Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 004.9
DOI: https://doi.org/10.24867/25BE21Kovacevic

PREGLED I IMPLEMENTACIJA ODABRANIH RETROAKTIVNIH STRUKTURA PODATAKA

OVERVIEW AND IMPLEMENTATION OF SELECTED RETROACTIVE DATA STRUCTURES

Vera Kovačević, *Fakultet tehničkih nauka, Novi Sad*

Oblast – ELEKTROTEHNIKA I RAČUNARSTVO
Kratak sadržaj – *Retroaktivne strukture podataka podržavaju modifikaciju niza operacija izvršenih nad datom strukturom. U ovom radu opisani su koncepti parcijalne i potpune retroaktivnosti, te je ilustrovana parcijalna retroaktivnost pomoću odabranih struktura: red, stek i red sa priotitetom. Glavni fokus rada jeste opis metoda implementacije odabranih struktura pomoću dvostruko povezane liste i strukture treap. Zaključeno je da se može postići vrijeme izvršavanja slično odgovarajućim standardnim strukturama.*
Ključne reči: *napredne strukture podataka, retroaktivne strukture podataka, red, stek, red sa prioritetom*
Abstract – *Retroactive data structures support modification of a series of operations performed on a given structure. In this paper, the concepts of partial and complete retroactivity are introduced, and partial retroactivity is illustrated using following structures: queue, stack and priority queue. The main focus of the paper is the description of methods of implementation of selected structures using doubly linked list and treap structure. It is concluded that execution time similar to the corresponding standard structures can be achieved.*
Keywords: *advanced data structures, retroactive data structures, queue, stack, priority queue*

1. UVOD

Jedna od vrsta naprednih struktura podataka su vremenske strukture (eng. temporal data structures), koje uvode vrijeme kao novu dimenziju. Na osnovu toga na koji način koriste vrijeme kao novu dimenziju, one mogu biti perzistentne (eng. persistent data structures) ili retroaktivne (eng. retroactive data structures) [1].

Retroaktivne strukture podataka podržavaju modifikaciju niza operacija izvršenih nad datom strukturom. To podrazumijeva naknadno dodavanje, brisanje ili izmjenu operacija koje su izvršene u nekom trenutku u prošlosti. Na taj način se postiže da promjena koja je naknadno izvršena u prošlosti utiče na trenutno stanje, kao i na sva stanja između trenutka te promjene i sadašnjeg trenutka. Razlikuju se parcijalno retroaktivne strukture, koje dozvoljavaju modifikaciju podataka u prošlosti, ali se upiti mogu izvršavati isključivo u sadašnjem trenutku, i potpuno retroaktive strukture, koje dozvoljavaju izvršavanje upita i u prošlosti [1].

NAPOMENA:
Ovaj rad proistekao je iz master rada čiji mentor je bila dr Dunja Vrbaški, docent.

Naredna poglavlja ovog rada baviće se upravo istraživanjem različitih metoda implementacije odabranih retroaktivnih struktura. U sljedećem poglavlju biće dat pregled naučnih radova od čijih autora potiče sama ideja, kao i nekih radova koji se nadovezuju na osnovnu ideju. Nakon toga će biti objašnjena opšta teorija o retroaktivnosti i opisani predloženi načini implementacije. Zatim, biće prikazana implementacija tri odabrane retroaktivne strukture, a nakon toga slijedi diskusija rezultata, primjena i prijedlozi za dalja unapređenja implementacije.

2. STANJE U OBLASTI

Ideja za izradu ovog rada potekla je od kursa „Napredne strukture podataka“ profesora Erika Demejna sa univerziteta MIT [1]. Profesor je osnovnu ideju o retroaktivnim strukturama podataka predstavio je u radu [2], gdje se uvodi koncept retroaktivnih struktura i opšta teorija o retroaktivnosti, a zatim se istražuju različite varijante retroaktivnosti, parcijalna i potpuna. Demonstrirano je da je moguće implementirati efikasne retroaktivne verzije reda, reda sa dva kraja, reda sa prioritetom i disjunktnog seta.

