الگوریتم پیشرفته

استراتژی تحول و غلبه: کاهش مسئله (Problem Reduction)

دکتر حامد فهیمی

دانشگاه فردوسی مشهد

دانشکده علوم ریاضی

مبحث کاهش مسئله یک استراتژی قدرتمند در طراحی الگوریتم‌ها و بخشی از رویکرد کلی **تحول و غلبه (Transform-and-Conquer)** است.

---

۱. تعریف و مفهوم کاهش مسئله

کاهش مسئله (Problem Reduction) یک استراتژی حل مسئله است که در آن، اگر نیاز به حل **مسئله ۱** باشد، آن را به **مسئله ۲** که روش حل آن مشخص است، **کاهش** می‌دهیم.

به بیان ساده، شما یک مسئله‌ی داده شده را به نمونه‌ای از یک مسئله‌ی دیگر تبدیل می‌کنید که می‌توان آن را با یک الگوریتم از پیش **شناخته شده** حل کرد.

- **اهمیت عملی:** یک الگوریتم مبتنی بر کاهش، زمانی ارزش عملی دارد که اجرای آن (شامل کاهش مسئله ۱ به مسئله ۲ و سپس حل مسئله ۲) **کارآمدتر** از حل مستقیم مسئله ۱ باشد.
- **اهمیت نظری:** این استراتژی در علوم کامپیوتر نظری نقش اساسی دارد و برای **دسته‌بندی مسائل بر اساس پیچیدگی** (که در نظریه NP-Complete مطرح می‌شود) استفاده می‌شود.

---

۲. مثال‌های کاربردی کاهش مسئله

این استراتژی در بسیاری از الگوریتم‌ها و حوزه‌های علوم کامپیوتر کاربرد دارد:

الف) محاسبه کمترین مضرب مشترک (lcm)

- **مسئله اصلی:** محاسبه **کمترین مضرب مشترک (Least Common Multiple)** دو عدد صحیح مثبت m و n، یعنی $\text{lcm}(m, n)$.
- **روش کاهش:** به جای استفاده از روش‌های ناکارآمد مبتنی بر فاکتورگیری اول، این مسئله را به مسئله محاسبه **بزرگترین مقسوم‌علیه مشترک (gcd - Greatest Common Divisor)** کاهش می‌دهیم. الگوریتم اقلیدس یک روش بسیار کارآمد برای محاسبه $\text{gcd}(m, n)$ است.
- **فرمول کاهش:**

$$\frac{m \cdot n}{\text{gcd}(m, n)} = \text{lcm}(m, n)$$

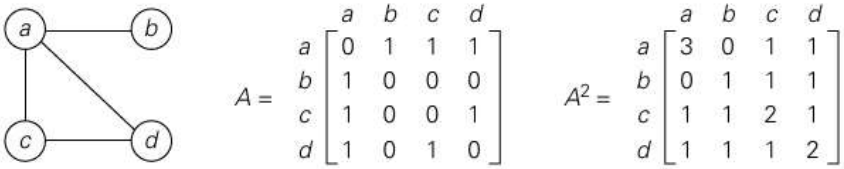
- **مثال:** $\text{lcm}(24, 60)$: ابتدا $\text{gcd}(24, 60) = 12$ را محاسبه می‌کنیم. سپس $\text{lcm}(24, 60) = \frac{60 \cdot 24}{12} = \frac{1440}{12} = 120$.
- **کارایی:** از آنجا که الگوریتم اقلیدس برای gcd دارای کارایی زمانی $O(\log n)$ است، کارایی الگوریتم lcm مبتنی بر این کاهش نیز در همین کلاس خواهد بود.

ب) شمارش مسیرها در یک گراف

- **مسئله اصلی:** شمارش تعداد مسیرهای با طول k بین دو رأس (vertex) در یک گراف (جهت‌دار یا بدون جهت).
- **روش کاهش:** این مسئله به محاسبه توان مناسبی از **ماتریس مجاورت (Adjacency Matrix)** گراف کاهش می‌یابد.
- **قاعده کاهش:** تعداد مسیرهای مختلف با طول $k > 0$ از رأس i-ام به رأس j-ام، برابر است با **عنصر** (i, j)-**ام ماتریس** A^k، که در آن A ماتریس مجاورت گراف است.
- **الگوریتم:** می‌توان از الگوریتم‌های توان‌رسانی دودویی که برای اعداد استفاده می‌شود، برای محاسبه A^k نیز استفاده کرد.

