

حل تمرین سری سوم (صف و لیست پیوندی)

ساختمان داده ها و الگوریتم

سوال ۱.

ارشد ۹۳

برای ساخت یک صف Q از دو پشته S_1 و S_2 استفاده می‌کنیم. برای درج x در انتهای Q ، عمل $Push(S_1, x)$ را انجام می‌دهیم. برای حذف یک عنصر از ابتدای Q ، اگر S_2 خالی نباشد، عمل $Pop(S_2)$ را انجام می‌دهیم. در غیر این صورت، تمامی عناصر S_1 را به ترتیب Pop کرده و $Push(S_2)$ می‌کنیم. اکنون عمل Pop بر روی S_2 عنصر ابتدایی Q را بر می‌گرداند.

اگر بر روی Q که در ابتدا خالی است، ۱۰۰ عمل صورت گیرد، حداکثر هزینه چه مقدار خواهد بود؟

(۱) 150

(۲) 151

(۳) 199

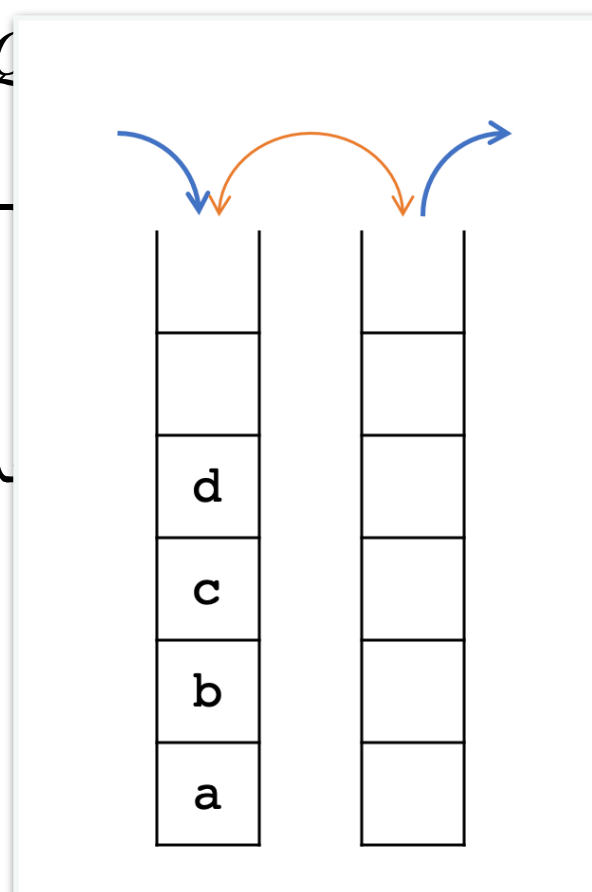
(۴) 200

سوال ۱.

ارشد ۹۳

برای ساخت یک صف Q از دو پشته S_1 و S_2 استفاده می‌کنیم. برای درج x در انتهای Q ، عمل $Push(S_1, x)$ را انجام می‌دهیم. برای حذف x از Q ، اگر S_2 خالی نباشد، عمل $Pop(S_2)$ را انجام می‌دهیم. در غیر این صورت، عمل $Pop(S_1)$ را انجام می‌دهیم و سپس $Push(S_2, x)$ را انجام می‌دهیم. اگر S_1 خالی باشد، عمل $Pop(S_1)$ را انجام می‌دهیم و سپس $Push(S_2, x)$ را انجام می‌کنیم.

مقدار کمترین هزینه چقدر خواهد بود؟



(۱) 150

(۲) 151

(۳) 199

(۴) 200


سوال ۱.


گزینه ۳


- هزینه $Push$ در هر شرایطی $O(1)$ است.
- هزینه Pop در صورتی که S_2 خالی نباشد، $O(1)$ است.
- هزینه Pop در صورتی که S_2 خالی باشد، $O(n_{S_1})$ است.
- پس باید سعی کنیم S_2 را تا جای ممکن خالی نگه داریم و سر آخر Pop کنیم:

$Push, Push, Push, \dots, Pop$

$$T(100) = 99 \times 1 + 99 + 1$$


 $Push$


انتقال از S_1
به S_2


 Pop

سوال ۲.

دکتری ۹۵

داده ساختار صف با سه عملیات افزودن به ابتدای صف، حذف از انتهای صف و استخراج عنصر کمینه را در نظر بگیرید.

بهترین پیاده‌سازی ممکن برای این داده ساختار هر یک از سه عملیات فوق را به صورت «سرشکن» در چه زمانی پشتیبانی می‌کند؟

۱) هر سه عملیات $O(1)$

۲) هر سه عملیات $O(\log n)$

۳) درج و حذف $O(1)$ و استخراج کمینه $O(\log n)$

۴) درج و حذف $O(\log n)$ و استخراج کمینه $O(n)$

سوال ۲.

دکتری ۹۵

داده ساختار صف با سه عملیات افزودن به ابتدای صف، حذف از انتهای صف و استخراج عنصر کمینه را در نظر بگیرید.

شکن» در

| | | | | | | | |
|---------|--|--|---|----|----|----|---|
| | | | 5 | 11 | 2 | 4 | 9 |
| POP | | | | 5 | 11 | 2 | 4 |
| POP MIN | | | | | 5 | 11 | 4 |
| PUSH 6 | | | | 6 | 5 | 11 | 4 |

بهترین پی

چه زمانی

(۱) هر سه

(۲) هر سه

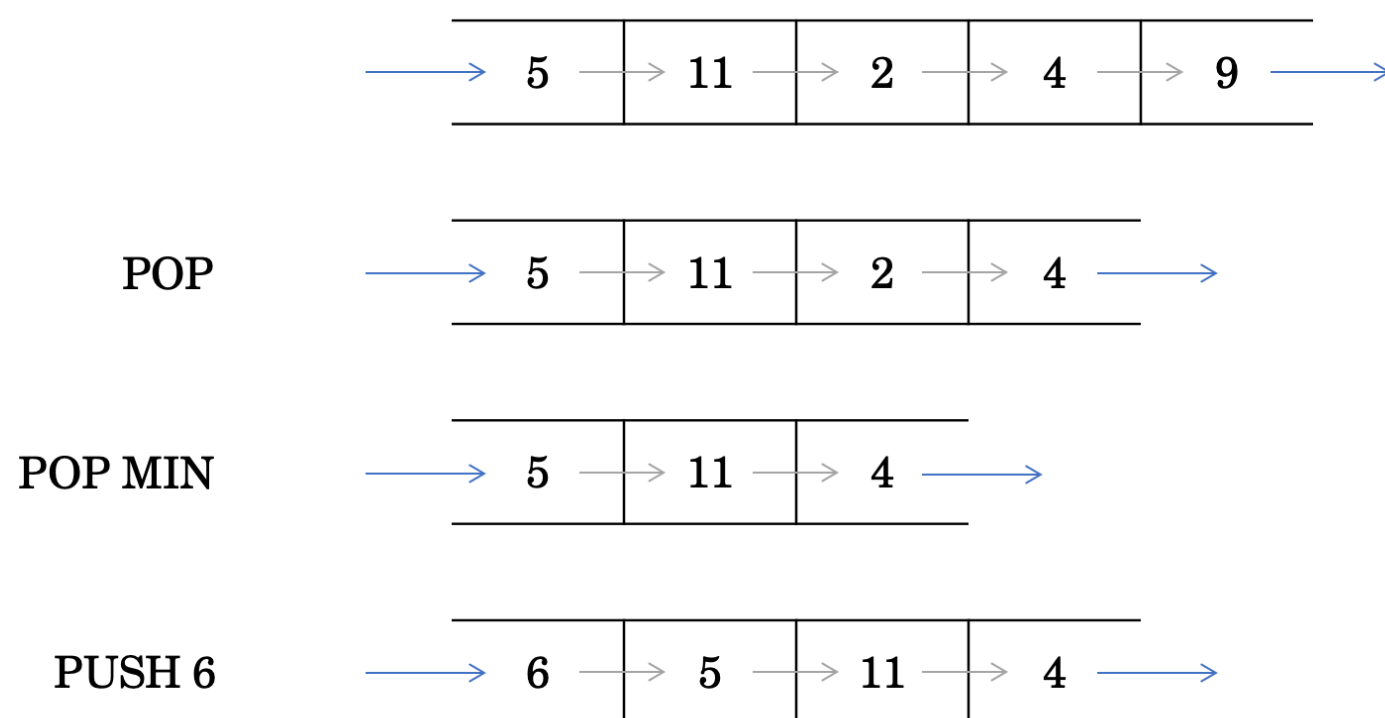
(۳) درج و

(۴) درج و

سوال ۲.

