ساختمان داده
دانشکده ریاضی. دانشگاه صنعتی خواجه نصیرالدین طوسی. پاییز ۱۴۰۲

۱ تحلیل سرشکنی Amortized Analysis

فرض کنید ساختمان داده D عمل A را پشتیبانی می‌کند. در تحلیل بدترین حالت، می‌خواهیم بدانیم عمل A زمان اجرایش حداکثر چقدر است. در این درس نمونه‌های زیادی را از این تحلیل دیده‌ایم. برای مثال نشان دادیم که عمل Search در درخت BST زمان اجرایش در بدترین حالت $O(h)$ است وقتی که h ارتفاع درخت است. به همین طریق نشان دادیم که عمل Insert در درخت AVL در بدترین حالت زمان اجرایش $O(\log n)$ است وقتی n تعداد رئوس درخت AVL باشد.

در تحلیل سرشکنی می‌خواهیم بدانیم که انجام m عمل متوالی (که می‌تواند ترکیبی از اعمال مختلف هم باشد) در بدترین حالت چقدر زمان می‌برد. اگر $f(n)$ زمان اجرای عمل A (در بدترین حالت) باشد، زمان اجرای دنباله‌ای از m عمل A حداکثر $mf(n)$ خواهد بود. این کران بالا می‌تواند کران بالای خوبی نباشد. برای مثال در ساختار مجموعه‌های مجزا با لیست پیوندی، دیدیم که زمان اجرای یک عمل Union در بدترین حالت $\Theta(n)$ است در حالیکه زمان اجرای $n-1$ عمل Union برابر با $\Theta(n\log n)$ است و نه $\Theta(n^2)$. در اینجا می‌گوییم که زمان اجرای سرشکنی عمل Union برابر با $O(\log n)$ است.

در این بخش می‌خواهیم مواردی دیگر از این حالت را مثال بزنیم که زمان اجرای دنباله‌ای از m عمل بسیار کمتر از کران بالای $mf(n)$ است. علاوه بر این، سه تکنیک مختلف برای اثبات چنین کران بالاهایی را ذکر می‌کنیم. برای این منظور دو مثال ساده را بررسی می‌کنیم.

- Multi-pop Stack

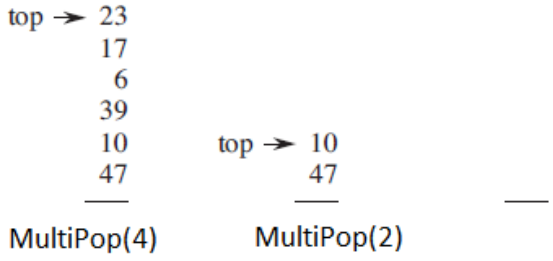
این ساختار داده یک توسعه ساده از پشته معمولی است و سه عمل اصلی را پشتیبانی می‌کند.

۱. Push(S,x)
آیتم x را روی پشته S قرار می‌دهد. برای سادگی، فرض کنید زمان اجرای این عمل 1 باشد.

۲. Pop(S)
عنصر بالای پشته S را در صورت وجود خارج می‌کند و برمی‌گرداند. فرض کنید زمان اجرای این عمل $\min\{|S|,1\}$ باشد.

۳. Multi-Pop(S,k)
به تعداد k عنصر بالای پشته S را خارج می‌کند. اگر تعداد عناصر پشته S از k کمتر باشد، تعداد عناصری که خارج می‌شوند $|S|$ خواهد بود. اینجا $|S|$ تعداد عناصر پشته است. زمان اجرای این عمل برابر با $\min\{|S|,k\}$ است.

اینجا می‌خواهیم بدانیم m فراخونی از اعمال مختلف پشته در مجموع پیچیدگی زمانی اش چقدر است.



- Binary Counter

این ساختار داده ساده متشکل از یک شمارنده A با k بیت است و تنها عمل آن Increment است که یک واحد به مقدار شماره می‌افزاید.

$$A \leftarrow A + 1 \qquad (Increment)$$

اگر همه بیت های شماره 1 باشد، فراخوانی Increment همه بیت ها را صفر خواهد کرد (انگار شمارنده reset شده است.) بدین ترتیب روشن است که اجرای عمل Increment در بدترین حالت زمان اجرایش k است.