Nakon što su sredinom dvijehiljaditih godina objavljeni prethodno navedeni radovi, ova oblast se nastavila razvijati u pravcu unapređenja metoda implementacije i vremena izvršavanja, kao i primjene u rješavanju praktičnih problema.

Autori rada [2] opisali su opšti način transformacije parcijalno retroaktivne u potpuno retroaktivnu strukturu pod uslovom da postoji efikasna perzistentna verzija date strukture. Nekoliko godina kasnije, u radu [3], originalni autor osmislio je novi pristup baziran na hijerarhijskim kontrolnim tačkama (eng. *hierarchical checkpointing*), koji predstavlja opštu logaritamsku transformaciju parcijalno retroaktivne u potpuno retroaktivnu strukturu. Na osnovu teorema iznijetih u radu [3], konkretna implementacija navedene strukture opisana je u radu [4]. Korišćeno je samobalansirajuće binarno stablo pretrage (eng. *self-balanced binary search tree*).

Drugi pravac razvoja ove oblasti vezan je za praktičnu primjenu retroaktivnih struktura. U radu [7] ilustrovana je primjena retroaktivnog reda sa prioritetom u rješavanju problema pronalaženja najkraćeg puta u dinamičkim grafovima.

3. METODOLOGIJA

Uopšteno, može se reći da svaka struktura podataka sadrži niz modifikacija podataka i upita u toku vremena. Prema opštoj teoriji o retroaktivnisti, listu operacija izvršenih

nad strukturom definišemo kao U = $[u_{t_1}, \dots, u_{t_m}]$, gdje u_{t_i} predstavlja operaciju izvršenu u trenutku t_i i $t_1 < \cdots < t_m$. Pretpostavka je da se u jednom trenutku izvršava samo jedna operacija.

Kao što je već rečeno, parcijalno retroaktivne strukture pored izvršavanja upita nad podacima u sadašnjem trenutku podržavaju i sljedeće dvije operacije:

- *Insert*(*t, insert*(*x*)) - dodavanje operacije dodavanja elementa *x* u niz operacija u trenutku *t,*
- *Insert*(*t, delete*()) - dodavanje operacije brisanja operacije koja je izvršena u trenutku *t.*

Dakle, omogućeno je da se „putuje kroz vrijeme“ do nekog prethodnog stanja strukture i da se nad tim stanjem izvrši operacija koja će potencijalno promijeniti bilo koju operaciju između tog trenutka i sadašnjeg trenutka. Pri definisanju ovih operacija podrazumijeva se da se izvršavaju samo validne operacije, odnosno da se ne može izvršiti brisanje u trenutku *t* ako u tom trenutku nije izvršena nijedna operacija. Za razliku od parcijalne, potpuna retroaktivnost podržava i direktno posmatranje prošlosti, a operacija koja se dodaje može se definisati kao:

- *Query*(*t, search*(*x*)) - pretraživanje elementa *x* u bilo kom trenutku *t* u prošlosti.

3.1. Retroaktivni red

Red (eng. *queue*) je linearna struktura koja prati princip FIFO (eng. *First In, First Out*), što znači da će prvi dodati element biti prvi uklonjen. Podržava dvije primarne operacije: *enqueue*, koja dodaje element na kraj reda, i *dequeue*, koja uklanja element sa početka reda.

Red podržava operacije *enqueue(x), dequeue(), front(),* koja vraća sljedeći element na redu za obradu i *back(),* koja vraća posljednji dodat element. Dakle, da bi se dobio retroaktivni red potrebno je podržati sljedeće operacije: *Insert(t, enqueue(x)), Insert(t, dequeue()), Query(t, front()) i Query(t, back()).*

Kada je u pitanju parcijalno retroaktivni red, vrijednost *t* kod upita uvijek će biti ∞, odnosno sadašnji trenutak. Metoda za postizanje retroaktivnosti kod ove strukture data je u radu [2] i zasniva se na korišćenju dvostruko povezane pomoćne liste. U pomoćnoj listi se čuvaju sve operacije sortrane po trenutku izvršavanja. Struktura takođe sadrži i dva pokazivača: *F*, koji pokazuje na prvi element u redu i *B*, koji pokazuje na posljednji element. Kada se retroaktivno doda operacija *enqueue*, novi element se dodaje u listu na odgovarajuće mjesto u zavisnosti od trenutka izvršavanja, i po potrebi se modifikuju pokazivači. Analogno tome se izvršava i uklanjanje operacije *enqueue* i dodavanje i uklanjanje operacije *dequeue*.