دکتری ۹۵

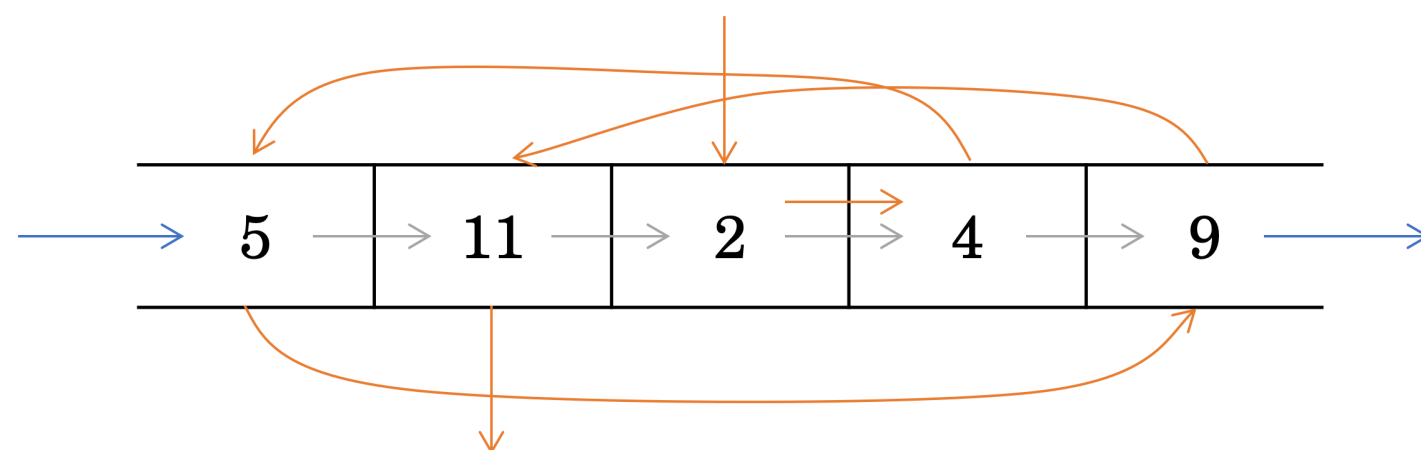
فرض کنید که صف ما با یک لیست پیوندی دوطرفه پیاده‌سازی شده باشد.
بنابراین هر خانه از یک دوتایی $(key, *next)$ تشکیل شده است و برای لیست نیز یک اشاره گر $begin$ داریم.



سوال ۲.

گزینه ۱

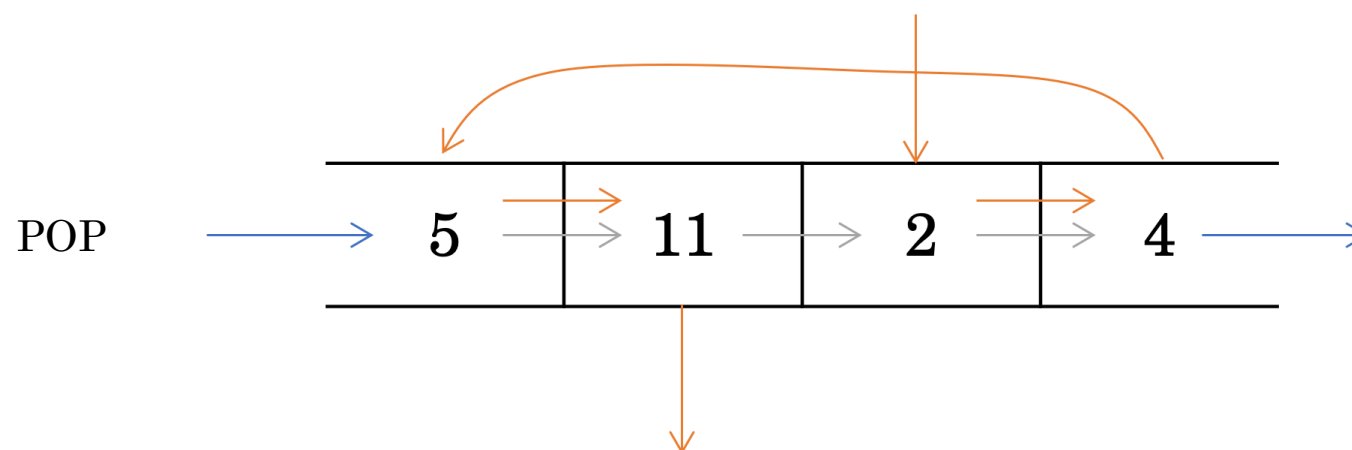
معماری خانه‌ها را به $(key, *next, *next\ min)$ تغییر می‌دهیم.
حالا به خانه کمینه از $O(1)$ دسترسی داریم و برای حذف و درج گره جدید هم مشکلی نداریم.



سوال ۲.

گزینه ۱

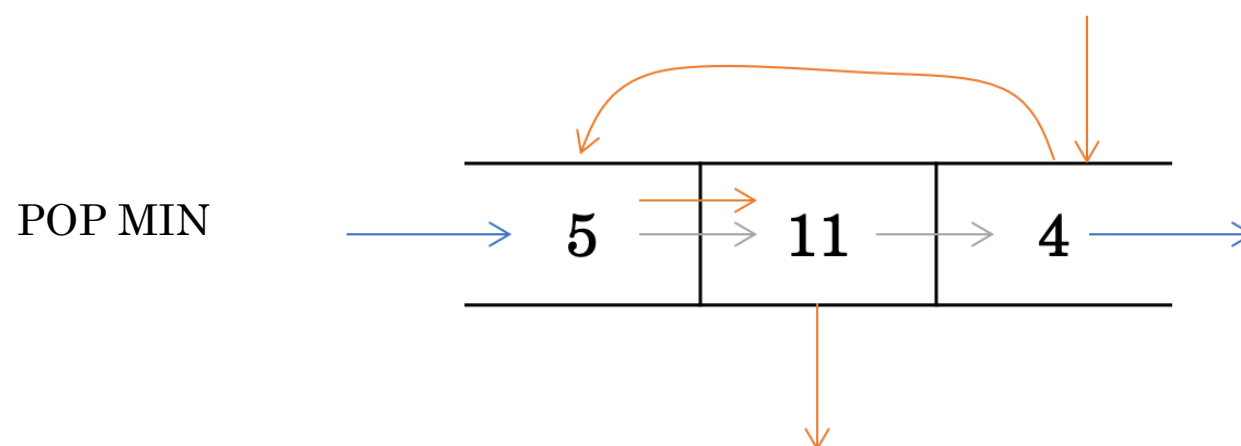
معماری خانه‌ها را به $(key, *next, *next\ min)$ تغییر می‌دهیم.
حالا به خانه کمینه از $O(1)$ دسترسی داریم و برای حذف و درج گره جدید هم مشکلی نداریم.



سوال ۲.

گزینه ۱

معماری خانه‌ها را به $(key, *next, *next\ min)$ تغییر می‌دهیم.
حالا به خانه کمینه از $O(1)$ دسترسی داریم و برای حذف و درج گره جدید هم مشکلی نداریم.



سوال ۲.

گزینه ۱

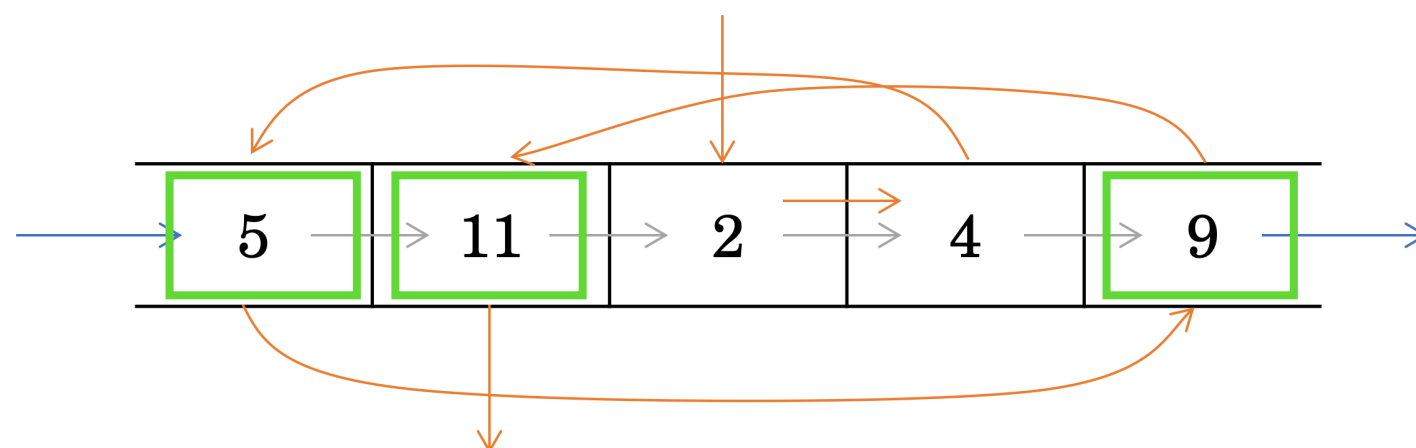
برای عملیات حذف عادی و حذف کمینه که مشکلی نداشتیم.
ادعا می‌کنیم که برای عملیات درج به همان سبک عمومی نیز مشکلی نداریم.

```
def insert_in_minQ(value):  
    x=Q.min()  
    while (Q[x].nextmin>=value):  
        x=(Q[x].nextmin).nextmin  
    (Q.nextmin).nextmin=value  
    Q.nextmin=value
```

سوال ۲.