مثال خاص: شمارش مسیرها با ماتریس مجاورت

به عنوان یک مثال مشخص، **گراف شکل** را در نظر بگیرید.**ماتریس مجاورت** آن (A) و **مربع آن** (A^2)، به ترتیب، تعداد مسیرهای به **طول ۱** و **طول ۲** را بین رئوس متناظر گراف نشان می‌دهند.
به طور خاص، **سه مسیر** به طول ۲ وجود دارد که از رأس a شروع شده و به رأس a ختم می‌شوند: $(a-b-a, a-c-a, a-d-a)$؛اما تنها **یک مسیر** به طول ۲ از a به c وجود دارد که عبارت است از: $(a-d-c)$.



A graph, its adjacency matrix A, and its square A^2. The elements of A and A^2 indicate the numbers of paths of lengths 1 and 2, respectively.

ج) کاهش مسائل بهینه‌سازی (Optimization Problems)

مسائل بهینه‌سازی می‌توانند به دو دسته اصلی تقسیم شوند: **ماکزیمم‌سازی** و **مینیمم‌سازی**. این دو مسئله به راحتی به یکدیگر قابل کاهش هستند.

- **کاهش مینیمم‌سازی به ماکزیمم‌سازی:**

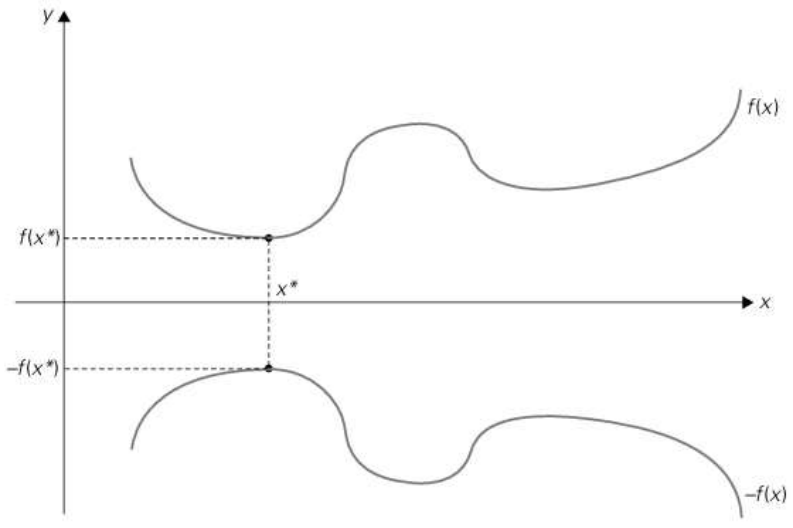
$$\max[-f(x)] - = \min f(x)$$

  - **توضیح:** برای یافتن کمترین مقدار یک تابع $(f(x))$، می‌توان ابتدا بیشترین مقدار منفی آن $(\max[-f(x)])$ را پیدا کرد و سپس علامت نتیجه را تغییر داد.
- **کاهش ماکزیمم‌سازی به مینیمم‌سازی:**

$$\min[-f(x)] - = \max f(x)$$

[

توجه (کاهش در حساب دیفرانسیل): رویه استاندارد حساب دیفرانسیل برای یافتن نقاط اکسترمم یک تابع $(f(x))$ نیز بر مبنای کاهش مسئله استوار است؛ به این صورت که مسئله بهینه‌سازی به **حل معادله‌ی** $f'(x) = 0$ (یافتن نقاط بحرانی) کاهش می‌یابد.



Relationship between minimization and maximization problems: $\min f(x) = -\max[-f(x)]$.

(مطالعه برنامه ریزی خطی آزاد است.)

د) برنامه‌ریزی خطی (Linear Programming)

- **تعریف:** مسئله‌ای که در آن به دنبال بهینه‌سازی (ماکزیمم یا مینیمم کردن) یک **تابع خطی** از چند متغیر، تحت محدودیت‌هایی به شکل **معادلات خطی** و **نامعادلات خطی** هستیم.

- **اهمیت کاهش:** بسیاری از مسائل تصمیم‌گیری بهینه (مانند زمانبندی خدمه، شبکه‌های حمل و نقل و برنامه‌ریزی تولید) می‌توانند به یک نمونه از مسئله برنامه‌ریزی خطی کاهش داده شوند.
- **فرم کلی مسئله:**
  - ماکزیمم (یا مینیمم) کردن $\sum_{j=1}^{n} c_j x_j$
  - تحت محدودیت‌های: $a_{i1}x_1 + \cdots + a_{in}x_n \leq (\text{یا } \geq \text{ یا } =) b_i$ برای $i = 1, \ldots, m$
  - و محدودیت‌های عدم منفی بودن: $x_1 \geq 0, \ldots, x_n \geq 0$.

