



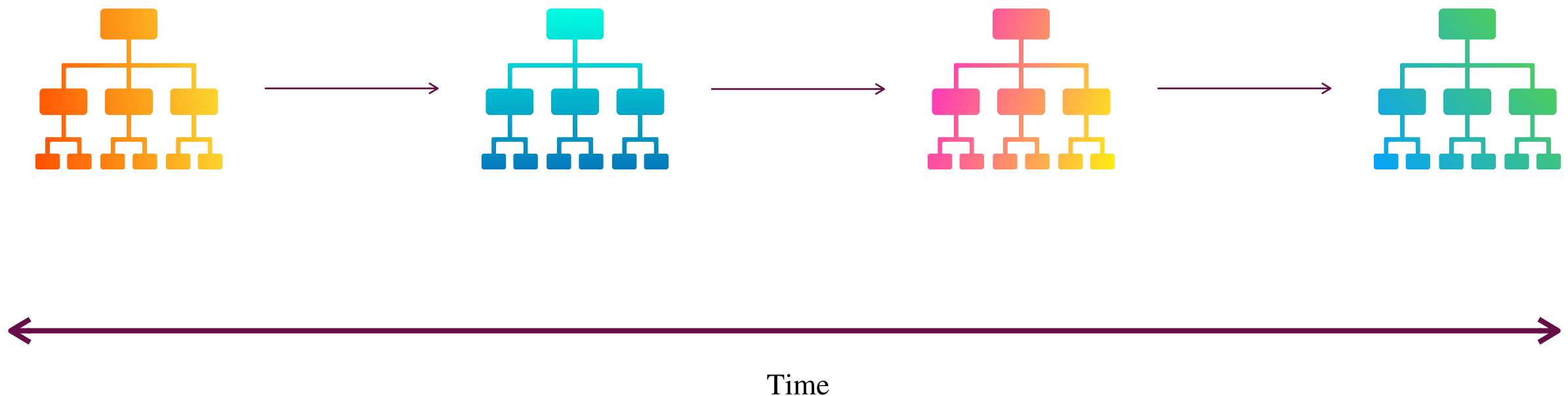
ساختمان‌های داده مانا

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

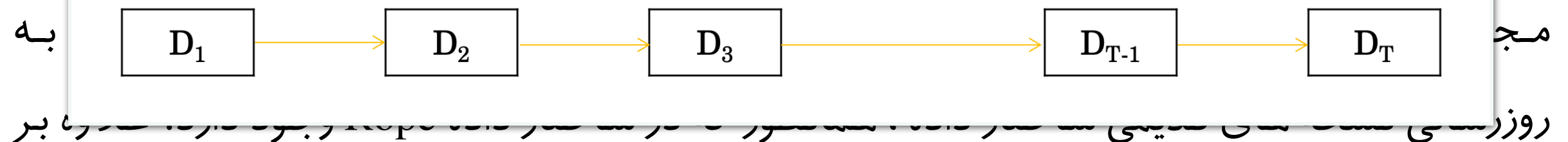
در مدل ماندگاری جزئی ، یک برنامه نویس ممکن است هر نسخه قبلی از ساختار داده را پرس و جو کند ، اما ممکن است فقط آخرین نسخه را به روز کند. این به معنی یک ترتیب خطی در بین هر نسخه از ساختار داده است. در مدل کاملاً پایدار ، به روزرسانی ها و پرس و جوها در هر نسخه از ساختار داده مجاز هستند. در برخی موارد ، 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. کاملاً مانا به معنی این است که هر نسخه از ساختار داده را می‌توان به یک نسخه دیگر پیوند داد.