3.2. Retroaktivni red sa dva kraja

Kombinovanjem functionalnosti steka i reda nastaje red sa dva kraja (eng. *deque, double-ended queue*), poseban tip reda koji podržava dodavanje i uklanjanje elemenata i na početku i na kraju reda.

U retroaktivnoj verziji koriste se dvije pomoćne liste, U_L i U_R, koje čuvaju sve operacije na lijevom, odnosno desnom kraju sortirane po trenutku izvršavanja. Pored njih, koristi se i modifikacija binarnog stabla koja čuva prefiksne sume, kao i promjenljive L i R. Prefiksna suma je kumulativna suma elemenata niza, gdje svaki element rezultantnog niza predstavlja sumu svih elemenata originalnog niza do posmatranog elementa. Promjenljive L i R predstavljaju pokazivače na trenutni krajnji lijevi i krajnji desni element. Za listu U_R operacijama dodavanja pridružuje se težina +1, dok se operacijama brisanja pridružuje težina -1 i te težine se čuvaju u binarnom stablu. Prema tome, vrijednost pokazivača R u trenutku t može se izračunati kao prefiksna suma elemenata sa vremenom izvršavanja do tog trenutka. Analogno tome koristi se i binarno stablo za prefiksne sume liste U_L.

3.3. Retroaktivni red sa prioritetom

Ova vrsta reda pored elemenata čuva i njihove prioritete, što omogućava da se prvo pristupi elementima sa višim prioritetom. Da bi se postigla parcijalna retroaktivnost potrebno je podržati sljedeće operacije: Insert(t, insert(k)), Insert(t, delete-min()), Delete(t) i Query(find-min()). Kao pomoćna struktura koristi se skup elemenata koji se trenutno nalaze u redu sa prioritetom označen sa Q_now.

Ideja na kojoj se zasniva metoda implementacije parcijalno retroaktivnog reda sa prioritetom je da je kardinalitet skupa Q_now uvijek jednak razlici broja izvršenih operacija dodavanja i operacija brisanja. Kada se dodaje operacija insert(k) u bilo kom trenutku, u skup Q_now se dodaje jedan element, a isto se dešava i kada se briše operacija delete-min(). U posebnom slučaju, kada se ubacuje element k sa najvišim prioritetom u datom trenutku i on bi trebalo da bude uklonjen nekom ranijom delete-min() operacijom, umjesto njega se vraća element koji je bio zahvaćen operacijom brisanja prije dodavanja elementa k.

Definisane su formule za određivanje koji element treba biti ubačen u skup Q_now u odgovarajućim slučajevim. Da bi se lakše odredio taj element uvodi se koncept bridge. Ako je Q_t skup elemenata koji su sadržani u redu sa prioritetom u trenutku t i ako je on podskup skupa Q_now, onda postoji bridge u trenutku t.

Skup Q_now čuva se u binarnom stablu čiji čvorovi sadrže ključeve i pokazivače na odgovarajuće operacije. Slično kao kod retroaktivnog reda sa dva kraja, za određivanje bridge trenutaka koristi se modifikacija binarnog stabla koja čuva prefiksne sume.

4. IMPLEMENTACIJA I DISKUSIJA REZULTATA

Nakon analize različitih metoda za implementaciju retroaktivnosti kod linearnih struktura, odabrano je nekoliko njih za ilustraciju ovog koncepta, a to su dvije jednostavnije implementacije pomoću dvostruko povezane liste (parcijalno retroaktivni red i parcijalno retroaktivni stek) i jedna kompleksnija implementacija koja koristi i pomoćne strukture podataka (parcijalno retroaktivni red sa prioritetom).

