الگوریتم پیشرفته

نظریه NP-کامل بودن (NP-Completeness)

دانشگاه فردوسی مشهد

دانشکده علوم ریاضی

دکتر حامد فهیمی

این فصل به معرفی تکنیک‌هایی می‌پردازد که با استفاده از آن‌ها می‌توان ثابت کرد که برای یک مسئله‌ی مشخص، **الگوریتم کارآمدی (Efficient Algorithm) وجود ندارد**. اگرچه این نظریه نتایجی منفی ارائه می‌دهد، اما یک ابزار فوق‌العاده مفید برای طراح الگوریتم است. نظریه NP-کامل بودن به ما امکان می‌دهد تا تلاش‌هایمان را پربارتر متمرکز کنیم و مشخص سازد که چه زمانی جستجو برای یک الگوریتم کارآمد، محکوم به شکست است.

---

۱. مسائل و مفهوم کاهش (Problems and Reductions)

ما در بسیاری از مسائل، الگوریتم کارآمدی پیدا نکرده‌ایم. نظریه NP-کامل بودن ابزارهایی را فراهم می‌کند تا نشان دهیم که این مسائل، در سطح بنیادین، **واقعاً یک مسئله واحد هستند**.

تعریف مسئله و نمونه (Problem and Instance)

1. **مسئله‌ی الگوریتمی (Algorithmic Problem):** یک سؤال کلی است که دارای پارامترهای ورودی و شرایطی برای پاسخ یا راه‌حل رضایت‌بخش است.
2. **نمونه (Instance):** یک مسئله الگوریتمی است که پارامترهای ورودی آن مشخص شده باشند.

مثال: مسئله فروشنده دوره‌گرد (Traveling Salesman Problem - TSP)

- **مسئله:** کدام تور $(v_1, v_2, \ldots, v_n)$ هزینه‌ی $\sum_{i=1}^{n-1} d[v_i, v_{i+1}] + d[v_n, v_1]$ را به حداقل می‌رساند؟
- **ورودی (Input):** یک گراف وزن‌دار (G).
- هر گراف وزن‌دار یک **نمونه** از مسئله TSP را تعریف می‌کند. مسئله‌ی کلی TSP به دنبال یافتن الگوریتمی برای یافتن تور بهینه برای هر نمونه ممکن است.

ایده‌ی اصلی: کاهش (The Key Idea: Reduction)

کاهش (Reduction) یا **تبدیل (Translation)**، الگوریتمی است که یک مسئله را به مسئله‌ای دیگر تبدیل می‌کند.

دو مسئله‌ی الگوریتمی (Bandersnatch) و (Bo-billy) را در نظر بگیرید. فرض کنید الگوریتم زیر برای حل مسئله Bandersnatch وجود داشته باشد:

```
الگوریتم Bandersnatch(G):
1. ورودی G را به یک نمونه Y از مسئله Bo-billy تبدیل کن.
2. زیرروال Bo-billy را برای حل نمونه Y فراخوانی کن.
3. پاسخ Bo-billy(Y) را به عنوان پاسخ Bandersnatch(G) بازگردان.
```

این الگوریتم زمانی مسئله Bandersnatch را به درستی حل می‌کند که تبدیل به Bo-billy همواره **صحت پاسخ** را حفظ کند.

کاهش یعنی یک تبدیل نمونه‌ها از یک نوع مسئله به نمونه‌های نوع دیگر، به طوری که پاسخ‌ها حفظ شوند:

$$\text{پاسخ } \text{Bandersnatch}(G) = \text{پاسخ } \text{Bo-billy}(Y)$$

---

۲. مفهوم کاهش و کران پایین (Lower Bounds)

اگر تبدیل نمونه G به Y در زمان $O(P(n))$ انجام شود، دو نتیجه ممکن است وجود داشته باشد:

1. **اگر الگوریتم Bo-billy در زمان $O(P'(n))$ اجرا شود:**
این امر منجر به یک الگوریتم برای حل Bandersnatch در زمان $O(P(n) + P'(n))$ می‌شود. (با تبدیل مسئله و سپس اجرای زیرروال Bo-billy برای حل آن.)
2. **اگر کران پایین $\Omega(P''(n))$ برای حل Bandersnatch مشخص باشد:**
در این صورت، $\Omega(P''(n) - P(n))$ باید کران پایین برای محاسبه Bo-billy باشد.