گزینه ۱

برای اثبات نیز کافیت همانند تحلیل عملیات درج در لیست پرشی تصادفی عمل کنیم.
تابع پتانسیل را «تعداد اشاره گر های nextmin در زیر لیست که بعد از x می آید» تعریف می کنیم:

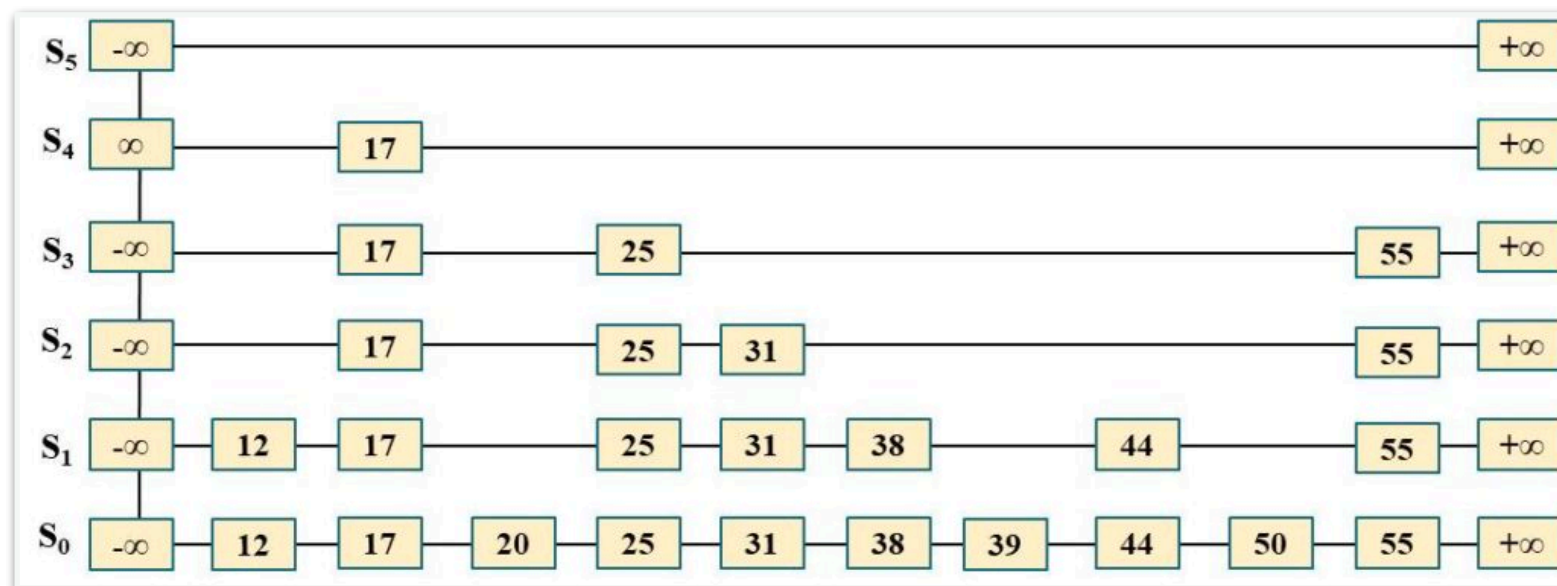


مثلا زیر لیست ۵ (در لیست اشاره گر های nextmin) با رنگ سبز نشان داده شده است.

سوال ۲.

گزینه ۱

برای اثبات نیز کافیت همانند تحلیل عملیات درج در لیست پرشی تصادفی عمل کنیم.
تابع پتانسیل را «تعداد اشاره گر های nextmin در زیر لیست که بعد از x می آید» تعریف می کنیم:



سوال ۳.

ارشد ۹۴

فرض کنید صف Q با یک آرایه‌ی حلقوی به اندازه‌ی m پیاده‌سازی شده است که اندیس‌های آن از صفر تا $m - 1$ است و عناصر آن به صورت چرخه‌ای و در جهت ساعتگرد ذخیره شده‌اند. مولفه‌های $front(Q)$ و $rear(Q)$ به ترتیب اندیس اولین عنصر و عنصر بعد از آخرین عنصر صف را ذخیره می‌کنند. تعداد عناصر داخل صف و شرط پر بودن صف کدام گزینه زیر است؟

$$(۱) \quad rear(Q) - front(Q) + 1 \mod m \quad \text{و} \quad front(Q) = rear(Q)$$

$$(۲) \quad rear(Q) - front(Q) \mod m \quad \text{و} \quad front(Q) = rear(Q)$$

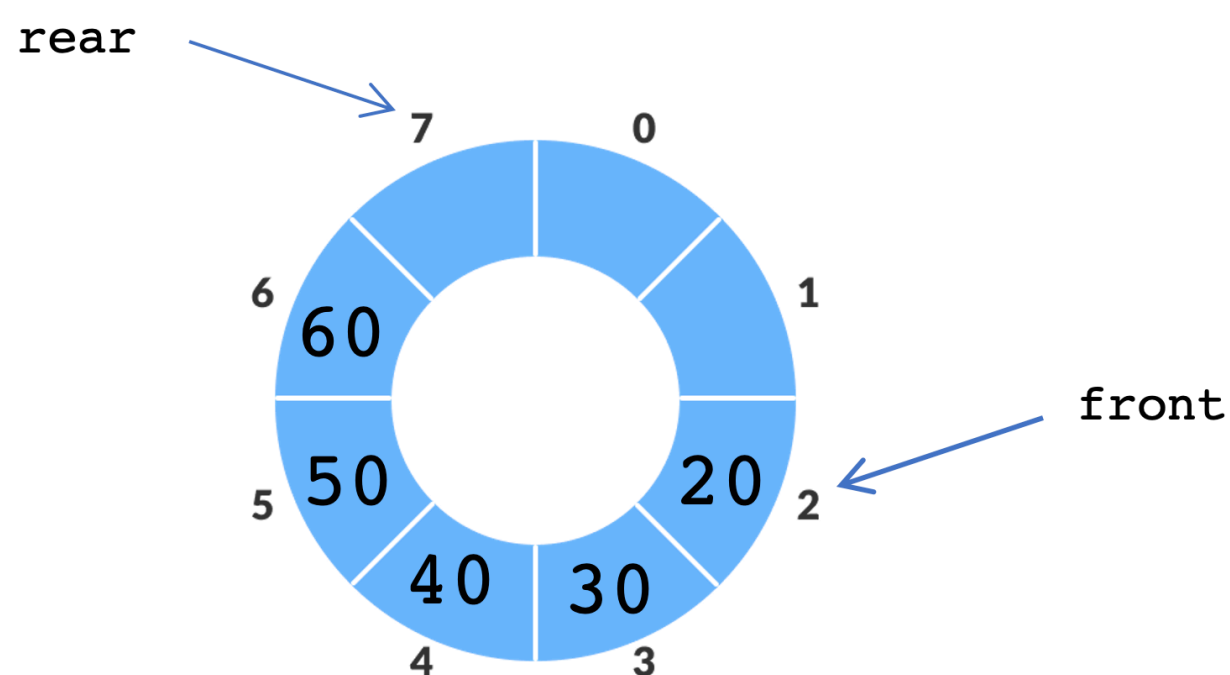
$$(۳) \quad rear(Q) - front(Q) + 1 \mod m \quad \text{و} \quad front(Q) = rear(Q) + 1 \mod m$$

$$(۴) \quad rear(Q) - front(Q) \mod m \quad \text{و} \quad front(Q) = rear(Q) + 1 \mod m$$

سوال ۳.

گزینه ۴

ندیس‌های آن از صفر
شده‌اند. مولفه‌های
مفر را ذخیره می‌کنند.



فرض کنید صف Q با

تا $m - 1$ است و عنا

$rear(Q)$ و $front(Q)$

تعداد عناصر داخل صف

$(Q) + 1 \mod m$ (۱)

$front(Q) \mod m$ (۲)

$(Q) + 1 \mod m$ (۳)

$front(Q) = rear(Q) + 1 \mod m$ و $rear(Q) - front(Q) \mod m$ (۴)

سوال ۴.

ارشد ۹۴

یک لیست پیوندی خطی با دو اشاره گر به ابتدا و انتهای آن در نظر بگیرید. کدام یک از اعمال زیر وابسته به طول لیست می باشد؟

(۱) حذف عنصر از ابتدا لیست

(۲) اضافه کردن عنصر به ابتدا لیست

(۳) اضافه کردن عنصر به انتهای لیست

(۴) حذف عنصر از انتهای لیست

سوال ۴.

گزینه ۴

یک لیست پیوندی خطی با دو اشاره گر به ابتدا و انتهای آن در نظر بگیرید. کدام یک از اعمال زیر وابسته به طول لیست می باشد؟



(۱) حذف

(۲) اضافه کردن

(۳) اضافه کردن

(۴) حذف

سوال ۵.

ارشد ۹۳

روی لیست پیوندی و دوسویه ی Q که عناصر آن عدد هستند و اشاره گر به عنصر اول و آخر آن را داریم، اعمال زیر تعریف شده اند:

$Delete(k)$: k عنصر ابتدای Q را به ترتیب حذف می کند.

$Append(C)$: عنصر آخر Q را نگاه می کند، اگر مقدارش از C بیشتر بود آن را حذف می کند. این کار را تکرار می کند تا عنصر انتهایی کمتر یا مساوی C شود (یا Q تهی شود). در آن صورت عنصر C را به انتهای صف درج می کند.

اگر دنباله ای از n تا از این دو عمل را با ترتیب دلخواه روی یک لیست تهی Q انجام دهیم. مجموع کل هزینه ها به کدام گزینه زیر نزدیک تر است؟

(۱) $n - k$

(۲) n

(۳) $2n$

(۴) $3n$

سوال ۵.

ارشد ۹۳

روی لیست پیوندی و دوسویه‌ی Q که عناصر آن عدد هستند و اشاره‌گر به عنصر اول و آخر آن را داریم، اعمال زیر تعریف شده‌اند:

$Delete(k)$: k عنصر ابتدای Q را به ترتیب حذف می‌کند.