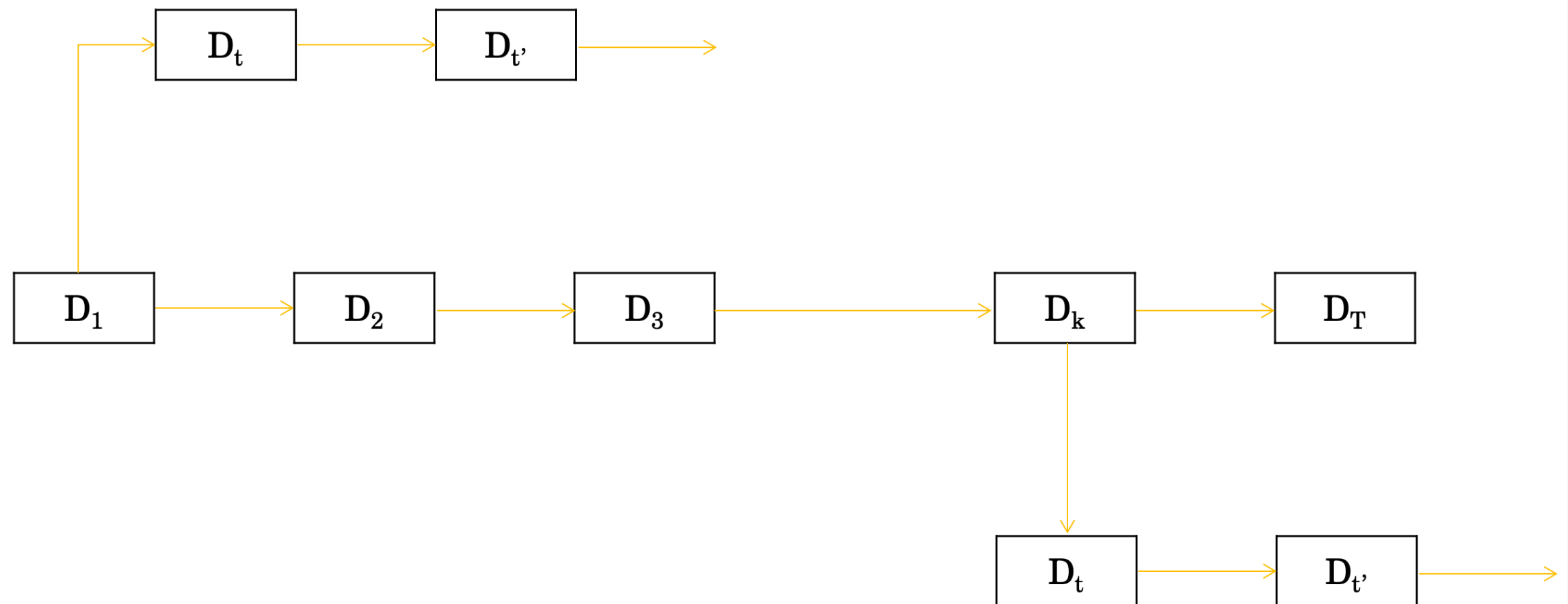


روزانه می‌توان نسخه‌های جدیدی از ساختار داده را به روز کرد. اگر علاوه بر پایدار بودن کامل، دو نسخه از ساختار داده یکسان بتوانند با هم ترکیب شوند و یک نسخه جدید را ایجاد کنند که همچنان کاملاً پایدار است، می‌توان از یک ساختار داده به عنوان همپای ماندگار یاد کرد.