کاهش مسئله کوله‌پشتی (Knapsack Problem) به برنامه‌ریزی خطی:

- **مسئله کوله‌پشتی پیوسته (Continuous Knapsack Problem):** این مسئله به یک برنامه‌ریزی خطی استاندارد تبدیل می‌شود.
  - **هدف:** ماکزیمم کردن $\sum_{j=1}^{n} v_j x_j$ (کل ارزش)
  - **محدودیت‌ها:** $\sum_{j=1}^{n} w_j x_j \leq W$ (کل وزن کمتر از ظرفیت W) و $0 \leq x_j \leq 1$ (x_j کسر گرفته شده از شیء j است).
- **مسئله کوله‌پشتی صفر-یک (0-1 Knapsack Problem):** این مسئله به یک **برنامه‌ریزی خطی عدد صحیح (Integer Linear Programming)** تبدیل می‌شود.
  - **هدف:** ماکزیمم کردن $\sum_{j=1}^{n} v_j x_j$
  - **محدودیت‌ها:** $\sum_{j=1}^{n} w_j x_j \leq W$ و $x_j \in \{0, 1\}$ (x_j یا شیء را می‌گیریم (۱) یا نمی‌گیریم (۰)).
  - **پیچیدگی:** این تغییر به ظاهر کوچک، پیچیدگی مسئله را به شدت افزایش می‌دهد و یافتن الگوریتم زمان چندجمله‌ای برای حل یک نمونه دلخواه از برنامه‌ریزی خطی عدد صحیح، هنوز یک مسئله باز است.

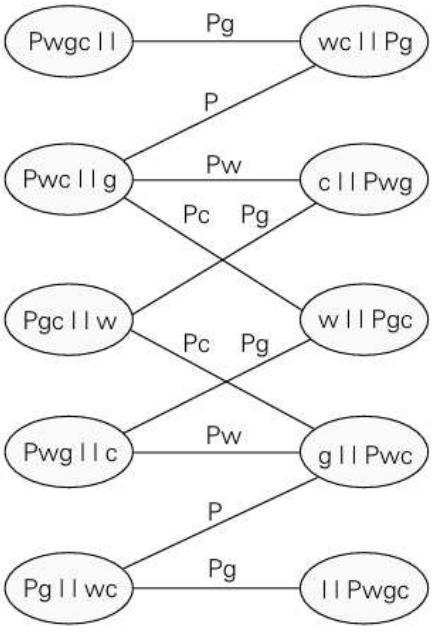
ه) کاهش به مسائل گراف (State-Space Graphs)

بسیاری از معماها و بازی‌ها با استفاده از کاهش به مسائل گراف حل می‌شوند.

- **گراف فضای حالت (State-Space Graph):** در این نوع کاهش:
  - **رأس‌ها (Vertices):** نمایانگر حالت‌های (وضعیت‌های) ممکن مسئله هستند.
  - **یال‌ها (Edges):** نمایانگر انتقال‌های مجاز بین حالت‌ها هستند.
  - **حل مسئله:** تبدیل می‌شود به یافتن یک **مسیر (Path)** از رأس حالت اولیه (Initial State) به رأس حالت هدف (Goal State).

مثال: معمای دهقان، گرگ، بز و کلم (The River-Crossing Puzzle)

- **مسئله:** دهقانی باید یک گرگ، یک بز و یک کلم را به آن طرف رودخانه ببرد. قایق فقط ظرفیت دهقان و یک شیء دیگر را دارد. در غیاب دهقان، گرگ بز را می‌خورد و بز کلم را. راه حلی پیدا کنید.
- **کاهش:** با رسم گراف فضای حالت (مانند شکل) که رأس‌های آن وضعیت‌های ایمن (مانند `||Pwgc` یا `Pwgc||`) و یال‌های آن حرکت‌های مجاز هستند، مسئله به **یافتن مسیر** از حالت اولیه (`||Pwgc`) به حالت هدف (`Pwgc||`) کاهش می‌یابد.
- **راه حل:** در این گراف، دو مسیر ساده مجزا با طول حداقل (۷ عبور از رودخانه) وجود دارد.



State-space graph for the peasant, wolf, goat, and cabbage puzzle.

---

۳. تمرین‌های حل شده

تمرین ۱: اثبات رابطه $\gcd$ و lcm

سؤال: تساوی $\text{lcm}(m,n) = \frac{m \cdot n}{\gcd(m,n)}$ را که زیربنای الگوریتم $\text{lcm}(m,n)$ است، اثبات کنید.

حل:

از تجزیه اعداد به عوامل اول استفاده می‌کنیم. فرض کنید تجزیه اول m و n به صورت زیر باشد:

$$m = p_1^{\alpha_1} p_2^{\alpha_2} \dots p_k^{\alpha_k}$$

$$n = p_1^{\beta_1} p_2^{\beta_2} \dots p_k^{\beta_k}$$

که در آن p_i تمام عوامل اول مشترک یا منحصر به فرد m و n هستند و $\alpha_i, \beta_i \geq 0$.