$Append(C)$: عنصر آخر Q را نگاه می‌کند، اگر مقدارش از C بیشتر بود آن را حذف می‌کند. این کار را تکرار می‌کند تا

گزینه‌ها به

| | | | | | | | | | | |
|-----------|----|----|----|----|----|----|---|----|----|----|
| | 11 | 23 | 62 | 45 | 91 | 83 | 3 | 32 | 74 | 15 |
| Append(4) | 11 | 23 | 62 | 45 | 91 | 83 | 3 | 4 | | |
| Delete(3) | | | | 45 | 91 | 83 | 3 | 4 | | |

عنصر انتها

اگر دنباله

کدام گزین

(۱) $n - k$

(۲) n

(۳) $2n$

(۴) $3n$

سوال ۵.

چشم ها را باید شست، جور دیگر باید دید.

سهراب سپهری

● هزینه عمل $Delete(k)$ وابسته به تعداد عناصر موجود n_t و k هست، که هر دو متغیر اند!

● $O(\min\{k, n_t\})$

● هزینه عمل $Append(C)$ هم وابسته به مقدار C و همچنین تعداد عناصر موجود n_t هست!

● $O(\min\{C, n_t\})$

● خب حالا چجوری مجموع هزینه اجرای n عمل رو بدست بیاریم؟

سوال ۵.

گزینه ۳

برای حساب کردن مجموع هزینه، بجای محاسبه هزینه هر عمل و کثافت بازی (مثلا میانگین اینا)، میتونیم هزینه رو برای هر عنصر از سیستمون در نظر بگیریم و مجموع رو برای اونها حساب کنیم.

- هر عنصر C توسط $Append$ یک بار اضافه می‌شه.
- هر عنصر C یک بار و فقط یک بار توسط $Append$ یا $Delete$ حذف می‌شه.

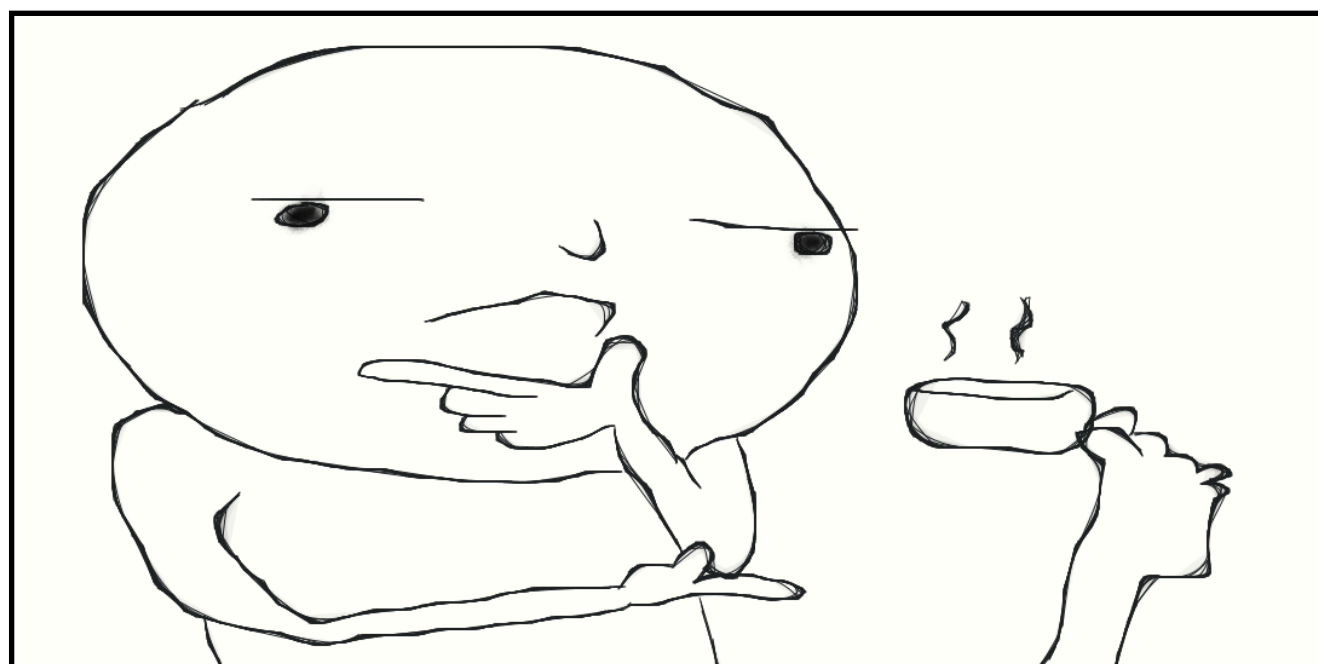
بنابراین برای هر عنصر حداکثر ۲ عمل و حداقل ۱ عمل انجام میشه.

این یعنی کلا به طور متوسط $2n$ عملیات انجام می‌دیم.

سوال؟

ای بی حافظه شده پس از نوبت‌ها شوک برقی!

رضا براهنی





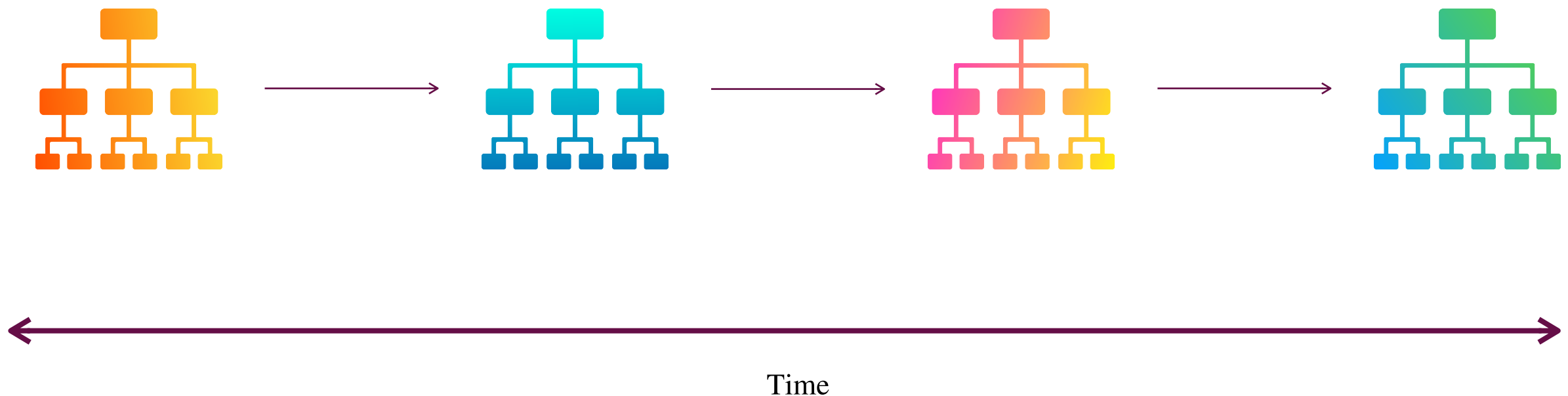
ساختمان‌های داده مانا

ساختمان‌های داده و الگوریتم

ساختمان داده‌های مانا

سفر در زمان

فرض کنید یک ساختمان داده با دو نوع عملیات پرسش و بروزرسانی داریم، چگونه می‌توان در مورد نسخه‌های قدیمی و قبل از بروزرسانی نیز عملیات پرسش را انجام داد؟



کاملاً مانا و تا اندازه‌ای مانا

Partial versus full persistence

ترجمه شده توسط مترجم گوگل

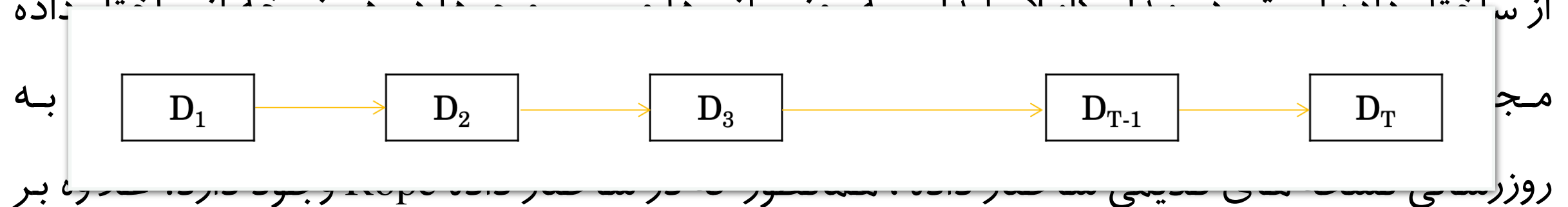
در مدل ماندگاری جزئی ، یک برنامه نویس ممکن است هر نسخه قبلی از ساختار داده را پرس و جو کند ، اما ممکن است فقط آخرین نسخه را به روز کند. این به معنی یک ترتیب خطی در بین هر نسخه از ساختار داده است. در مدل کاملاً پایدار ، به روزرسانی ها و پرس و جوها در هر نسخه از ساختار داده مجاز هستند. در برخی موارد ، ممکن است اجازه داده شود مشخصات عملکرد پرس و جو یا به روزرسانی نسخه های قدیمی ساختار داده ، همانطور که در ساختار داده Rope وجود دارد. علاوه بر این ، اگر علاوه بر پایدار بودن کامل ، دو نسخه از ساختار داده یکسان بتوانند با هم ترکیب شوند و یک نسخه جدید را ایجاد کنند که همچنان کاملاً پایدار است ، می توان از یک ساختار داده به عنوان همپای ماندگار یاد کرد.