- **دلیل:** اگر می‌توانستیم Bo-billy را سریع‌تر از این زمان حل کنیم، کاهش فوق، کران پایین مشخص شده برای Bandersnatch را نقض می‌کرد. از آنجا که این امر غیرممکن است، راهی برای حل سریع‌تر Bo-billy وجود ندارد.

نتیجه‌گیری مهم: این کاهش نشان می‌دهد که **Bo-billy از Bandersnatch آسان‌تر نیست**. بنابراین، **اگر Bandersnatch سخت باشد، Bo-billy نیز باید سخت باشد.**

درس کلیدی: کاهش‌ها راهی برای نشان دادن این است که دو مسئله اساساً **یکسان** هستند. یک الگوریتم سریع (یا فقدان آن) برای یکی از مسائل، وجود یک الگوریتم سریع (یا فقدان آن) را برای دیگری نتیجه می‌دهد.

---

۳. مسائل تصمیم‌گیری (Decision Problems)

کاهش‌ها تبدیل‌هایی بین مسائل هستند به طوری که پاسخ‌های آن‌ها در هر نمونه‌ی مسئله یکسان باشد.

مسائل تصمیم‌گیری ساده‌ترین و جالب‌ترین دسته از مسائل هستند که پاسخ‌های آن‌ها محدود به **درست (True)** و **نادرست (False)** است.

مزیت: کار کردن با تبدیل بین مسائل تصمیم‌گیری راحت‌تر است، زیرا هر دو تنها پاسخ‌های درست و غلط را مجاز می‌دانند. خوشبختانه، بسیاری از مسائل بهینه‌سازی (Optimization Problems) جالب را می‌توان به صورت مسائل تصمیم‌گیری بیان کرد که جوهر محاسباتی مسئله اصلی را حفظ می‌کنند.

مثال: تبدیل مسئله TSP به مسئله تصمیم‌گیری TSP (TSDP)

- **مسئله فروشنده دوره‌گرد تصمیم‌گیری (TSDP):**
  - **ورودی:** یک گراف وزن‌دار G و یک عدد صحیح k.
  - **خروجی:** آیا توری برای TSP با هزینه $\geq k$ وجود دارد؟

این نسخه‌ی تصمیم‌گیری جوهره‌ی مسئله فروشنده دوره‌گرد را در بر می‌گیرد:

- اگر یک الگوریتم سریع برای مسئله تصمیم‌گیری داشته باشید، می‌توانید با جستجوی دودویی بر روی مقادیر مختلف k، به سرعت به هزینه‌ی راه‌حل بهینه‌ی TSP دست یابید.
- با کمی هوشمندی بیشتر، می‌توانید **ترتیب واقعی تور** را با استفاده از یک راه‌حل سریع برای مسئله تصمیم‌گیری بازسازی کنید.

از این به بعد، به طور کلی در مورد مسائل تصمیم‌گیری صحبت می‌کنیم، زیرا کار با آن‌ها آسان‌تر است و همچنان قدرت نظریه NP-کامل بودن را حفظ می‌کنند.

---

۴. حل تمرین: کاهش برای الگوریتم‌های کارآمد (Closest Pair)

کاهش‌ها راهی محترمانه برای ایجاد الگوریتم‌های جدید از الگوریتم‌های قدیمی هستند. در این قسمت به مثالی از یک کاهش می‌پردازیم که منجر به یک الگوریتم کارآمد می‌شود.

مسئله جفت نزدیک‌ترین (Closest Pair)

- **مسئله:** یافتن جفتی از اعداد در یک مجموعه‌ی S که کوچکترین اختلاف بین آن‌ها را دارند.
  - *مثال:* نزدیک‌ترین جفت در $S = \{10, 4, 8, 3, 12\}$، جفت $(3, 4)$ است.