Counter value	A[7]	A[6]	A[5]	A[4]	A[3]	A[2]	A[1]	A[0]
0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	1
2	0	0	0	0	0	0	1	0
3	0	0	0	0	0	0	1	1
4	0	0	0	0	0	1	0	0
5	0	0	0	0	0	1	0	1
6	0	0	0	0	0	1	1	0
7	0	0	0	0	0	1	1	1
8	0	0	0	0	1	0	0	0
9	0	0	0	0	1	0	0	1
10	0	0	0	0	1	0	1	0
11	0	0	0	0	1	0	1	1
12	0	0	0	0	1	1	0	0
13	0	0	0	0	1	1	0	1
14	0	0	0	0	1	1	1	0
15	0	0	0	0	1	1	1	1
16	0	0	0	1	0	0	0	0

می‌خواهیم بدانیم که اجرای m دنباله متوالی از Increment زمان اجرایش چقدر خواهد بود؟ بطور پیش فرض، مقدار اولیه شمارنده صفر است.

۲ روشهای مختلف برای تحلیل سرشکنی

- **روش تجمعی Aggregate Method** در این روش یک کران بالای برای زمان m عمل متوالی بدست می‌آوریم. نمونه این کار را قبلا برای ساختار داده مجموعه‌های مجزا انجام دادیم. آنجا نشان دادیم که اجرای $n-1$ عمل Union در مجموع زمانش نمی‌تواند از $n \log n$ بیشتر باشد (استدلال کردیم که هر اشاره‌گر به تعداد حداکثر $\log n$ بار بروزرسانی می‌شود.)

۱. می‌توانیم از روش تجمعی استفاده کنیم و یک تحلیل سرشکنی برای ساختار داده پشته MultiPop ارائه دهیم. یک مشاهده خیلی ساده این است که تعداد Pop های موفق (چه آنهایی که داخل عمل MultiPop انجام می‌شوند و چه آنهایی که موقع فراخوانی Pop انجام می‌شوند) نمی‌تواند از تعداد Push ها بیشتر باشد. لذا با فرض اینکه زمان اجرای Pop های ناموفق صفر است، زمان اجرای m عمل حداکثر

$$(2n_{push}) \leq 2m$$

خواهد بود. اینجا n_{push} تعداد فراخوانی های Push در میان m عمل است.

۲. برای تحلیل شمارنده باینری با استفاده از روش تجمعی، دقت کنید زمان انجام m عمل Increment برابر با تعداد دفعاتی است که یک بیت در طول اجرا تغییر می‌کند. حال بیایید تعداد دفعاتی که هر بیت تغییر می‌کند را بشماریم. به شکل بالا دقت کنید. بیت اندیس صفر $A[0]$ هر بار تغییر می‌کند (m بار). بیت بعدی هر دو بار تغییر می‌کند ($\lfloor m/2 \rfloor$). بدین ترتیب بیت اندیس i هر 2^i بار تغییر می‌کند ($\lfloor m/2^i \rfloor$). لذا تعداد کل تغییرات برابر است

$$\sum_{i=0}^{k} \lfloor m/2^i \rfloor < m \sum_{i=0}^{\infty} 1/2^i = 2m$$

- **روش حسابداری Accounting Method** در این روش زمان لازم برای انجام یک عمل را مانند هزینه‌ای که با پول پرداخت می‌شود تصور می‌کنیم. لذا اگر یک عمل t واحد زمان نیاز داشته باشد مانند این است که به t واحد پول نیاز داشته باشد. برای اثبات یک کران بالا برای زمان m عمل (میزان پولی که باید برای انجام m عمل پرداخت شود) بدین صورت عمل می‌کنیم. فرض کنید یک حساب پولی داریم و هر زمان کاری انجام می‌دهیم هزینه‌اش را از حساب برمی‌داریم. مانده حساب باید همیشه غیر منفی باشد. برای هر نوع عمل یک هزینه سرشکنی معین می‌کنیم. مثلا برای انجام عمل A باید ۱۰ تومان به حساب ریخته شود (هزینه سرشکنی A) و برای عمل B باید ۵ تومان پول به حساب ریخته شود (هزینه سرشکنی B). دقت کنید هزینه سرشکنی یک عمل ممکن است از هزینه واقعی آن کمتر یا بیشتر باشد. مهم این است که در حساب پول کافی داشته باشیم تا هزینه واقعی هر عمل را از آن پرداخت کنیم. اگر مانده حساب هیچ وقت منفی نباشد، یعنی اینکه کل دریافتی ها از هزینه واقعی اعمال کمتر نبوده. لذا کل دریافتی ها یک کران بالا برای زمان مورد نظر خواهد بود.