کاملاً مانا و تا اندازه‌ای مانا

Partial versus full persistence

ترجمه شده توسط مترجم گوگل

در مدل ماندگاری جزئی، یک برنامه نویس ممکن است هر نسخه قبلی از ساختار داده را پرس و جو کند، اما ممکن است فقط آخرین نسخه را به روز کند. این به معنی یک ترتیب خطی در بین هر نسخه از ساختار داده است. به عنوان مثال، اگر یک ساختار داده را به صورت زیر نمایش دهیم:

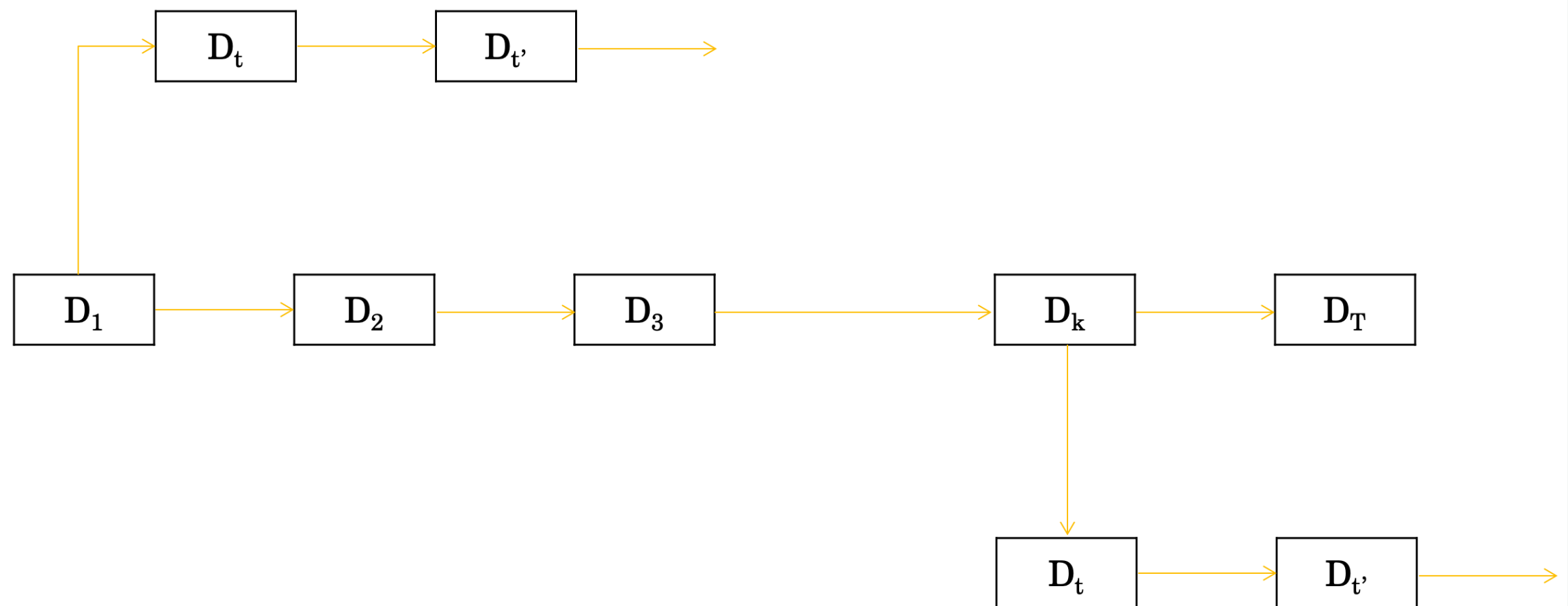


این، اگر علاوه بر پایدار بودن کامل، دو نسخه از ساختار داده یکسان بتوانند با هم ترکیب شوند و یک نسخه جدید را ایجاد کنند که همچنان کاملاً پایدار است، می‌توان از یک ساختار داده به عنوان همپای ماندگار یاد کرد.

کاملاً مانا و تا اندازه‌ای مانا

Partial versus full persistence

ترجمه شده توسط مترجم گوگل

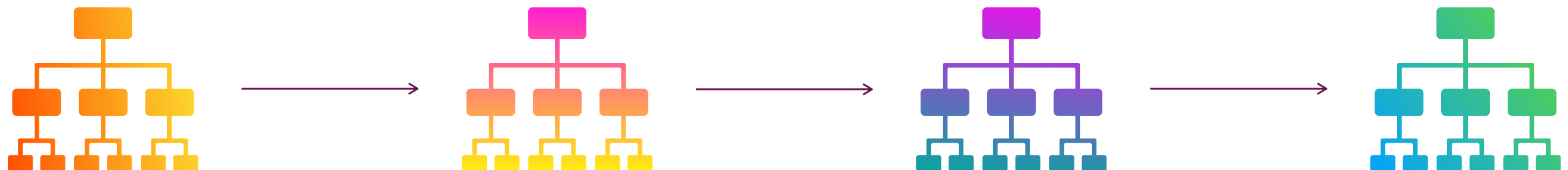


بازنویسی در حرکت

Copy On Write

ترجمه شده توسط مترجم گوگل

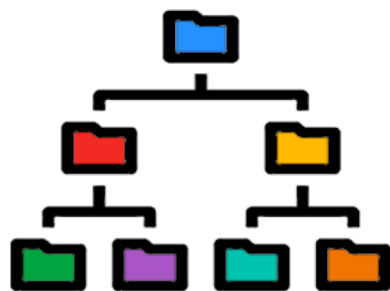
یک روش برای ایجاد یک ساختار داده پایدار ، استفاده از یک ساختار داده زودگذر مانند آرایه ای برای ذخیره داده ها در ساختار داده ها و کپی کردن کل ساختار داده ها با استفاده از معنای کپی بر روی نوشتن برای هرگونه به روزرسانی داده ها است. ساختار این یک روش ناکارآمد است زیرا کل ساختار داده پشتیبان باید برای هر نوشتن کپی شود ، که منجر به بدترین حالت $O(n \times m)$ ویژگی های عملکرد برای تغییرات m از یک آرایه از اندازه n می شود.



آیا روش بهتری نیست؟

اصلا قبله روش بود! که میگی بهتر؟

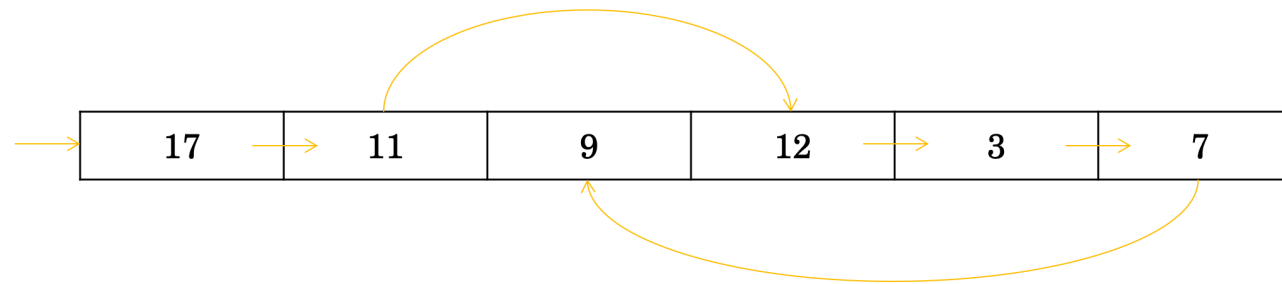
- همه داده‌های مرحله قبل را کپی می‌کند که لزوماً تغییری نکرده‌اند.
- بنابراین برای بهبود، می‌توانیم فقط عنصرهایی که تغییر کرده‌اند را ذخیره کنیم.
- اما چطوری؟
- گره چاق‌کنی
- بازنویسی مسیر



گره چاق کنی

Fat node

- برای هر گره، یک لیست از تاریخچه‌اش نگهداری می‌کنیم.
- برای دسترسی به نسخه زمانی، کفایت از جدیدترین نسخه‌ی قبل آن زمان استفاده کنیم.

