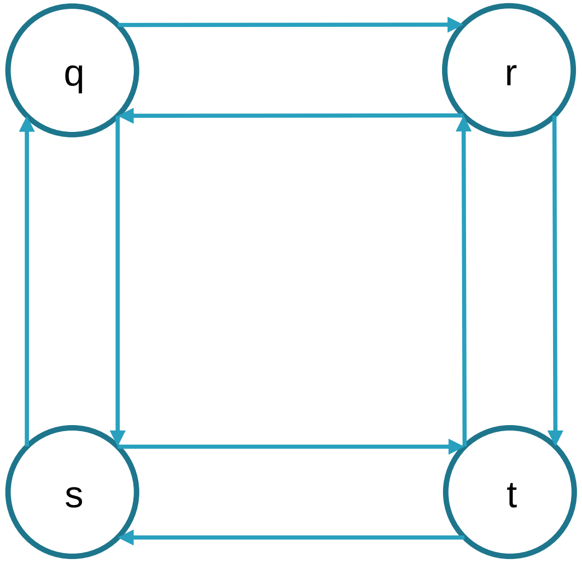
$$0, 1 \rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow 5 \rightarrow 8$$

چه مسائلی با DP قابل حل هستند؟

- مسائلی که:
  - ✓ بازگشتی باشند.
  - ✓ رابطه بهینه برای آنها موجود باشد: یعنی هر زیر مسئله خود از پاسخ بهینه زیرمسائل کوچکتر ساخته شده باشد.
    - به عنوان مثال اگر در کوتاهترین مسیر از تهران به شاهرود پس از عبور از سمنان، از دامغان می‌گذرید، مطمئنا در کوتاهترین مسیر از سمنان به شاهرود هم باید از دامغان عبور کرد.
    - مثالی که ساختار بهینه ندارد: یافتن بلندترین مسیر ساده در یک گراف.
  - ✓ تکرار در زیرمسائل آنها وجود داشته باشد.
    - به عنوان مثال: فیبوناچی
    - مثالی که تکرار ندارد: Merge Sort

مثال‌هایی از مسائلی با DP قابل حل نیستند.

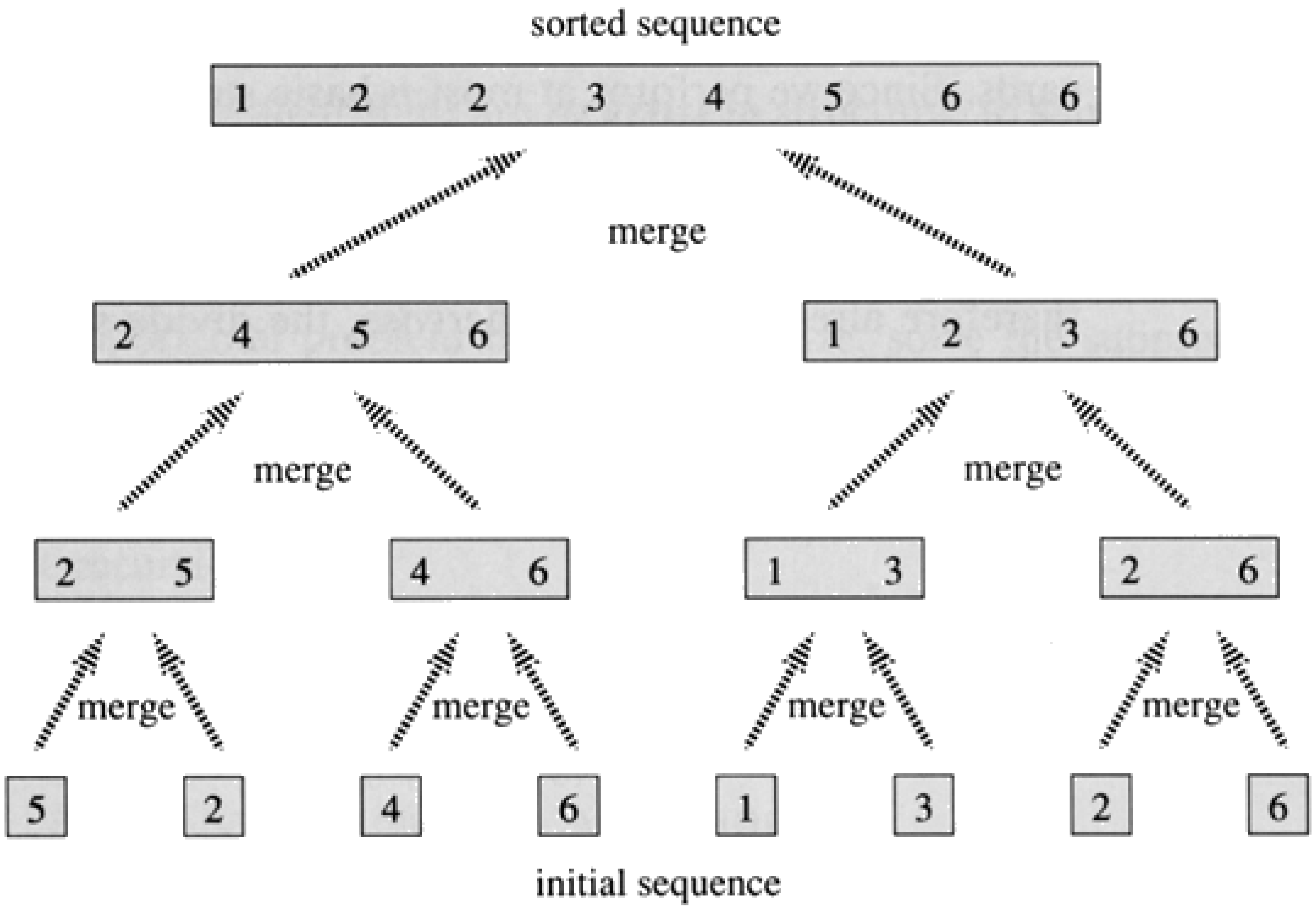
- یافتن بلندترین مسیر ساده در یک گراف
  - ✓ یک مسیر زمانی ساده نامیده می‌شود که دارای رئوس تکراری (دور) نباشد.