4.1. Implementacija parcijalno retroaktivnog reda

Osnova implementacije parcijalno retroaktivnog reda je dvostruko povezana lista koja čuva elemente reda i pomoćna lista koja čuva sve enqueue i dequeue operacije. Klase koje se koriste su sljedeće:

Node – predstavlja jedan element liste,

Operation – predstavlja jednu operaciju,

enqueue ili dequeue i PartiallyRetroactiveQueue – predstavlja parcijalno retroaktivni red.

Klasu PartiallyRetroactiveQueue čine dvostruko povezana lista elemenata koji se trenutno nalaze u redu i lista svih operacija sortirana prema rastućem redsolijedu trenutaka izvršavanja. Dakle, atributi klase su referenca na prvi element reda (eng. first), referenca na posljednji element reda (eng. last) i lista operacija (eng. operations).
Implementirana struktura podržava sljedeće operacije:
- *insert_enqueue(value, time)* - dodavanje operacije *enqueue* u određenom trenutku,
- *insert_dequeue(time)* - dodavanje operacije *dequeue* u određenom trenutku,
- *delete_operation(time)* - uklanjanje operacije izvršene u određenom trenutku,
- *get_first()* - vraća prvi dodat element,
- *get_last()* - vraća posljednji dodat element.

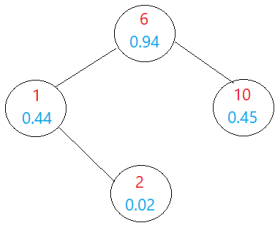
4.2. Implementacija parcijalno retroaktivnog steka

Implementacija parcijalne verzije retroaktivnog steka se u velikoj mjeri oslanja na implementaciju parcijalno retroaktivnog reda, sa nekim izmjenama. Koriste se slične klase: Node, Operation i PartiallyRetroactiveStack.
Podržane su sljedeće operacije:
- *insert_push(value, time)* - dodavanje operacije *push* u određenom trenutku,
- *insert_pop(time)* - dodavanje operacije *pop* u određenom trenutku,
- *delete_operation(time)* - uklanjanje operacije izvršene u određenom trenutku,
- *get_top()* - vraća posljednji dodat element.

4.3. Implementacija parcijalno retroaktivnog reda sa prioritetom

Retroaktivni red sa prioritetom je kompleksnija fstruktura od prethodno navedenih, jer jedna retroaktivna operacija može dovesti do kaskadnog efekta i promjene životnog ciklusa drugih elemenata. Zbog toga je i njena implementacija kompleksnija i oslanja se na pomoćnu strukturu treap (tree + heap).
Treap je binarno stablo koje istovremeno ima svojstva binarnog stabla pretrage (eng. binary search tree, BST) i strukture heap. Čvorovi ovog stabla sadrže dvije vrijednosti: ključ (eng. key) i prioritet (eng. priority). Ključevi omogućavaju održavanje svojstava binarnog stabla pretrage, a prioriteti su nasumične vrijednosti koje omogućavaju održavanje svojstava strukture heap.
Klase koje se koriste u implementaciji parcijalno retroaktivnog reda sa prioritetom su sljedeće:
- *TreapNode* – predstavlja jedan element strukture treap,
- *Treap* – pomoćna struktura za čuvanje elemenata,
- *MinPrefixSumAggregator* – predstavlja agregator prefiksnih suma koji podržava čuvanje prefiksne sume i intervala nad kojim je izračunata,
- *ZeroPrefixTreap* – modifikacija strukture treap za čuvanje prefiksnih suma,
- *PartiallyRetroactivePriorityQueue* – predstavlja parcijalno retroaktivni red sa prioritetom.

