تحلیل سرشکن به روش پتانسیل (Potential Method)

مقدمه: تحلیل سرشکن چیست؟

در تحلیل الگوریتم‌ها، گاهی هزینه اجرای یک عمل (Operation) در بدترین حالت (Worst Case) بسیار زیاد است، اما این حالت به ندرت اتفاق می‌افتد. اگر دنباله‌ای از n عمل را اجرا کنیم، هزینه کل ممکن است آنقدرها هم بد نباشد. **تحلیل سرشکن (Amortized Analysis)** به ما کمک می‌کند تا هزینه "متوسط" هر عمل را در طول یک دنباله از عملیات محاسبه کنیم، حتی اگر برخی از آن عملیات‌ها به تنهایی بسیار پرهزینه باشند.

نکته مهم این است که تحلیل سرشکن با تحلیل حالت متوسط (Average-Case Analysis) فرق دارد؛ در اینجا هیچ احتمالی در کار نیست و ما یک **تضمین** برای هزینه متوسط هر عمل در بدترین دنباله از عملیات ارائه می‌دهیم.

سه روش متداول برای تحلیل سرشکن وجود دارد:

1. **روش تجمعی (Aggregate Analysis):** هزینه کل n عمل را $T(n)$ محاسبه کرده و هزینه سرشکن هر عمل را $T(n)/n$ در نظر می‌گیریم.
2. **روش حسابداری (Accounting Method):** به هر عمل یک هزینه سرشکن اختصاص می‌دهیم. اگر این هزینه از هزینه واقعی بیشتر باشد، "اعتبار" (Credit) ذخیره می‌کنیم تا بعداً برای عملیات پرهزینه استفاده شود.
3. **روش پتانسیل (Potential Method):** این روش که موضوع اصلی این بحث است، اعتبار را به صورت "انرژی پتانسیل" کل ساختمان داده در نظر می‌گیرد.

روش پتانسیل (The Potential Method)

این روش، کار از پیش پرداخت شده (prepaid work) را به عنوان **انرژی پتانسیل** (Φ) در نظر می‌گیرد که می‌تواند برای پرداخت هزینه عملیات‌های بعدی آزاد شود. ما پتانسیل را به کل ساختمان داده نسبت می‌دهیم، نه به اشیاء خاص درون آن.

تعاریف اصلی

- فرض کنید D_0 ساختمان داده اولیه ما باشد. ما دنباله‌ای از n عمل را روی آن انجام می‌دهیم.
- c_i هزینه **واقعی** عمل i-ام است.
- D_i حالت ساختمان داده پس از انجام عمل i-ام روی D_{i-1} است.
- **تابع پتانسیل** Φ : این تابع هر حالت D_i از ساختمان داده را به یک عدد حقیقی $\Phi(D_i)$ نگاشت می‌کند. این عدد، پتانسیل آن حالت است.

هزینه سرشکن (Amortized Cost) عمل i-ام ($\hat{c}_i$) به صورت زیر تعریف می‌شود:

$$\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$$

بنابراین، هزینه سرشکن برابر است با هزینه واقعی به اضافه **تغییر در پتانسیل** ناشی از آن عمل.

- اگر $\Phi(D_i) > \Phi(D_{i-1})$ ، تغییر پتانسیل مثبت است. یعنی هزینه سرشکن $\hat{c}_i$ از هزینه واقعی c_i بیشتر است. ما ساختمان داده را "بیشتر شارژ" کرده‌ایم و پتانسیل آن را افزایش داده‌ایم.
- اگر $\Phi(D_i) < \Phi(D_{i-1})$ ، تغییر پتانسیل منفی است. یعنی هزینه سرشکن $\hat{c}_i$ از هزینه واقعی c_i کمتر است. کاهش پتانسیل، هزینه واقعی عمل را "پرداخت" می‌کند.

هزینه سرشکن کل

هزینه سرشکن کل برای n عمل برابر است با:

$$\Sigma(\hat{c}_i) \text{ [for i=1 to n]} = \Sigma(c_i + \Phi(D_i) - \Phi(D_{i-1})) \text{ [for i=1 to n]}$$

این یک سری تلسکوپی است و نتیجه آن می‌شود:

$$\Sigma(\hat{c}_i) = \Sigma(c_i) + \Phi(D_n) - \Phi(D_0)$$

برای اینکه هزینه سرشکن کل یک کران بالا (Upper Bound) برای هزینه واقعی کل باشد، باید داشته باشیم:

$$\Phi(D_n) \geq \Phi(D_0)$$

در عمل، ما معمولاً $\Phi(D_0) = 0$ تعریف می‌کنیم و سپس نشان می‌دهیم که $\Phi(D_i) \geq 0$ برای تمام i ها. این کار تضمین می‌کند که پتانسیل هرگز منفی نمی‌شود و هزینه سرشکن کل همواره یک کران بالا برای هزینه واقعی کل است.