ترفند کلی این است که برای اعمال با هزینه کم پول بیشتر پرداخت کنیم تا در حساب ذخیره کنیم برای پرداخت هزینه اعمال پرهزینه. اگر به یاد داشته باشید مشابه این عمل را برای تحلیل آرایه پویا انجام دادیم. آنجا برای هر Append معمولی در آرایه که هزینه واقعی اش ۱ واحد بود، ۲ واحد پرداخت کردیم تا موقع دو برابر شدن اندازه آرایه پول لازم برای انجام آن را داشته باشیم.

۱. برای تحلیل سرشکنی مثال پشته MultiPop از روش حسابداری می‌توانیم استفاده کنیم. برای سادگی فرض کنید هزینه واقعی ،Push هزینه واقعی Pop موفق و ناموفق برابر با 1 واحد است.
اینجا باید هزینه سرشکنی هر عمل را مشخص کنیم. هزینه سرشکنی Push را 2 تومان تعریف می‌کنیم. قرار می‌دهیم

$$a_{push} = 2$$

و هزینه سرشکنی Pop و MultiPop را 0 تومان تعریف می‌کنیم.

$$a_{singlepop} = 0, \qquad a_{multipop} = 0$$

روشن است که با این تعاریف و با توجه به اینکه هر عمل Pop موفق قبلش یک Push موفق بوده لذا همیشه باید در حساب پول کافی برای پرداخت هزینه را داشته باشیم. در واقع تک واحد پولی که موقع انجام Push ذخیره می‌کنیم موقع انجام Pop موفق خرج می‌کنیم. لذا نتیجه می‌گیریم زمان دنباله m عمل برای این ساختار داده، حداکثر $2m$ است.

۲. در مورد شمارنده باینری، فرض کنید هر بار که یک بیت را به 1 تغییر دادیم (set کردن) 2 تومان پرداخت کنیم. یک تومانش صرف هزینه خود عمل می‌شود و یک تومانش در حساب برای وقتی که همان بیت را به 0 تغییر می‌دهیم (موقع reset کردن) نگه می‌داریم. با توجه به اینکه شماره از صفر شروع می‌شود، در همه حال پول کافی در حساب داریم. از آنجا که در هر عمل Increment حداکثر یک بار عمل set انجام می‌شود هزینه سرشکنی Increment برابر با 2 تومان خواهد بود.

$$a_{increment} = 2$$

پس انجام m فراخوانی Increment زمانش حداکثر برابر با $2m$ است.

- **روش تابع پتانسیل Potenital Function Method** این روش را می‌توان بصورت تعمیمی از روش حسابداری در نظر گرفت. در روش حسابداری، دیدیم که اعمال کم هزینه اعتبار حساب را بالا می‌برند و اعمال پرهزینه از اعتبار حساب

خرج می‌کنند. یک تعبیر فیزیکی از حسابی که نگه می‌داریم می‌تواند پتانسیل باشد. انگار یک سیستم داینامیک داریم که پتانسیلش (یا انرژی اش) با اعمال کم هزینه بالا می‌رود و با اعمال پرهزینه پایین می‌آید. حال اینجا به جای اینکه از یک حساب مجزا استفاده کنیم، از خود ساختار داده برای نشان دادن میزان اعتبار موجود استفاده می‌کنیم. باید طوری پتانسیل را تعریف کنیم که مانند حساب عمل کند. فرض کنید D_0 وضعیت اولیه ساختار داده باشد و D_i وضعیت ساختار داده بعد از انجام عمل i ام باشد. تابع پتانسیل $\Phi : D \to \mathbb{R}$ را تعریف می‌کنیم که وابسته به وضعیت ساختار داده است. هزینه سرشکنی عمل i ام را تعریف می‌کنیم