کاملاً مانا و تا اندازه‌ای مانا

Partial versus full persistence

ترجمه شده توسط مترجم گوگل

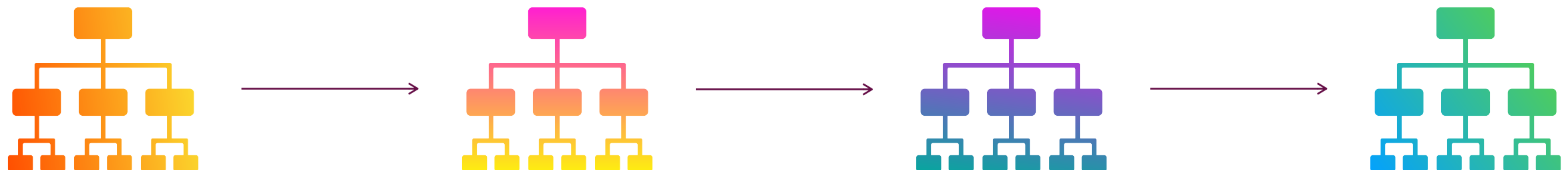


بازنویسی در حرکت

Copy On Write

ترجمه شده توسط مترجم گوگل

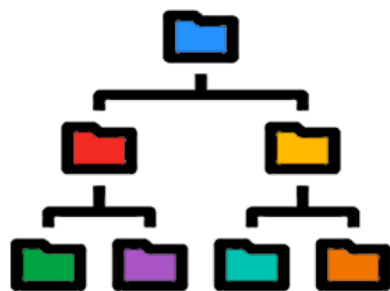
یک روش برای ایجاد یک ساختار داده پایدار ، استفاده از یک ساختار داده زودگذر مانند آرایه ای برای ذخیره داده ها در ساختار داده ها و کپی کردن کل ساختار داده ها با استفاده از معنای کپی بر روی نوشتن برای هرگونه به روزرسانی داده ها است. ساختار این یک روش ناکارآمد است زیرا کل ساختار داده پشتیبان باید برای هر 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؟

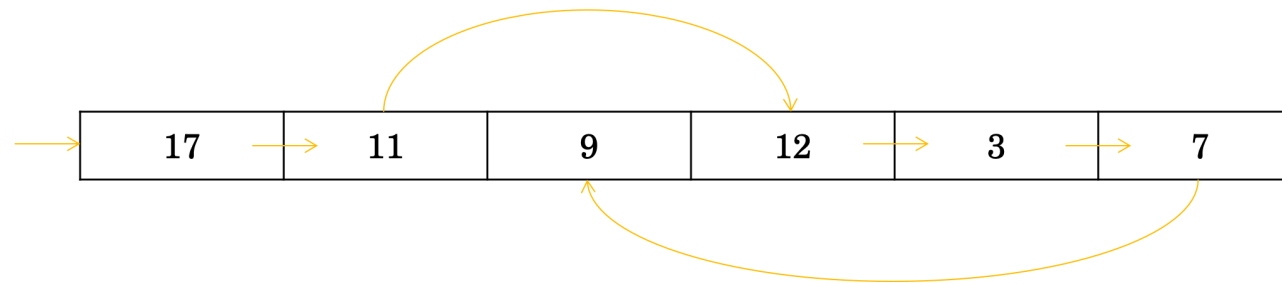
- همه داده‌های مرحله قبل را کپی می‌کند که لزوماً تغییری نکرده اند.
- بنابراین برای بهبود، می‌توانیم فقط عنصرهایی که تغییر کرده‌اند را ذخیره کنیم.
- اما چطوری؟
- گره چاق کنی
- بازنویسی مسیر



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.

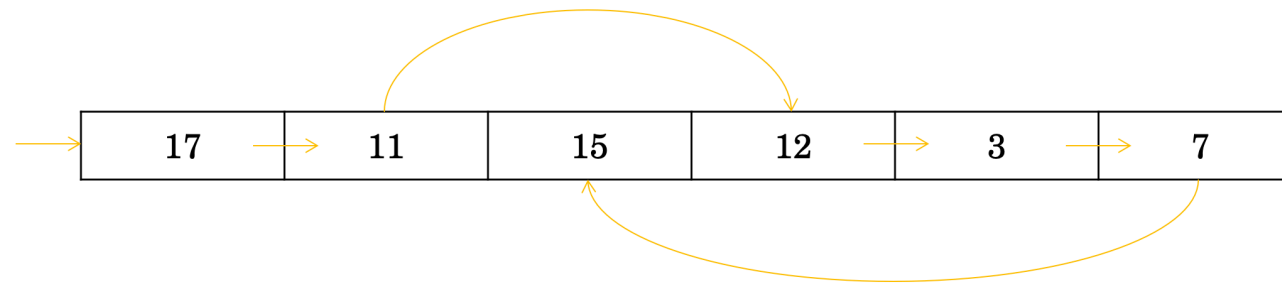


Time Data Pointer	Time Data Pointer	Time Data Pointer	Time Data Pointer	Time Data Pointer	Time Data Pointer
1 17 2	1 11 4	1 9 #	1 12 5	1 3 6	1 7 3
		2 15 #			

گره چاق کنی

Fat node

- برای هر گره، یک لیست از تاریخچه‌اش نگهداری می‌کنیم.
- برای دسترسی به نسخه زمانی، 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 #			