مثال ۱: عملیات پشته (Stack)

یک پشته با سه عمل را در نظر بگیرید:

- `PUSH(S, x)` : هزینه واقعی $c_i = 1$.
- `POP(S)` : هزینه واقعی $c_i = 1$.
- `MULTIPOP(S, k)` : k عنصر را برمی‌دارد. هزینه واقعی $c_i = \min(k, s)$ که s تعداد عناصر پشته است.

تابع پتانسیل: $\Phi(D)$ را برابر با **تعداد عناصر موجود در پشته** تعریف می‌کنیم.
برای یک پشته خالی اولیه D_0 ، داریم $\Phi(D_0) = 0$. از آنجایی که تعداد عناصر پشته هرگز منفی نیست، $\Phi(D_i) \geq 0$ برای همه i ها.

تحلیل هزینه‌ها:

1. **`PUSH`** :
   - هزینه واقعی: $c_i = 1$.
   - فرض کنید قبل از عمل، s عنصر در پشته وجود دارد. پس از `PUSH` ، ما در پشته $s+1$ عنصر خواهیم داشت.
   - تغییر پتانسیل: $\Delta\Phi = \Phi(D_i) - \Phi(D_{i-1}) = (s+1) - s = 1$.
   - هزینه سرشکن: $\hat{c}_i = c_i + \Delta\Phi = 1 + 1 = 2$.
2. **`POP`** :
   - هزینه واقعی: $c_i = 1$.
   - قبل از عمل s عنصر و بعد از آن $s-1$ عنصر داریم.
   - تغییر پتانسیل: $\Delta\Phi = \Phi(D_i) - \Phi(D_{i-1}) = (s-1) - s = -1$.
   - هزینه سرشکن: $\hat{c}_i = c_i + \Delta\Phi = 1 + (-1) = 0$.
3. **`MULTIPOP(S, k)`** :
   - هزینه واقعی: $c_i = k'$ که $k' = \min(k, s)$.
   - قبل از عمل s عنصر و بعد از آن $s-k'$ عنصر داریم.
   - تغییر پتانسیل: $\Delta\Phi = \Phi(D_i) - \Phi(D_{i-1}) = (s-k') - s = -k'$.
   - هزینه سرشکن: $\hat{c}_i = c_i + \Delta\Phi = k' + (-k') = 0$.

نتیجه: هزینه سرشکن هر سه عمل $O(1)$ است. بنابراین، هزینه کل یک دنباله از n عمل در بدترین حالت $O(n)$ خواهد بود.

مثال ۲: شمارنده دودویی (Binary Counter)

یک شمارنده k-بیتی را در نظر بگیرید که با عمل `INCREMENT` یکی به آن اضافه می‌شود. هزینه واقعی `INCREMENT` برابر با تعداد بیت‌هایی است که تغییر می‌کنند (از 1 به 0 یا از 0 به 1). در بدترین حالت (وقتی همه بیت‌ها 1 هستند)، هزینه k است.

تابع پتانسیل: $\Phi(D)$ را برابر با **تعداد بیت‌های 1 در شمارنده** تعریف می‌کنیم.

تحلیل هزینه `INCREMENT` :

- فرض کنید عمل i-ام `INCREMENT` باعث می‌شود t_i بیت از 1 به 0 تغییر کنند (reset شوند).
- این عمل حداکثر یک بیت را از 0 به 1 تغییر می‌دهد (set می‌کند).
- پس هزینه واقعی: $c_i \leq t_i + 1$.
- فرض کنید b_i تعداد بیت‌های 1 پس از عمل i-ام باشد. b_{i-1} تعداد بیت‌های 1 قبل از آن است.
- پس از ریست شدن t_i بیت و ست شدن حداکثر یک بیت، تعداد جدید بیت‌های 1 برابر است با: $b_i \leq b_{i-1} - t_i + 1$.