ما می‌توانیم آن را به یک مسئله تصمیم‌گیری تبدیل کنیم (آیا کوچکترین اختلاف کمتر یا مساوی یک آستانه t است؟).

- **مسئله تصمیم‌گیری جفت نزدیک به اندازه کافی (Close Enough Pair):**
  - **ورودی:** یک مجموعه‌ی S از n عدد و آستانه‌ی t.
  - **خروجی:** آیا جفتی $s_i, s_j \in S$ وجود دارد به طوری که $|s_i - s_j| \leq t$؟

الگوریتم مبتنی بر کاهش به مرتب‌سازی (Reduction to Sorting)

یافتن نزدیک‌ترین جفت، یک کاربرد ساده‌ی **مرتب‌سازی (Sorting)** است، زیرا نزدیک‌ترین جفت **باید** بعد از مرتب‌سازی، همسایه‌ی یکدیگر باشند.

الگوریتم CloseEnoughPair(S, t):

1. S را مرتب کن.
2. آیا $min_{1<=i<n}|s_{i+1} - s_i| <= t$ است؟

(یعنی کوچکترین اختلاف بین عناصر متوالی در لیست مرتب شده، کمتر یا مساوی t است؟)

تحلیل کارایی:

- **کاهش:** این مسئله به **مرتب‌سازی** کاهش داده شده است.
- **پیچیدگی زمانی:** اگر از الگوریتم مرتب‌سازی $O(n \log n)$ استفاده کنیم، زمان اجرای کل الگوریتم برای یافتن نزدیک‌ترین جفت $O(n \log n + n)$ خواهد بود که در نهایت $O(n \log n)$ است.

---

۵. حل تمرین: اثبات کران پایین مرتب‌سازی با استفاده از کاهش

این کاهش (جفت نزدیک‌ترین به مرتب‌سازی) می‌تواند برای اثبات کران‌های پایین نیز استفاده شود.

سؤال: فرض کنید ما بدانیم که مسئله **جفت نزدیک‌ترین به اندازه کافی** (Close Enough Pair) در بدترین حالت به $\Omega(n \log n)$ زمان نیاز دارد. با استفاده از کاهش فوق، چگونه می‌توان ثابت کرد که مرتب‌سازی نمی‌تواند سریع‌تر از $\Omega(n \log n)$ حل شود؟

حل:

ما یک کاهش از **جفت نزدیک‌ترین به اندازه کافی (A)** به **مرتب‌سازی (B)** داریم.

1. **فرض (Premise):** کران پایین برای حل مسئله A برابر با $\Omega(n \log n)$ است.
2. **فرضیه‌ی نقض:** فرض کنید یک الگوریتم سریع‌تر برای **مرتب‌سازی (B)** وجود دارد، مثلاً یک الگوریتم $O(P'(n))$ که $P'(n)$ سریع‌تر از $O(n \log n)$ است.
3. **ترکیب (Composition):** طبق کاهش، ما می‌توانیم مسئله A را در زمان $(\text{زمان حل B} + \text{زمان کاهش})O$ حل کنیم. زمان کاهش برای این مسئله $O(n)$ (برای بررسی اختلافات همسایه‌ها) است. پس زمان کل برای حل A می‌شود $O(n + P'(n))$.
4. **تناقض:** اگر $P'(n)$ سریع‌تر از $O(n \log n)$ باشد، کل زمان حل A یعنی $O(n + P'(n))$ نیز سریع‌تر از $\Omega(n \log n)$ خواهد بود. این نتیجه، فرض اولیه (کران پایین A برابر با $\Omega(n \log n)$ است) را نقض می‌کند.
5. **نتیجه‌گیری:** چون نقض کران پایین امکان‌پذیر نیست، فرضیه‌ی نقض ما (وجود الگوریتمی سریع‌تر از $O(n \log n)$ برای مرتب‌سازی) باید غلط باشد.

اثبات: این کاهش کفایت می‌کند تا ثابت کند که **مرتب‌سازی** در بدترین حالت نمی‌تواند سریع‌تر از $\Omega(n \log n)$ حل شود.