- ✓ فرض کنید می‌خواهیم بلندترین مسیر ساده از q به t را بیابیم.
- ✓ واضح است که پاسخ $q \to r \to t$ (یا مسیر مشابه آن از s) است.
- ✓ بنابراین اگر ساختار بهینه برقرار باشد، باید بلندترین مسیر ساده از r به t هم $r \to t$ باشد. آیا چنین است؟
- ✓ خیر پاسخ $r \to q \to s \to t$ است.

مثال‌هایی از مسائلی با DP قابل حل نیستند.

- Merge Sort



روند DP

- در DP اگر شرایط ذکرشده برقرار باشد:
  - ✓ رابطه بازگشتی نوشته می‌شود.
  - ✓ تک تک زیر مسائل از پایین با بالا حل می‌شوند.
  - ✓ ترتیب حل این زیرمسائل باید به گونه‌ای باشد که برای هر زیرمسئله بزرگتر زیرمسائل کوچکتر مورد نیاز آن قبلا حل شده باشند.
  - ✓ حل زیرمسائل تا حل مسئله اصلی (بزرگترین زیرمسئله) ادامه می‌یابد.
  - ✓ در اکثر مسائل، علاوه بر جواب آخر (مثلا طول کوتاهترین مسیر در گراف) مسیر رسیدن به آن (مثلا خود کوتاهترین میسر در گراف) نیز مهم است. که باید با بازگشت به عقب و دنبال کردن زیر مسائل آن را یافت.
  - ✓ تا تمام زیرمسائل حل‌نشده باشند این مسیر قابل یافتن نیست.

مزایای DP

- استفاده از DP معمولا مسائلی با زمان نمایی را به مسائلی با زمان چندجمله‌ای و گاه حتی خطی بدل می‌کند. چرا؟

- استفاده از DP حافظه‌ی مصرفی را هم کاهش می‌دهد. چرا؟

مسائلی که به بررسی آن‌ها خواهیم پرداخت

- برنامه‌ریزی خط‌تولید کارخانه (Assembly Line scheduling)
- ضرب زنجیره‌ای ماتریس‌ها (Matrix-chain multiplication)
- بزرگترین زیردنباله مشترک (Longest common subsequence)
- درخت جستجوی دودویی بهینه (Optimal binary search trees)