- تغییر پتانسیل: $\Delta\Phi = \Phi(D_i) - \Phi(D_{i-1}) = b_i - b_{i-1} \le (b_{i-1} - t_i + 1) - b_{i-1} = 1 - t_i$.
- هزینه سرشکن: $\hat{c}_i = c_i + \Delta\Phi \le (t_i + 1) + (1 - t_i) = 2$.

نتیجه: هزینه سرشکن عمل `INCREMENT` حداکثر 2، یعنی $O(1)$ است. بنابراین هزینه n عمل `INCREMENT` برابر $O(n)$ است.

تمرین‌ها و مسائل حل شده (از CLRS)

در این بخش، چند تمرین از فصل ۱۷ کتاب CLRS را با استفاده از روش پتانسیل حل می‌کنیم.

تمرین 17.3-1

مسئله: فرض کنید یک تابع پتانسیل Φ داریم که $\Phi(D_i) \ge \Phi(D_0)$ برای تمام i ها، اما $\Phi(D_0) \neq 0$. نشان دهید یک تابع پتانسیل Φ' وجود دارد که $\Phi'(D_0) = 0$ و $\Phi'(D_i) \ge 0$ و هزینه‌های سرشکن با آن یکسان هستند.

حل:
تابع پتانسیل جدید را به صورت زیر تعریف می‌کنیم:
$\Phi'(D_i) = \Phi(D_i) - \Phi(D_0)$

1. **بررسی شرایط اولیه:**
   - $\Phi'(D_0) = \Phi(D_0) - \Phi(D_0) = 0$. شرط اول برقرار است.
   - طبق فرض، $\Phi(D_i) \ge \Phi(D_0)$ است، بنابراین $\Phi(D_i) - \Phi(D_0) \ge 0$. پس $\Phi'(D_i) \ge 0$ برای تمام i ها. شرط دوم نیز برقرار است.
2. **بررسی هزینه سرشکن:**
   هزینه سرشکن جدید $\hat{c}'_i$ را محاسبه می‌کنیم:
   $\hat{c}'_i = c_i + \Phi'(D_i) - \Phi'(D_{i-1})$
   $\hat{c}'_i = c_i + (\Phi(D_i) - \Phi(D_0)) - (\Phi(D_{i-1}) - \Phi(D_0))$
   $\hat{c}'_i = c_i + \Phi(D_i) - \Phi(D_0) - \Phi(D_{i-1}) + \Phi(D_0)$
   $\hat{c}'_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$
   این عبارت دقیقاً برابر با هزینه سرشکن قبلی ($\hat{c}_i$) است. بنابراین هزینه‌های سرشکن یکسان باقی می‌مانند.

تمرین 17.3-2 (بازحل تمرین 17.1-3)

مسئله: دنباله‌ای از n عمل را روی یک ساختمان داده انجام می‌دهیم. هزینه عمل i-ام برابر i است اگر i توانی دقیق از 2 باشد، و در غیر این صورت 1 است. از روش پتانسیل برای تعیین هزینه سرشکن هر عمل استفاده کنید.

حل:
می‌خواهیم یک هزینه سرشکن ثابت $\hat{c}$ پیدا کنیم.
تابع پتانسیل: پتانسیل را طوری تعریف می‌کنیم که قبل از عملیات گران (وقتی i توان 2 است) به اندازه کافی انرژی ذخیره کرده باشیم.
$\Phi(D_i) = 2 * (i - 2^{\text{floor}(\log_2 i)})$
این تابع در واقع 2 ضربدر فاصله i از نزدیک‌ترین توان 2 کوچکتر یا مساوی آن است. $\Phi(D_0)=0$.

تحلیل هزینه‌ها:

1. **حالت اول: i توان دقیقی از 2 نیست.**
   - هزینه واقعی: $c_i = 1$.
   - در این حالت $\text{floor}(\log_2(i-1)) = \text{floor}(\log_2 i)$. بگذارید $m = 2^{\text{floor}(\log_2 i)}$.
   - $\Phi(D_i) = 2(i - m)$
   - $\Phi(D_{i-1}) = 2((i-1) - m)$
   - $\Delta\Phi = \Phi(D_i) - \Phi(D_{i-1}) = 2(i-m) - 2(i-1-m) = 2$.
   - هزینه سرشکن: $\hat{c}_i = c_i + \Delta\Phi = 1 + 2 = 3$.
2. **حالت دوم: i توان دقیقی از 2 است.**
   - هزینه واقعی: $c_i = i$.
   - در این حالت $i = 2^{\log_2 i}$ ، بنابراین $i - 2^{\text{floor}(\log_2 i)} = 0$. پس $\Phi(D_i) = 0$.
   - برای $i-1$ ، نزدیک‌ترین توان 2 کوچکتر، $i/2$ است.
   - $\Phi(D_{i-1}) = 2 * ((i-1) - 2^{\text{floor}(\log_2(i-1))}) = 2 * ((i-1) - i/2) = 2 * (i/2 - 1) = i - 2$.
   - $\Delta\Phi = \Phi(D_i) - \Phi(D_{i-1}) = 0 - (i - 2) = -i + 2$.