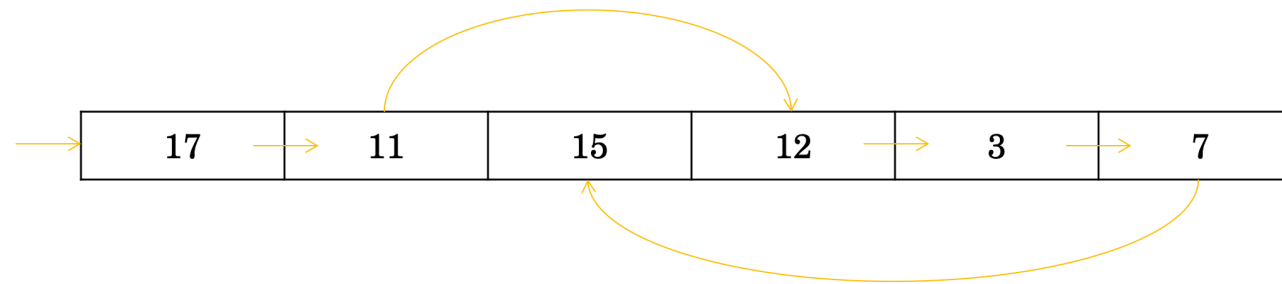


| Time Data Pointer | Time Data Pointer | Time Data Pointer | Time Data Pointer | Time Data Pointer | Time Data Pointer |
|-------------------------|-------------------------|-------------------------|-------------------------|-------------------------|-------------------------|
| 1 17 2 | 1 11 4 | 1 9 # | 1 12 5 | 1 3 6 | 1 7 3 |
| | | 2 15 # | | | |

گره چاق کنی

Fat node

- برای هر گره، یک لیست از تاریخچه‌اش نگهداری می‌کنیم.
- برای دسترسی به نسخه زمانی، کافیست از جدیدترین نسخه‌ی قبل آن زمان استفاده کنیم.



| Time Data Pointer | Time Data Pointer | Time Data Pointer | Time Data Pointer | Time Data Pointer | Time Data Pointer |
|-------------------------|-------------------------|-------------------------|-------------------------|-------------------------|-------------------------|
| 1 17 2 | 1 11 4 | 1 9 # | 1 12 5 | 1 3 6 | 1 7 3 |
| | | 2 15 # | | | |

تحلیل

گره چاق کنی

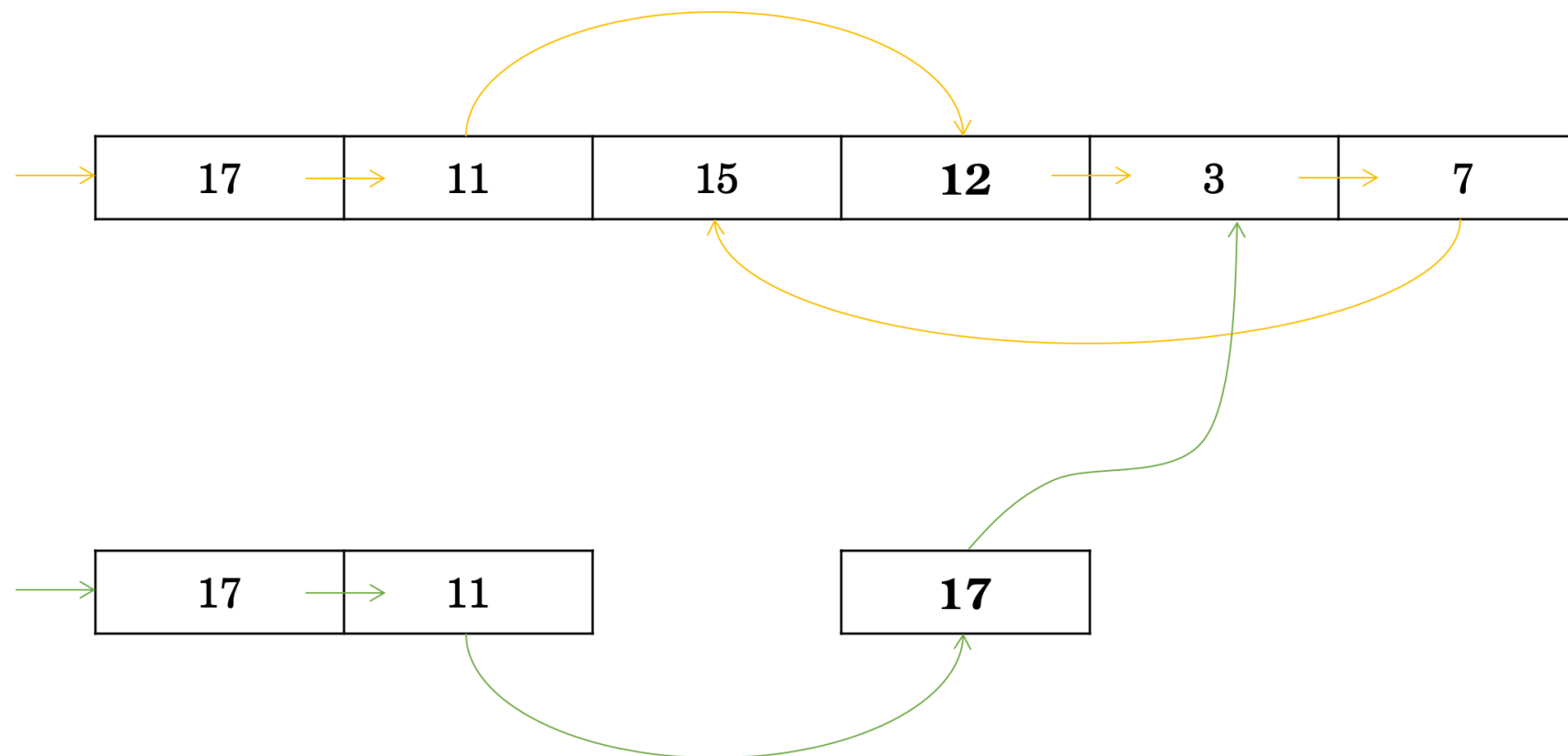
- در این روش برای هر به‌روزرسانی و ایجاد تغییرات از $O(1)$ هزینه می‌کنیم.
- حافظه نیز دقیقاً برابر با حجم تغییرات ایجاد شده است. (از این بهتر نمی‌شود!)
- برای استفاده از هر گره، باید بگردیم ببینیم جدیدترین نسخه که قبل از این نسخه داریم
چیه؟
- خبر خوب: از آنجا که زمان افزایشی است، با استفاده از یک جستجو دودویی می‌شه فهمید.

| پرسش | به‌روز رسانی |
|---------------------------------|-----------------------------|
| $O(T_{query}(n) \times \log m)$ | $O(1 \times T_{update}(n))$ |

بازنویسی مسیر

Path Copying

- گره های موجود در مسیر به روزرسانی را کپی کن، سپس به روزرسانی را انجام بده.
- برای پرسش از یک نسخه زمانی، کافیست تا از کپی مربوط به آن نسخه استفاده کنیم.



تحلیل

بازنویسی مسیر

- در این روش برای هر به‌روزرسانی و ایجاد تغییرات از $O(T_{copy} \times n_{copy})$ هزینه می‌کنیم.
- با تصادفی بودن فرآیندها، امید ریاضی این مقدار برای $T_{copy} = O(1)$ برابر با $O(\log m)$ است.
- حافظه نیز وابسته به طول مسیر تغییرات ایجاد شده است. (حالا واقعا هم حافظه مهمه؟)
- برای عملیات پرسش، تنها کافیست تا از اشاره‌گر شروع نسخه زمانی مورد نظر شروع کنیم که از $O(1)$ زمان خواهد برد.

| پرسش | به‌روز رسانی |
|----------------------------|----------------------------------|
| $O(1 \times T_{query}(n))$ | $O(T_{update}(n) \times \log m)$ |

آیا از این بهتر هم می شود؟

مگه میشه تو این دنیا، چیزی نتونه نشه؟

Making Data Structures Persistent*

JAMES R. DRISCOLL[†]

*Computer Science Department, Carnegie-Mellon University,
Pittsburgh, Pennsylvania 15218*

NEIL SARNAK

IBM T.J. Watson Research Center, Yorktown Heights, New York 10598

DANIEL D. SLEATOR

*Computer Science Department, Carnegie-Mellon University,
Pittsburgh, Pennsylvania 15218*

AND

ROBERT E. TARJAN[‡]

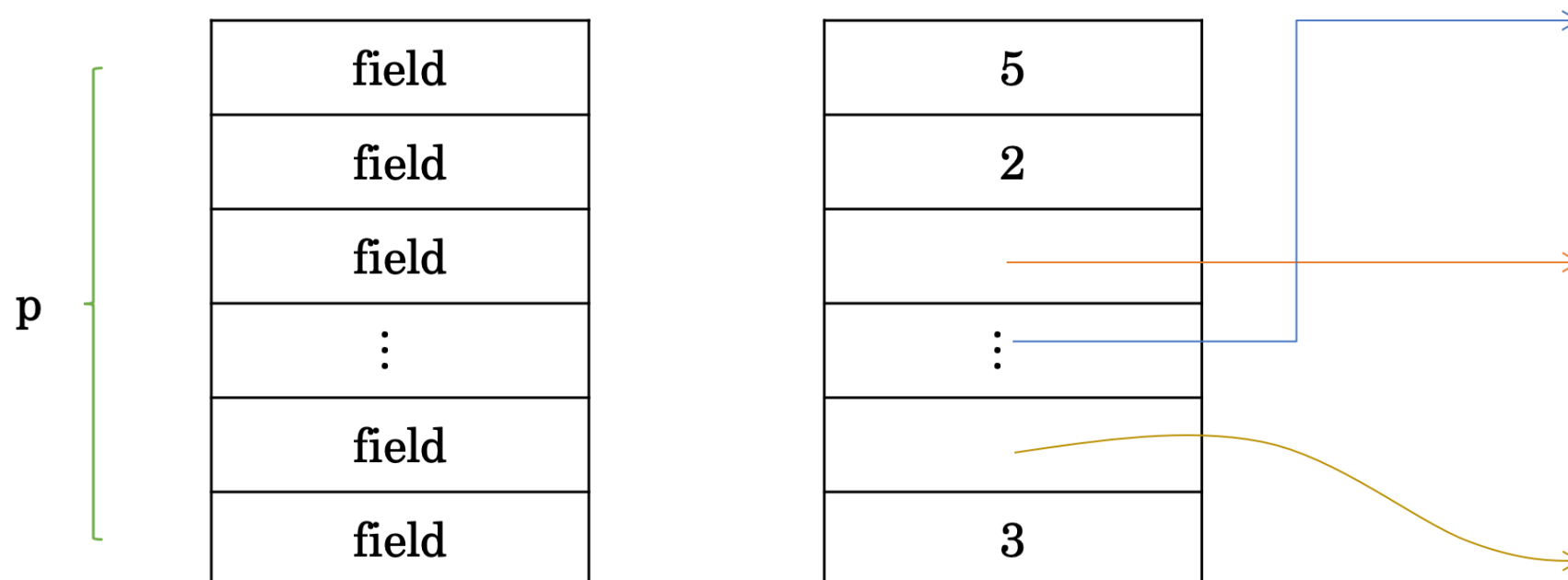
*Computer Science Department, Princeton University, Princeton, New Jersey 08544 and
AT&T Bell Laboratories, Murray Hill, New Jersey 07974*

Received August 5, 1986

ساختار گره‌ها

مار پوست خودشو ول میکنه اما خوی خودشو ول نمیکنه!

