Accounting method

- برای هر insertion آیتم x ، 3$ شارژ می‌پردازیم:
  - 1$ برای خود عمل insertion آیتم x
  - 1$ برای آیتم x که بعداً جابجا می‌شود.
  - 1$ برای جابجایی یک آیتم دیگر.

- فرض کنید که هم اکنون expand انجام داده‌ایم بنابراین size = m است و بعد از expansion size = 2m شده است.
- فرض کنید که expansion تمام credit ها را صرف کند، بنابراین درست بلافاصله پس از expansion، credit وجود ندارد.
- جدول دوباره expand می‌شود، اگر m تا insertion دیگر رخ دهد.
- هر insertion، 1$ بر روی آیتم insert شده قرار می‌دهد و 1$ بر روی یک آیتم موجود در جدول.
- 2m$ credit برای expansion بعدی ذخیره می‌شود زمانی که باید 2m آیتم جابجا شود که کافی است و در اصل credit هم باقی نمی‌ماند.

Potential method

در این روش، $\phi(T)$ برابر با $2.num[T] - size[T]$ در نظر می‌گیریم

- در ابتدا، $num = size = 0 \Rightarrow \phi = 0$
- درست بعد از expansion: $size = 2.num \Rightarrow \phi = 0$
- درست قبل از expansion:

$size = num \Rightarrow$

$\phi = num \Rightarrow$ have enough potential to pay for moving all items

نیاز است که $\phi \geq 0$ باشد (همیشه).

همیشه داریم که

$$size \geq num \geq \frac{1}{2} . size \Rightarrow$$

$$2 . num \geq size \Rightarrow$$

$$\phi \geq 0$$

هزینه سرشکن برای i امین عملیات:

num_i = num after ith operation

$size_i$ = size after ith operation

ϕ_i = ϕ after ith operation

اگر هیچ expansion رخ نمود:

$$size_i = size_{i-1}$$

$$num_i = num_{i-1} + 1$$

$$c_i = 1$$

و داریم:

$$\hat{c_i} = c_i + \phi_i - \phi_{i-1}$$

$$= 1 + (2 . num_i - size_i) - (2 . num_{i-1} - size_{i-1})$$

$$= 1 + (2 . num_i - size_i) - (2(num_i - 1) - size_i)$$

$$= 1 + 2$$

$$= 3$$

در صورت expansion

$$size_i = 2\, size_{i-1}$$

$$size_{i-1} = num_{i-1} = num_i - 1$$

$$c_i = num_{i-1} + 1 = num_i$$

و داریم

$$\hat{c_i} = c_i + \phi_i + \phi_{i-1}$$

$$= num_i + (2.num_i - size_i) - (2\, num_{i-1} - size_{i-1})$$

$$= num_i + (2num_i - 2(num_i - 1)) - (2(num_i - 1) - (num_i - 1))$$

$$= num_i + 2 - (num_i - 1)$$

$$= 3$$

Dynamic Programming

- یک روش موثر و کارآمد برای حل مسائل بهینه سازی و جستجو است.
- دو ویژگی زیرمسئله های هم پوشان و زیرساخت های بهینه.
- چارچوب استانداردی برای فرموله کردن مسئله وجود ندارد.
- شبیه روش تقسیم و حل، جواب مسئله را با ترکیب کردن جواب زیر مسئله ها بدست می آورد.
- برنامه ریزی پویا بهتر است در مسائلی استفاده کنیم که زیرمسائل آنها مستقل نیست.
- الگوریتم برنامه ریزی پویا زیرمسئله ها را یکبار حل و جواب آنها را در یک جدول ذخیره می کند و با این کار از تکرار اجرای زیرمسائل در زمانی که مجددا به جواب آن احتیاج داریم جلوگیری می شود.
- جواب یک مسئله بهینه ساز را بتوان از ترکیب جوابهای بهینه زیرمسئله های آن بدست آورد
- زیرساختار بهینه، یعنی جواب هر زیرمسئله از مسئله اصلی خود بهینه باشد.
- اگر زیرمسئله ها هم پوشان نباشد به آن تقسیم و حل می گویند. مرتب سازی ادغامی و سریع هر دو تقسیم و حل هستند.
- زیرمسئله های هم پوشان به معنای کوچک بودن فضای زیرمسئله هاست، به این معنا که هر الگوریتم بازگشتی برای حل این مسئله باید به جای ایجاد زیرمسئله های جدید، زیرمسئله های تکراری را بارها و بارها حل کند برای مثال فرمول بازگشتی دنباله فیبوناچی

$$F_i = F_{i-1} + F_{i-2}$$