- هزینه سرشکن: $\hat{c}_i = c_i + \Delta\Phi = i + (-i + 2) = 2$.

نتیجه: هزینه سرشکن هر عمل حداکثر 3 است، یعنی $O(1)$.

تمرین 17.3-4

مسئله: هزینه واقعی کل اجرای n عمل پشته (PUSH, POP, MULTIPOP) چقدر است، با فرض اینکه پشته با s_0 شیء شروع شده و با s_n شیء تمام می‌شود؟

حل:
از فرمول اصلی هزینه کل استفاده می‌کنیم:
$\Sigma c_i = \Sigma \hat{c}_i - (\Phi(D_n) - \Phi(D_0))$

- $\Phi(D)$ تعداد اشیاء در پشته است. پس $\Phi(D_0) = s_0$ و $\Phi(D_n) = s_n$.
- هزینه سرشکن PUSH برابر 2 و برای POP و MULTIPOP برابر 0 است.
- فرض کنید در دنباله n عمل، n_push عمل PUSH و $n\_pop+n\_multipop$ عمل دیگر داشته باشیم.
- هزینه سرشکن کل: $\Sigma \hat{c}_i = 2 * n\_push$.
- جایگذاری در فرمول:
  $\Sigma c_i = (2 * n\_push) - (s_n - s_0)$

این عبارت، هزینه واقعی کل را بر حسب تعداد PUSH ها و تعداد عناصر اولیه و نهایی بیان می‌کند.

تمرین 17.3-5

مسئله: فرض کنید یک شمارنده با b بیت 1 شروع به کار می‌کند. نشان دهید هزینه n عمل INCREMENT برابر $O(n)$ است اگر $n = \Omega(b)$.

حل:
از تحلیل مثال شمارنده دودویی استفاده می‌کنیم.

- هزینه واقعی کل: $\Sigma c_i \leq \Sigma \hat{c}_i - (\Phi(D_n) - \Phi(D_0))$.
- هزینه سرشکن هر INCREMENT حداکثر 2 بود. پس $\Sigma \hat{c}_i \leq 2n$.
- $\Phi(D)$ تعداد بیت‌های 1 است.
- $\Phi(D_0) = b$ (تعداد بیت‌های 1 در ابتدا).
- $\Phi(D_n) = b_n$ (تعداد بیت‌های 1 در انتها).
- پس: $\Sigma c_i \leq 2n - (b_n - b) = 2n - b_n + b$.
- از آنجایی که تعداد بیت‌های 1 هرگز منفی نیست، $b_n \geq 0$.
- بنابراین: $\Sigma c_i \leq 2n + b$.
- فرض مسئله این است که $n = \Omega(b)$ ، یعنی n با نرخ b یا سریع‌تر رشد می‌کند. به عبارت دیگر، یک ثابت $c > 0$ وجود دارد که $n \geq c*b$ ، یا $b \leq n/c$.
- این را در نابرابری هزینه کل جایگذاری می‌کنیم:
  $\Sigma c_i \leq 2n + n/c = (2 + 1/c) * n$.
- از آنجایی که c یک ثابت است، کل هزینه $O(n)$ است.

موفق باشید!

صحبتی با ChatGPT

چطور تابع پتانسیل خوب انتخاب کنیم؟

🎯 هدف تابع پتانسیل چیه؟

تابع پتانسیل Φ یه چیز **اختیاری ولی هوشمندانه**ست که ما انتخابش می‌کنیم تا:

- تغییرات هزینه بین عملیات‌ها رو «صاف» کنه؛
- عملیات گرون (costly) رو با عملیات ارزون جبران کنه.

به زبان ساده:

> پتانسیل مثل یه "حساب بانکی" برای هزینه‌ست:
> در بعضی عملیات‌ها پول (هزینه) ذخیره می‌کنی، در بعضی دیگه خرج می‌کنی،
> ولی میانگین هزینه همیشه کم می‌مونه.