فرمول‌های $\gcd$ و lcm با استفاده از توان‌های عوامل اول به صورت زیر تعریف می‌شوند:

$$\gcd(m,n) = p_1^{\min(\alpha_1,\beta_1)} p_2^{\min(\alpha_2,\beta_2)} \dots p_k^{\min(\alpha_k,\beta_k)}$$

$$\text{lcm}(m,n) = p_1^{\max(\alpha_1,\beta_1)} p_2^{\max(\alpha_2,\beta_2)} \dots p_k^{\max(\alpha_k,\beta_k)}$$

اکنون حاصل ضرب $\gcd(m,n) \cdot \text{lcm}(m,n)$ را محاسبه می‌کنیم:

$$\gcd(m,n) \cdot \text{lcm}(m,n) = p_1^{\min(\alpha_1,\beta_1)+\max(\alpha_1,\beta_1)} \dots p_k^{\min(\alpha_k,\beta_k)+\max(\alpha_k,\beta_k)}$$

ما می‌دانیم که برای هر دو عدد دلخواه a و b:

$$\min(a,b) + \max(a,b) = a + b$$

بنابراین، توان هر عامل اول p_i در حاصل ضرب، برابر با مجموع توان‌های آن عامل در m و n خواهد بود $(\alpha_i + \beta_i)$:

$$\gcd(m,n) \cdot \text{lcm}(m,n) = p_1^{\alpha_1+\beta_1} \dots p_k^{\alpha_k+\beta_k}$$

$$\gcd(m,n) \cdot \text{lcm}(m,n) = (p_1^{\alpha_1} \dots p_k^{\alpha_k}) \cdot (p_1^{\beta_1} \dots p_k^{\beta_k})$$

$$\gcd(m,n) \cdot \text{lcm}(m,n) = m \cdot n$$

با تقسیم طرفین بر $\gcd(m,n)$، تساوی اثبات می‌شود:

$$\text{lcm}(m,n) = \frac{m \cdot n}{\gcd(m,n)}$$

تمرین ۲: پیاده‌سازی پشته با استفاده از صف

سؤال: پشته (Stack) از اصل **آخرین ورودی، اولین خروجی (LIFO)** پیروی می‌کند. صف (Queue) از اصل **اولین ورودی، اولین خروجی (FIFO)** پیروی می‌کند. چگونه می‌توانید عملیات‌های `Push` (اضافه کردن) و `Pop` (حذف کردن) یک پشته را با استفاده از عملیات‌های `Insert` (یا Enqueue) و `Delete` (یا Dequeue) یک صف پیاده‌سازی کنید؟

حل (کاهش عملیات پشته به عملیات صف):

برای پیاده‌سازی یک پشته با استفاده از دو صف (Q_1 و Q_2)، می‌توان عملیات `Push` را کارآمد ($O(1)$) و عملیات `Pop` را ناکارآمد ($O(N)$) طراحی کرد.

عملیات Push (اضافه کردن به پشته):
هدف این است که عنصر جدید، اولین عنصری باشد که در عملیات `Pop` بعدی حذف می‌شود (LIFO).

1. عنصر جدید (x) را به صف اصلی (Q_1) اضافه کنید.
- **تابع:** `Insert(Q1, x)`
- **پیچیدگی:** $O(1)$

عملیات Pop (حذف از پشته):
هدف این است که آخرین عنصر اضافه‌شده به پشته (که اکنون در انتهای صف Q_1 است) حذف شود.

1. تمام عناصر صف Q_1، به جز آخرین عنصر، را به صف کمکی (Q_2) منتقل کنید.
- تا زمانی که اندازه Q_1 بزرگتر از ۱ است:
  - $y = \text{Delete}(Q_1)$
  - $\text{Insert}(Q_2, y)$
2. عنصری که در Q_1 باقی مانده (که آخرین عنصر اضافه‌شده به پشته است)، همان عنصر مورد نیاز برای `Pop` است. آن را حذف کرده و برگردانید.
- $x = \text{Delete}(Q_1)$
3. نقش صف‌ها را جابجا کنید. (نام Q_1 و Q_2 را عوض کنید) تا Q_1 دوباره صف اصلی باشد.

پیچیدگی: عملیات `Push` در زمان ثابت ($O(1)$) و عملیات `Pop` در زمان خطی ($O(N)$) انجام می‌شود، که N تعداد عناصر موجود در پشته است. این یک نمونه از کاهش مسئله با تریدآف در کارایی عملیات‌ها است.