Klasa PartiallyRetroactivePriorityQueue sadrži četiri strukture: queue_now – treap koji čuva sve elemente koji se trenutno nalaze u redu, inserts – treap koji čuva sve operacije dodavanja elementa, deleted_inserts – treap koji čuva sve operacije uklanjanja elementa i bridges – ZeroPrefixTreap koji čuva bridge trenutke sa dodatnim informacijama.
S obzirom na to da se ova implementacija oslanja na metodu opisanu u ovom radu, razlikuju se dva slučaja: ubacivanje elementa u red prilikom dodavanja novog elementa ili uklanjanja operacije brisanja i uklanjanje elementa iz reda priliko brisanja najmanjeg elementa ili uklanjanja operacije dodavanja.
Prvi slučaj pokriven je metodama add_insert i remove. Metoda ad¬d_insert prvo privremeno dodaje novi element u strukturu deleted¬_insert, kako bi on prilikom izbora koji element treba da se ubaci u red bio posmatran jednako kao i svi obrisani elementi. Iz strukture deleted_inserts prvo se pronađe posljednji bridge trenutak, a zatim element sa maksimalnom vrijednošću. To je element koji će se vratiti u queue_now, što znači da to ne mora uvijek biti onaj koji je posljednji ubačen, nego onaj koji ima najniži prioritet za izbacivanje.
Drugi slučaj čine operacije inverzne prethodno navedenim, a pokrivaju ga metode add_delete_min i remove, pri čemu se bira najmanji element reda koji je na redu za izbacvanje.



Slika 1 Prikaz pomoćne strukture queue_now

Na slici 1. je prikazan primjer pomoćne strukture *queue_now*, gdje su crvenom bojom označeni ključevi, a plavom prioriteti.
Metode koje podržava retroaktivni red su sljedeće: *add_insert(time, value, data), add_delete_min(time), remove(time).*

Slika 2 Prikaz promjene stanja parcijalno retroaktivnog reda sa prioritetom

Na slici 2. dat je prikaz korišćenja parcijalno retroaktivnog reda sa prioritetom kroz vrijeme. Zelenom bojom označene su operacije dodavanja elementa, narandžastom operacije uklanjanja minimalnog elementa, a crvenom brisanje operacija.

4.4. Diskusija rezultata

Efikasnost retroaktivnih struktura podataka u literaturi se mjeri vremenskom složenošću algoritma, odnosno pomoću notacije „veliko O“ [5].
Kako se operacije kod retroaktivnog reda i steka čuvaju u pomoćnoj listi sortiranoj po trenucima izvršavanja, vrijeme izvršavanja jedne operacije podrazumijeva pronalaženje odgovarajućeg mjesta za novu operaciju u sortiranoj listi i ubacivanje samog elementa u listu. Ovaj korak se izvršava u vremenu $O(\log n)$, pri čemu je n

veličina liste. Međutim, samo ubacivanje ili brisanje elementa se izvršava u vremenu O(n). Algoritam bi se mogao unaprediti korišćenjem nekog oblika binarnog stabla za čuvanje operacija umjesto liste, kako bi se vijeme ubacivanja ili brisanja elementa svelo na logaritamsko vrijeme izvršavanja.

Što se tiče parcijalno retroaktivnog reda sa prioritetom, svaka operacija podrazumijeva dva koraka: pronalazak bridge trenutka i pronalazak maksimalnog elementa koji se ubacuje u red ili minimalnog elementa koji se uklanja iz reda. Oba koraka koriste pomoćne strukture zasnovane na implementaciji strukture treap, čije operacije dodavanja i uklanjanja se izvršavaju u logaritamskom vremenu, te je vrijeme izvršavanja jedne operacije $O(\log n)$.

5. PRIMJENA

U praksi postoji nekoliko slučajeva u kojima se mogu primijeniti retroaktivne strukture. Osnovna primjena je oporavak niza transakcija u slučaju greške brisanjem ili izmjenom transakcije koja je izazvala grešku u prošlosti. U radu [6] naveden je primjer gdje više meteoroloških stanica šalje podatke jednom centralnom sistemu i gdje u takvom sistemu retroaktivne strukture omogućuju naknadnu ispravku podataka, što utiče i na statističke podatke koji se računaju na osnovu tih zapisa, kao što je prosjek temperaturnih vrijednosti u nekom periodu.