تحلیل

گره چاق کنی

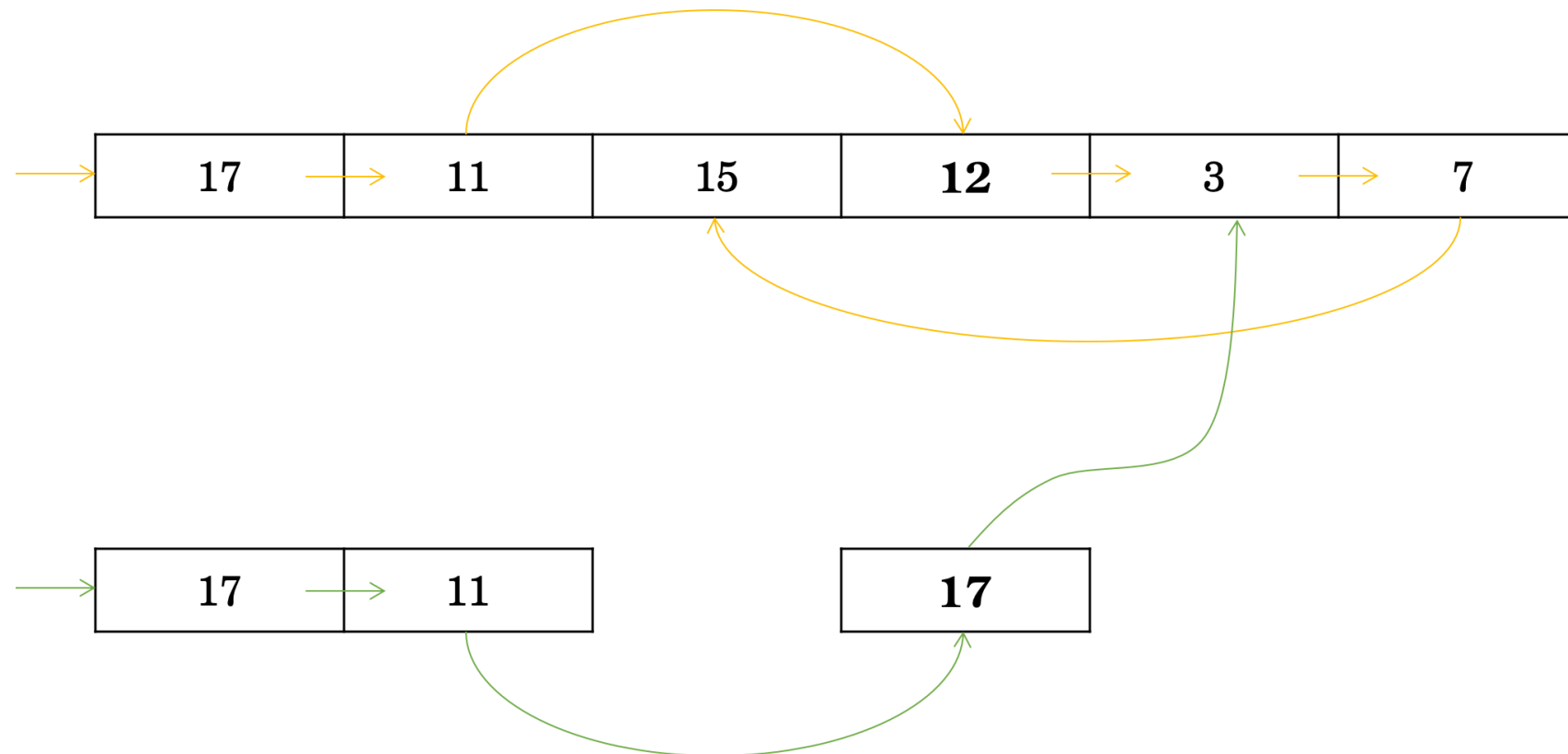
- در این روش برای هر به‌روزرسانی و ایجاد تغییرات از $O(1)$ هزینه می‌کنیم.
- حافظه نیز دقیقاً برابر با حجم تغییرات ایجاد شده است. (از این بهتر نمی‌شود!)
- برای استفاده از هر گره، باید بگردیم ببینیم جدیدترین نسخه که قبل از این نسخه داریم
چیه؟
- خبر خوب: از آنجا که زمان افزایشی است، با استفاده از یک جستجو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