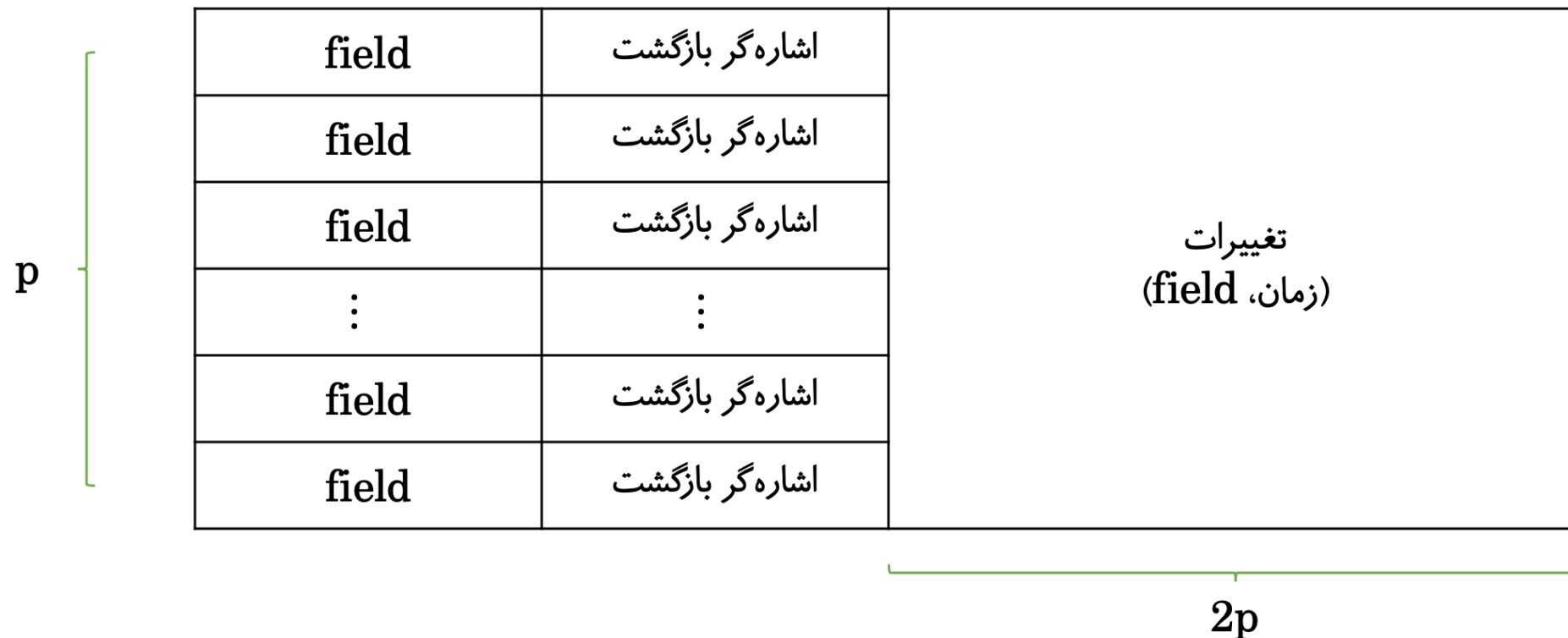
- فرض می‌کنیم در ساختمان داده عادی هر گره شامل p رکورد داده باشد، که این رکورد ها می‌توانند اطلاعات یا اشاره گر باشند.
- برای قابل اجرا بودن این روش، به هر گره نیز حداکثر p اشاره گر وارد می‌شود.



ساختار گره‌ها

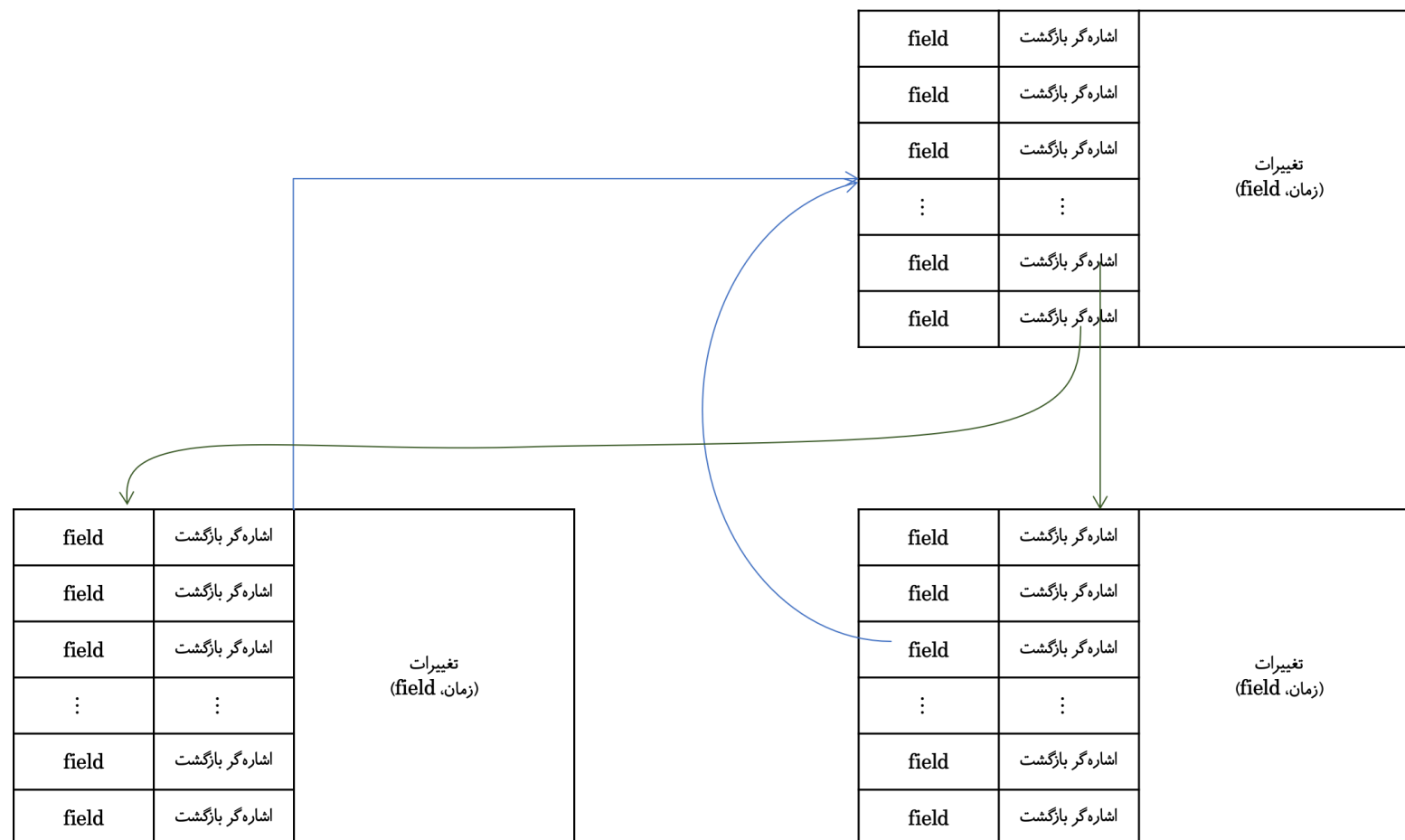
مار پوست خودشو ول میکنه اما خوی خودشو ول نمیکنه!

برای هر گره، دو قسمت دیگر نیز اضافه می‌کنیم، صندوق تغییرات (به گنجایش $2p$ رکورد) و همینطور p اشاره‌گر بازگشتی.



اشاره گر بازگشت

- برای هر گره، اشاره گری به گره‌هایی که به آن اشاره گر دارند، می‌دهیم.
- برای قابل اجرا بودن این روش، به هر گره نیز حداکثر p اشاره گر وارد می‌شود.