$$a_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$$

وقتی که c_i هزینه واقعی عمل i ام باشد. با این تعریف داریم

$$\sum_{i=1}^{m} a_i = \sum_{i=1}^{m} c_i + \Phi(D_m) - \Phi(D_0)$$

لذا اگر برای هر m داشته باشیم $\Phi(D_m) - \Phi(D_0) \geq 0$ آنگاه می‌توانیم بگوییم $\sum_{i=1}^{m} a_i$ یک کران بالا معتبر برای $\sum_{i=1}^{m} c_i$ است که همان هزینه واقعی است. بطور خاص اگر تابع پتانسیل طوری باشد که $\Phi(D_0) = 0$ کافی است که همه حال داشته باشیم $\Phi(D_i) \geq 0$. توجه کنید این مانند است که حساب نباید هیچ وقت منفی شود.

دقت کنید می‌خواهیم هزینه سرشکنی هر عمل کم باشد، لذا وقتی c_i بالاست، میزان افزایش پتانسیل یعنی

$$\Phi(D_i) - \Phi(D_{i-1})$$

باید منفی شود تا با a_i کوچک از پس c_i بزرگ بربیاییم. اینجا مانند است که از پتانسیل خرج کرده‌ایم.

توجه کنید استفاده از روش تابع پتانسیل امر ساده‌ای نیست و گاها مستلزم ابتکار فراوان است. شاید سوال پیش بیاید چرا از همان روش حسابداری که شهودش راحتتر است استفاده نکنیم؟ در جواب می‌توان گفت اگر در موقعیتی هستیم که راحت می‌توان از روش حسابداری استفاده کرد نیازی نداریم خود را با پتانسیل خود به زحمت بیاندازیم. اما باید توجه داشته باشیم که در روش حسابداری باید برای هر نوع عمل یک هزینه سرشکنی ثابت مشخص کنیم و این فارغ از وضعیت ساختار داده است. روش پتانسیل چهارچوبی را فراهم می‌کند که هزینه سرشکنی هم (مانند هزینه واقعی) متغیر باشد و به نوعی وابسته به وضعیت ساختار داده در زمان انجام عمل باشد. این انعطاف بیشتری را فراهم می‌کند و در عین حال نیازمند فهم عمیق‌تر از داینامیک ساختار داده است.

۱. مثال پشته را با روش تابع پتانسیل بررسی می‌کنیم. اگر از ایده روش حسابداری برای تحلیل این مثال الهام بگیریم، از آنجا که اضافه شدن عناصر به پشته اعتبار حساب را بالا می‌بردند، می‌توانیم پتانسیل ساختار داده را برابر با تعداد عناصر پشته تعریف کنیم. پس تعریف می‌کنیم.

$$\Phi(D) = \text{ number of items in the stack}$$

روشن است که $\Phi(D_0) = 0$ و همینطور داریم $\Phi(D) \geq 0$.

اگر عمل iام Push باشد داریم $c_i = 1$ و $\Phi(D_i) = \Phi(D_{i-1}) + 1$ پس در این حالت $a_i = 2$.

اگر عمل iام MultiPop(S,k) باشد داریم $c_i = \min\{|S|, k\}$ و $\Phi(D_i) = \Phi(D_{i-1}) - \min\{|S|, k\}$ پس در این حالت $a_i = 0$.

اگر عمل i ام Pop باشد، بطور مشابه داریم $a_i = 0$.

پس در هر حالت داریم $a_i \leq 2$. لذا

$$\sum_{i=1}^{m} a_i \leq 2m$$

۲. شمارنده باینری را هم می‌توان با تابع پتانسیل تحلیل کرد. باز هم از روش حسابداری الهام می‌گیریم و تعریف می‌کنیم

$$\Phi(D) = \text{ number of 1s in the counter}$$

روشن است که $\Phi(D_0) = 0$ و همینطور داریم $\Phi(D) \geq 0$.
فرض کنید عمل i ام به تعداد p بیت را reset کند. دقت کنید هر Increment حداکثر یک صفر را به 1 تغییر می‌دهد. لذا داریم

$$p \leq c_i \leq p + 1$$

و

$$-p \leq \Phi(D_i) - \Phi(D_{i-1}) \leq -p + 1$$

. پس $a_i \leq 2$. در نتیجه

$$\sum_{i=1}^{m} a_i \leq 2m$$