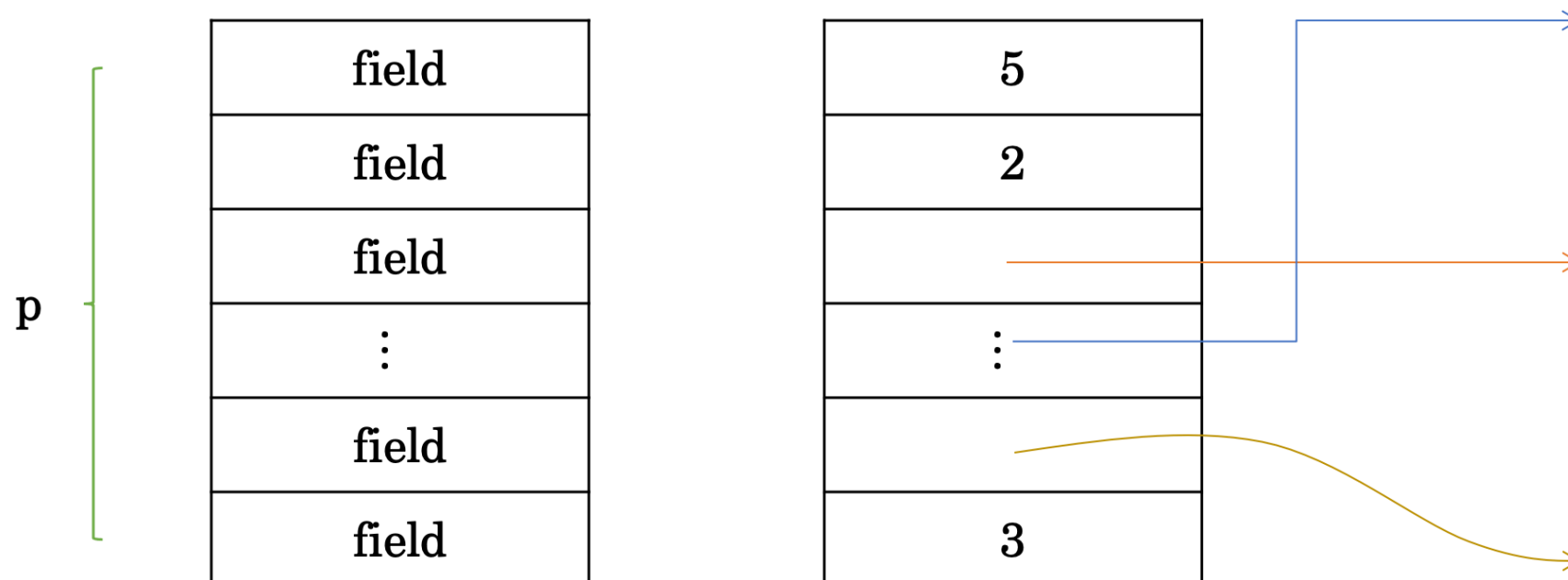
*Computer Science Department, Princeton University, Princeton, New Jersey 08544 and
AT&T Bell Laboratories, Murray Hill, New Jersey 07974*

Received August 5, 1986

ساختار گره‌ها

مار پوست خودشو ول میکنه اما خوی خودشو ول نمیکنه!