Zatim, ove strukture se mogu primijeniti i u rješavanju nekih geometrijskih problema, kao na primjer lociranje tačaka u ravni (eng. *planar point location*) [1]. Takođe, moguća je i njihova primjena u dinamizaciji statičkih algoritama pomoću dinamičkog dodavanja i brisanja elemenata, kao na primjer dinamizacija algoritma *Dijkstra* [7].

6. ZAKLJUČAK

U ovom radu predstavljeno je istraživanje retroaktivnih struktura podataka, kao jedne od oblasti naprednih struktura podataka proučavanih na istoimenom kursu na univerzitetu MIT. Opisani su koncepti parcijalne i potpune retroaktivnosti, te je ilustrovana parcijalna retroaktivnost pomoću odabranih struktura: red, stek i red sa priotitetom.

Glavni fokus rada jeste opis metode implementacije parcijalno retroaktivnog reda i steka pomoću dvostruko povezane liste i implementacije parcijalno retroaktivnog reda sa prioritetom pomoću strukture treap. Zaključeno je da nije uvijek moguće implementirati efikasnu retroaktivnu strukturu, ali i da se ovim implementacijama može postići vrijeme izvršavanja slično odgovarajućim standardnim strukturama.

Jedan od pravaca u kojem bi se moglo nastaviti ovo istraživanje je prilagođavanje implementiranih struktura nekim praktičnim problemima.

Takođe, pošto su implementirane samo parcijalno retroaktivne verzije, sljedeći korak bio bi implementacija potpuno retroaktivnih struktura na osnovu opisane metodologije. Ovo bi otvorilo i mogućnost poređenja performansi implementiranih parcijalno i potpuno retroaktivnih struktura, kao i mogućnost izbora između parcijalno i potpuno retroaktivnih verzija u rješavanju praktičnih problema.

7. LITERATURA

[1] MIT Open Courseware, „Advanced Data Structures“, Prof. Erik Demaine, ocw.mit.edu, Stranica kursa „Napredne strukture podataka“, pristupano: maj 2023.

[2] E. D. Demaine, J. Ianoco, S. Langerma, „Retroactive Data Structures“, Proceedings of the Fifteenth Snnual ACM-SIAM Symposium on Discrete Algorithms, SODA 2004, New Orleans, Lousiana, USA, January 11-14, 2004.

[3] E. D. Demaine, T. Kaler, Q. Liu A. Sidford, A. Yedidia, „Polylogarithmic Fully Retroactive Priority Queues via Hierarchical Checkpointing“, MIT CSAIL Cambridge, MA, USA, 2015.

[4] J. W. Andrade Junior, R. Duarte Seabra, „Fully Retroactive Priority Queues using Persistent Binary Search Trees“, *Journal of Computer Science, 16(7),* pp. 906-915, July 2020.

[5] GeeksForGeeks, „Complete Guide On Complexity Analysis – Data Structure and Algorithms Tutorial“, geeksforgeeks.org, članak na sajtu GeeksForGeeks, pristupano: maj 2023.

[6] S. Agarwal, P. Panwaria, „Implementation, Analysis and Application of Retroactive Data Structures“, Proc. of the Intl. Conf. on Advances in Computer Science and Electronics Engineering, 2012.

[7] Sunita, D. Garg, „Dynamizing Dijkstra: A solution to dynamic shortest path problem through retroactive priority queue“, Journal of King Saud University - Computer and Information Sciences Volume 33, Issue 3, pp. 364-373, March 2021

Kratka biografija:

Vera Kovačević rođena je u Kozarskoj Dubici 1998. god. Fakultet tehničkih nauka u Novom Sadu, studijski program Softversko inženjerstvo i informacione tehnologije, upisala je 2017. godine. Nakon završenih osnovnih studija, 2021. godine, upisala je master akademske studije, smjer Softversko inženjerstvo i informacioni sistemi – Inteligentni sistemi.
kontakt:
verakovacevic98@gmail.com