💡 **تعریف رسمی (خلاصه‌شده)**

برای هر حالت از داده‌ساختار D_i، یه تابع پتانسیل داریم:

$$0 \leq \Phi(D_i)$$

و برای هر عملیات i:

$$\Phi(D_{i-1}) - \Phi(D_i) + c_i = \hat{c}_i$$

که در اون:

- c_i : هزینه واقعی عملیات
- $\hat{c}_i$: هزینه سرشکن (amortized cost)

🧠 **اما سؤال اصلی: چطوری یه تابع پتانسیل خوب انتخاب کنیم؟**

۱. باید حس و شهود داشته باشه از "چیزی که هزینه رو تعیین می‌کنه"

پتانسیل باید نشون بده **چقدر "کار ذخیره‌شده" در ساختار** وجود داره — یعنی چیزی که اگه زیاد بشه، ممکنه در آینده هزینه بده.

مثلاً:

- توی **شمارنده بیتی**، بیت‌های ۱ نشون‌دهنده‌ی "کار ذخیره‌شده" هستن، چون هر بیت ۱ بالاخره یه روز باید صفر بشه (هزینه ایجاد کنه).
  👉 پس Φ = تعداد بیت‌های ۱
- توی **پشته با دوتا استک**، پتانسیل می‌تونه باشه "تعداد آیتم‌هایی که باید منتقل بشن ولی هنوز منتقل نشدن".

---

۲. باید اختلاف $\Phi(D_{i-1}) - \Phi(D_i)$ معنی‌دار باشه

یعنی وقتی یه عملیات انجام می‌دی:

- اگر عملیات "ساده" بود، پتانسیل زیاد بشه (چون کار رو انداختی برای آینده).
- اگر عملیات "سخت" بود، پتانسیل کم بشه (چون داری اون کار ذخیره‌شده رو انجام می‌دی).

به این ترتیب هزینه‌ها متعادل می‌شن.

---

۳. باید $\Phi(D_0) = 0$ و همیشه $0 \leq \Phi(D_i)$

تا تحلیل درست بمونه — چون پتانسیل منفی یعنی داری "بدهکار هزینه" می‌شی که معنی نداره.

📘 **چند مثال ساده از انتخاب تابع پتانسیل**

مثال	ایده‌ی شهودی	تابع پتانسیل پیشنهادی $\Phi(D_i)$	نتیجه
شمارنده بیتی	هر بیت ۱ بالاخره صفر میشه → "کار ذخیره‌شده"	تعداد بیت‌های ۱	هزینه سرشکن = O(1)
آرایه پویا (Dynamic Array)	وقتی آرایه نصف‌پر میشه → در آینده نیاز به resize داری	|num - size·2|	هزینه سرشکن = O(1) برای append
صف با دو استک	عناصر در استک ورودی باید در آینده به خروجی منتقل بشن	تعداد عناصر در استک ورودی	هزینه سرشکن = O(1) برای enqueue/dequeue

---

🔍 روش قدم‌به‌قدم برای پیدا کردن تابع پتانسیل

1. **بفهم عملیات‌ها چه هزینه‌ای دارن** (کدوم گرونن؟ کِی اتفاق می‌افتن؟)
2. **تشخیص بده چه چیزی در ساختار "انباشت میشه"** که باعث هزینه در آینده میشه.
3. تابع پتانسیل رو به نحوی بنویس که اون انباشت رو نشون بده.
4. بررسی کن که $\Phi(D_{i-1}) - \Phi(D_i)$ بتونه هزینه‌ها رو متعادل کنه.
5. مطمئن شو که $\Phi(D_0) = 0$ و همیشه غیرمنفیه.

💬 یه مثال تازه برای تمرین

فرض کن یه صف (queue) با دو پشته داری — یکی برای enqueue و یکی برای dequeue.
در این ساختار:

- وقتی enqueue می‌کنی → فقط push به استک اول (ارزون)
- وقتی dequeue می‌کنی → اگه استک دوم خالی باشه، کل عناصر از اولی به دومی منتقل می‌شن (گرون!)

پس تابع پتانسیل می‌تونه باشه:

$$\Phi = \text{تعداد عناصر در استک اول}$$

چون هر چی این عدد بزرگ‌تر بشه، احتمال یه عملیات گرون در آینده بیشتره.
و واقعاً این پتانسیل باعث میشه هزینه سرشکن O(1) بشه.