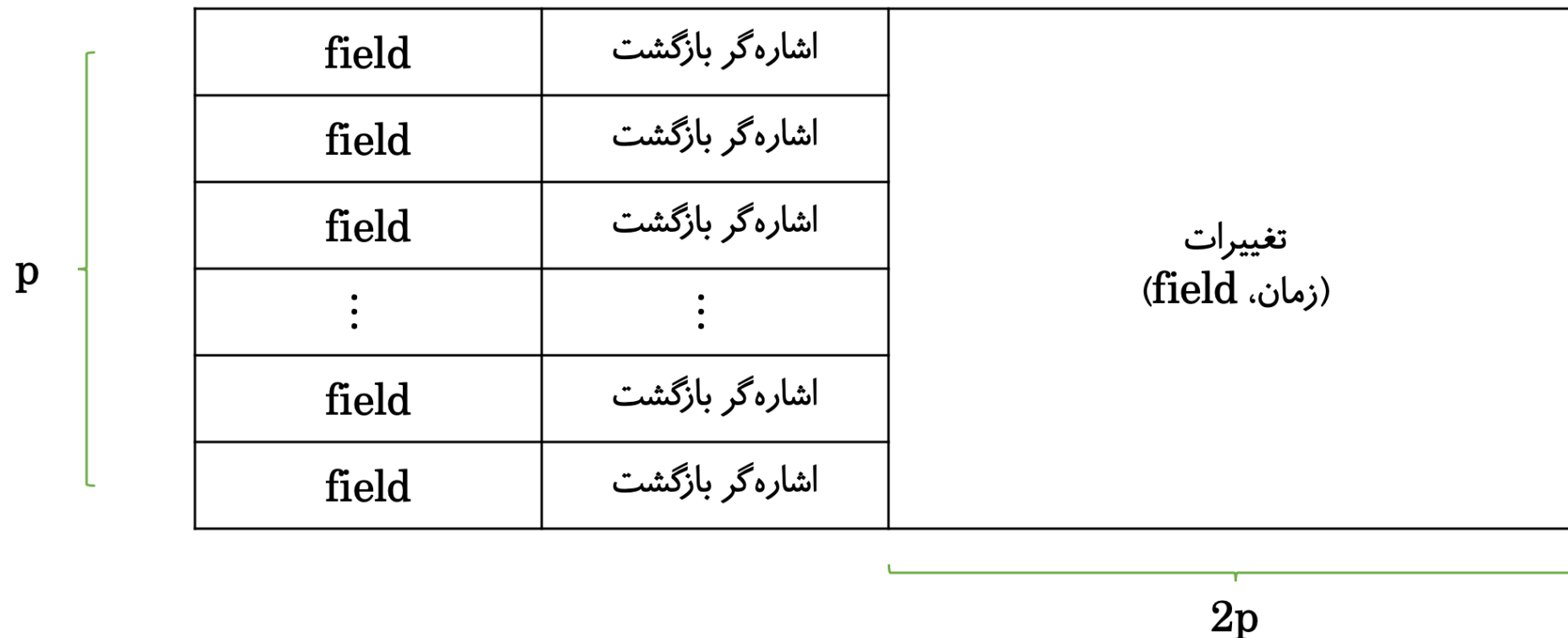
- فرض می‌کنیم در ساختمان داده عادی هر گره شامل p رکورد داده باشد، که این رکورد ها می‌توانند اطلاعات یا اشاره گر باشند.
- برای قابل اجرا بودن این روش، به هر گره نیز حداکثر p اشاره گر وارد می‌شود.