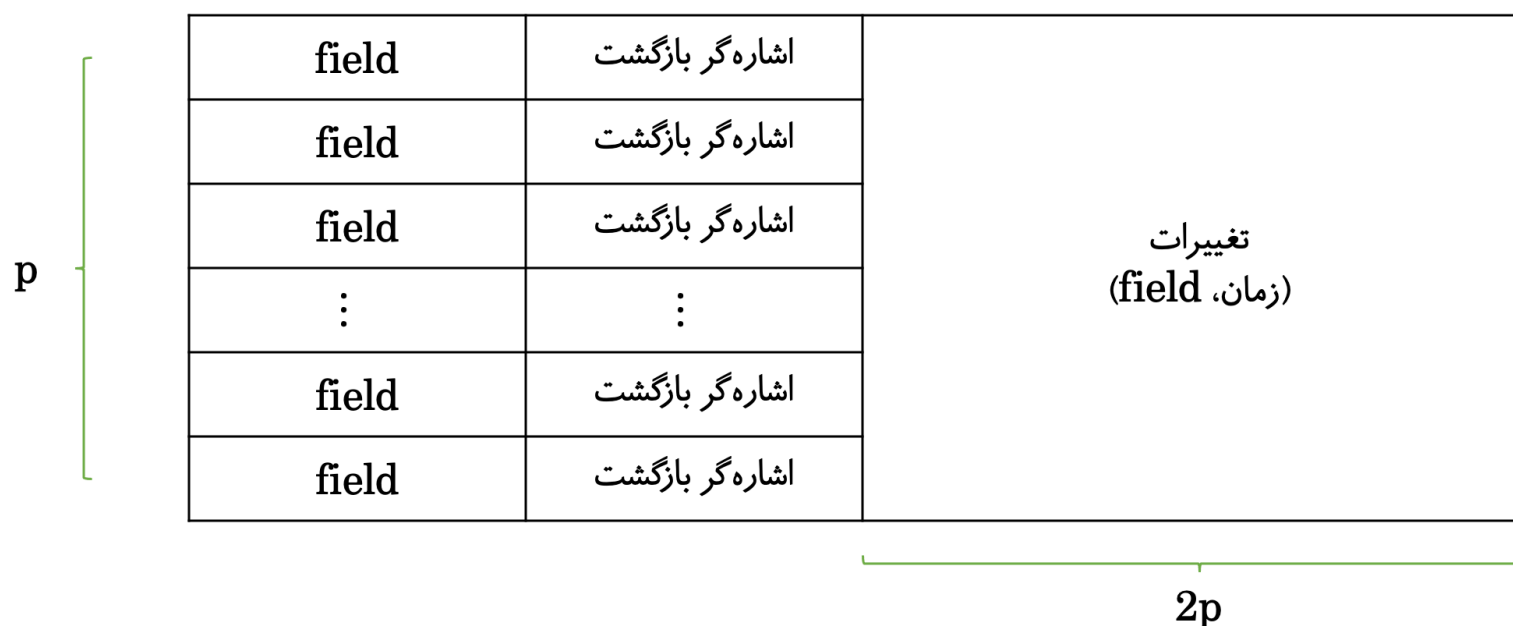


صندوق تغییرات

مقدار فعلی موجود در field ها را به هیچ عنوان تغییر نمی‌دهیم.

بلکه برای هر گره، بعد از به‌روز رسانی، تغییرات را در قالب یک چندتایی مرتب (زمان، تغییرات) به صندوق آن گره اضافه می‌کنیم.

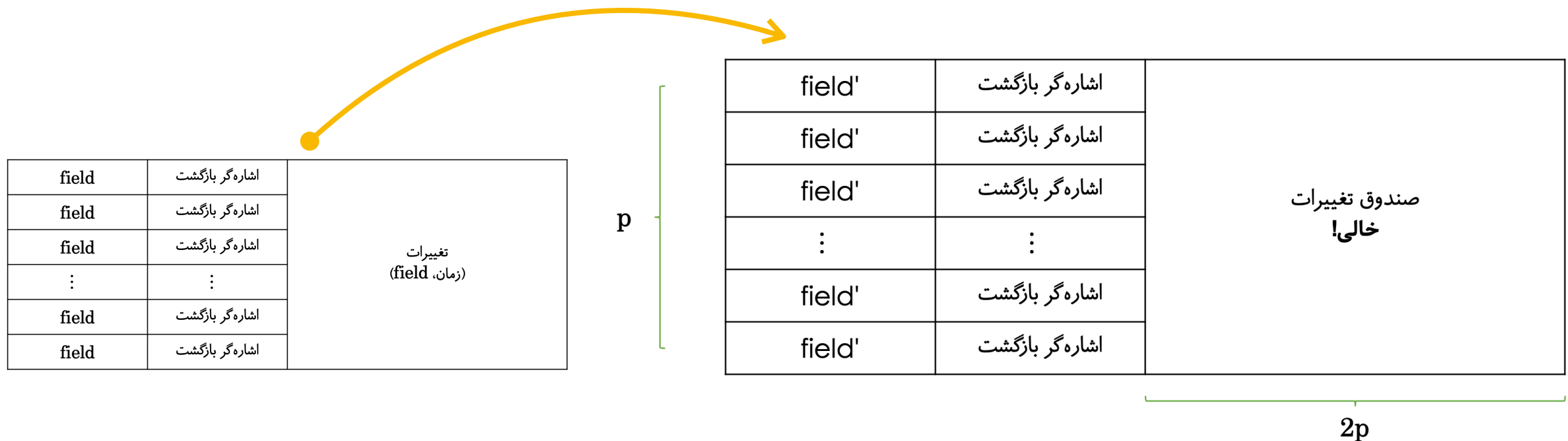
بنابراین به‌روز رسانی هر گره $O(p)$ زمان خواهد برد.



صندوق تغییرات

اگر صندوق پر شده بود، چه کنیم؟

در این صورت (همانند Path Copying) یک گره جدید، متناظر با گره قبلی در ساختمان داده (نسخه جدید) اضافه می کنیم. اما اینبار، تمام تغییرات موجود در صندوق را روی گره اعمال می کنیم



استفاده از اشاره گرهای بازگشت

بالاخره دارن به کار میان!

- حال باید در نسخه جدید ساختمان داده، تمام گره‌هایی که به گره قدیمی اشار میکردند را تغییر دهیم، آنها حالا باید به گره جدید ما اشاره کنند.
- یک لحظه، چگونه این کار را انجام دهیم؟
- با استفاده از اشاره گرهای بازگشت به آنها دسترسی پیدا می‌کنیم.
- با همین الگوریتم به روزرسانی گفته شده، با فراخوانی بازگشتی، آنها را نیز به روز رسانی می‌کنیم.

استفاده از اشاره گرهای بازگشت

آیا این هزینه زیادی در بر نخواهد داشت؟

- حال باید در نسخه جدید ساختمان داده، تمام گره‌هایی که به گره قدیمی اشاره میکردند را تغییر دهیم، آنها حالا باید به گره جدید ما اشاره کنند.
- یک لحظه، چگونه این کار را انجام دهیم؟
- با استفاده از اشاره گرهای بازگشت به آنها دسترسی پیدا می‌کنیم.
- با همین الگوریتم به روزرسانی گفته شده، با فراخوانی بازگشتی، آنها را نیز به روزرسانی می‌کنیم.

یه مرور بکنیم.

لطفا نیچید.

- برای هر گره، p خانه داده و اشاره گر داریم.
- برای هر گره، اشاره گری به تمام گره‌هایی که به این گره اشاره می‌کنند، نگه داشتیم.
- برای استفاده از گره و زمان T ، ابتدا تمام تغییرات موجود در صندوق که مربوط به قبل زمان T هستند را به طور لحظه‌ای اعمال کن و نتیجه را استفاده کن. ($O(2p)$)
- برای بروزرسانی هر گره:
- اگر صندوق تغییرات گره، هنوز پر نشده بود، تغییرات جدید را در آن اضافه کن.
- در غیر اینصورت، گره جدید بساز، همه تغییرات ($2p+1$) را روی آن اعمال کن و به طور بازگشتی و به کمک اشاره گرهای بازگشتی، تمام گره‌های اشاره کننده به گره قدیمی را به گره جدید هدایت کن.

تحلیل پیچیدگی

غول چراغ جادو، یا که خان هفتم؟

$$\delta(D_t) = C \times \sum_{v \in V(D)} n_{mod}(v, t) \quad \text{تغییرات مربوط در صندوق } v$$

$$\Rightarrow T(t) = \delta(D_t)$$

$$\Rightarrow T(t) = \bar{C} \times \left(C + C + \left[-2Cp + R(u) \right] \right)$$

که u گره به روز رسانی شده است؛
 R نیز تعداد فراخوانی تابع بازگشتی است.

$$\Rightarrow T(t) = \bar{C} \times \left(C + C + \left[-2Cp + \left(C + C + [-2Cp_u + R_u(u')] \right) \right] \right)$$

$$\Rightarrow T(t) = \bar{C} \times \left(C + C + \left[-2Cp + R(u') \right] \right)$$

$$\Rightarrow T(t) = \bar{C} \times \left(2C \right) \in O(1)$$

منابع

مایعی غلیظ و افروختنی به رنگ قهوه‌ای سوخته! [نیازمند منبع]

1. David Karger, 6.854 Advanced Algorithms , MIT ocw
2. "Lecture 7: Amortized Analysis". Carnegie Mellon University. Retrieved 14 March 2015.
3. Driscoll, James R., et al. "Making data structures persistent." Journal of computer and system sciences 38.1 (1989): 86-124.
4. Demaine, Erik D., John Iacono, and Stefan Langerman. "Retroactive data structures." ACM Transactions on Algorithms (TALG) 3.2 (2007): 13-es.

سوال؟