ساختار گره‌ها

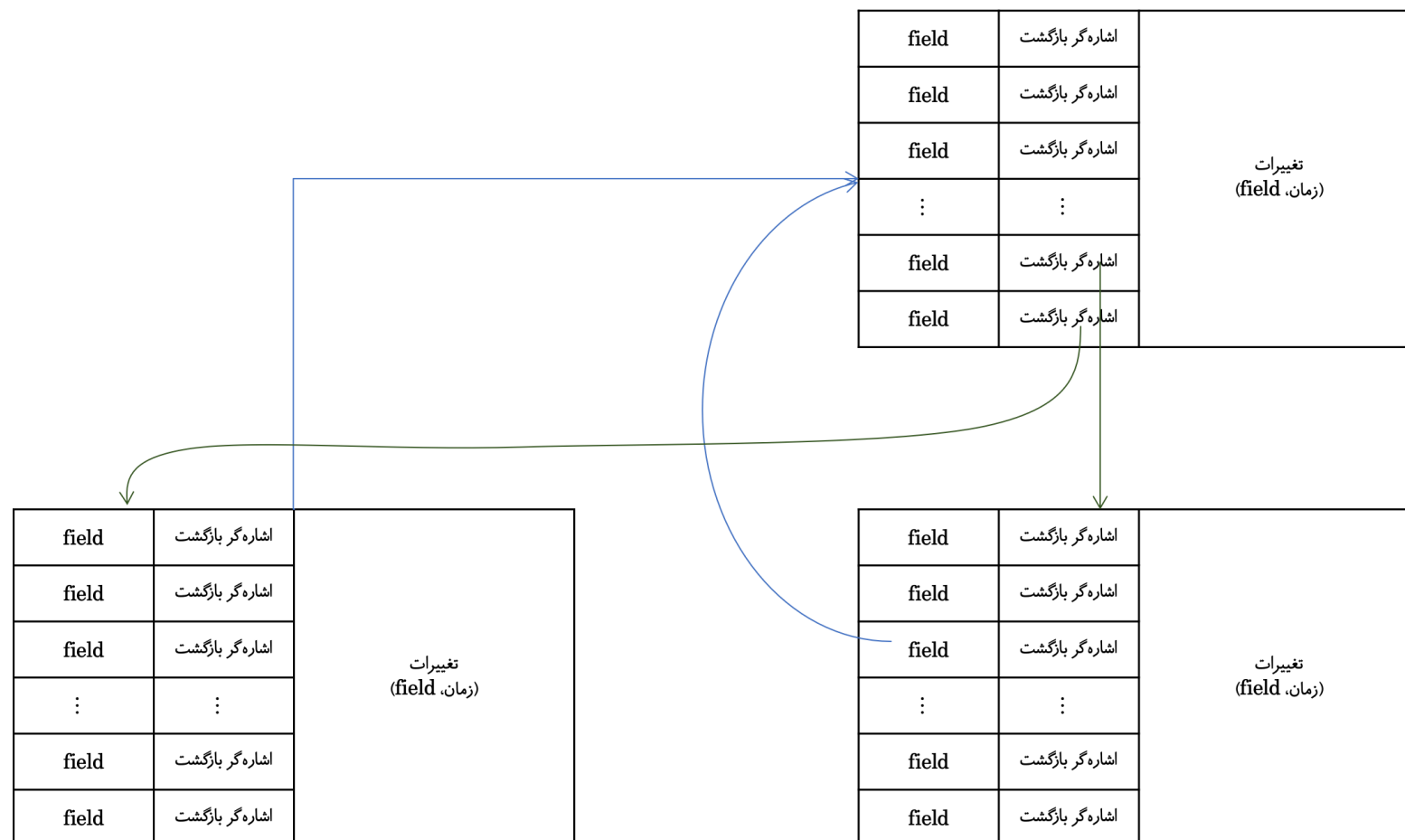
مار پوست خودشو ول میکنه اما خوی خودشو ول نمیکنه!

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.

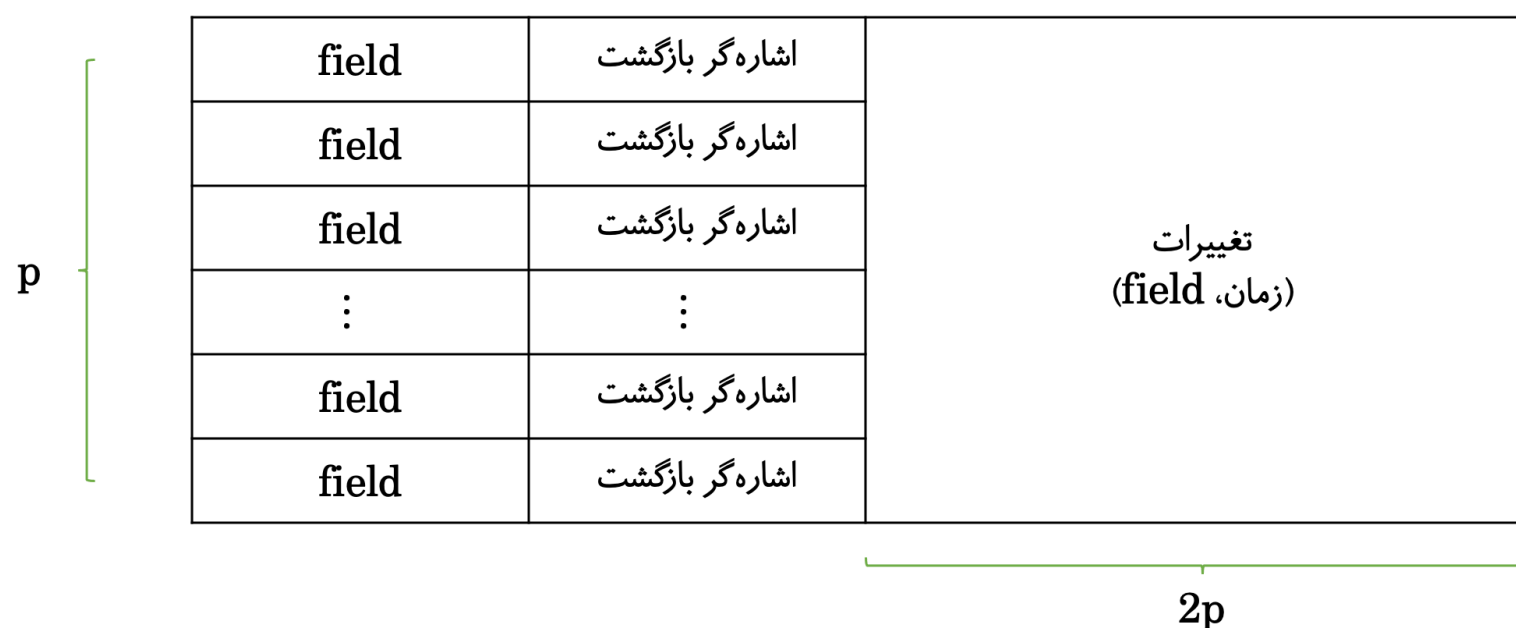


صندوق تغییرات

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.

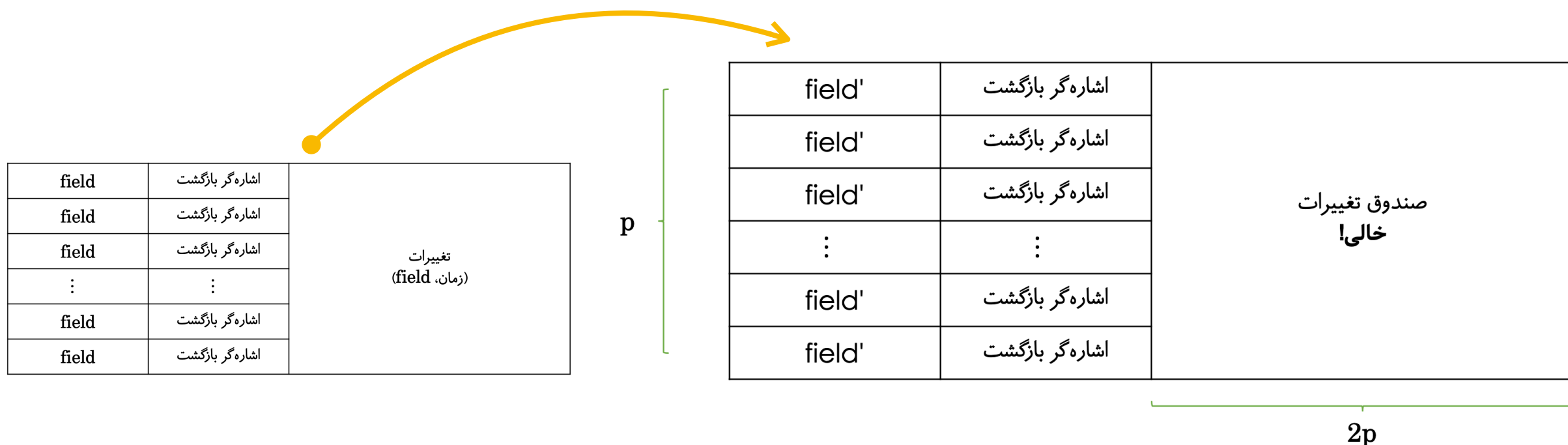
بنابراین به‌روز رسانی هر گره $O(p)$ زمان خواهد برد.



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 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.

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؟

