ივანე ჯავახიშვილის სახელობის თბილისის სახელმწიფო უნივერსიტეტი
ზუსტ და საბუნებისმეტყველო მეცნიერებათა ფაკულტეტი

კომპიუტერული მეცნიერების დეპარტამენტი

ნიკოლოზ გრძელიძე

მეხუთე საკონფერენციო მოხსენება

BST ხის დაბალანსების ალბორითმები

თბილისი 2017

სარჩევი

BST ხის დაბალანსების ალგორითმები 2
ნაშრომის დასახელება 2
სამეცნიერო კვლევის აქტუალობა 2
DSW ალგორითმი 3
კვანძის სტრუქტურა 5
ორობითი ხის სტრუქტურა 5
ტესტირების შედეგები 6
ზრდადობით შემომავალი მონაცემები 6
კლებადობით შემომავალი მონაცემები 7
შემთხვევით შემომავალი მონაცემები 8
კვლევის მიზანი 8

BST ხის დაბალანსების ალგორითმები

ნაშრომის დასახელება

ორობითი ხის დაბალანსების, სწრაფი ალგორითმის საჭიროება წარმოიშვა, სადოქტორო თემის - მონაცემთა ჰიბრიდული სტრუქტურების შემუშავებისას. არსებული იმპლემენტაცია, დაფუძნებულია სხვადასხვა ტიპის სტრუქტურების უპირატესობების გამოყენებით, ჰიბრიდული სტრუქტურის შექმნაზე, რაც არ იძლევა ჩვენთვის დამაკმაყოფილებელ გაუმჯობესებას. ამიტომ შემუშავებულია, დაბალანსების ახალი QBalance ალგორითმი.

სამეცნიერო კვლევის აქტუალობა

პრაქტიკაში უმეტესად გამოიყენება თვით დაბალანსებადი(Self-Balanced) ორობითი ხეები, მაგალითად AVL, RBT. ძირითად არეალს წარმოადგენს პროგრამული ენების სტანდარტული ბიბლიოთეკები, კერძოდ, კოლექციები. ვიცით, რომ თვით ბალანსირებადი ძებნის ორობითი ხეები, უზრუნველყოფენ დინამიურ დაბალანსებას მოდიფიკატორი(mutator) ოპერატორების გამოყენებსი შემდეგ, თუმცა პრაქტიკაში არის ამოცანები სადაც წინასწარ არის ცნობილი, როდის უნდა დაბალანსდეს ძებნის ორობითი ხე. შესაბამისად თვით ბალანსირებადი ძებნის ორობითი ხეების გამოყენება ასეთ შემთხვევაში ძვირიანი გადაწყვეტილებაა. ძებნის ორობითი ხის დაბალანსების ალგორითმი, არ წარმოადგენს სიახლეს. შემოთავაზებულია დაბალანსების სხვადასხვა ალგორითმები, მაგალითად: DSW(სტანდარტული), DSW(მოდიფიცირებული), sedgewick-ის ალგორითმი. ასეთი ალგორითმების შემთხვევაში იგულისხმება, რომ მონაცემთა სტრუქტურაზე ხორციელდება Bulk ჩასმა, და Bulk წაშლა შემდეგ კი სტრუქტურის დაბალანსება.

თითქმის ყველა შემთხვევაში ამოსავალ წერტილს წარმოადგენს სტრუქტურის დალაგებულ სიად გარდაქმნა, შემდეგ კი მისი დაბალანსება.

DSW ალგორითმი

1976 წელს Colin Day-მ შეიუმუშავა ორობითი ხის დინამიკური დაბალანსების ალგორითმი, რომელიც თავიდან ირიდებს ოპერაციების გარკვეულ რაოდენობას, რასაც AVL გვთავაზობს. (თუმცა ის მაინც რჩება ორობითი ხის ბალანსირების დინამიკურ ალგორითმად). Day-ის ალგორითმის პირველი რეალიზაცია შესრულებული იყო fortran-ზე. რეკურსიის და კლასების უქონლობამ თავისებურად გაართულა იგი. ალგორითმი შედგება ორი ფაზისგან:

1. ორობითი ხისაგან გადაგვარებული ხის აგება - ნებისმიერ კვანძს ჰყავს მარჯვენა შვილი.

2. გადაგვარებული ხისაგან დაბალანსებული ხის აგება - compression მეთოდის გამოყენებით.

გადაგვარებული ხის მისაღებად Day იყენებდა ბმულ სიას, რომელიც მოითხოვდა დამატებით მეხსიერებას. პროგრამირების ენების განვითარებასთან ერთად გაჩნდა ახალი შესაძლებლობები და ალგორითმმაც ცვლილება განიცადა.

10 წლის შემდეგ Quentin F. Stout-მა და Bette L. Warren-მა ცვლილებები შიტანეს Colin Day-ის ალგორითმში. ძირეული ცვლილება განიცადა ალგორითმის პირველმა ფაზამ და მცირედი - მეორემ. Stout-მა და Warren-მა შენიშნეს, რომ პირველ ფაზაში, გადაგვარებული ხის აგება შესაძლებელია კვანძის მობრუნების გამოყენებით, კერძოდ, მარჯვნივ მობრუნებების ოპერაციით. მათ იმპლემენტაციაში შემოიღეს ფსევდო-ფესვი -მარჯვენა შვილის გარანტირებული არსებობისათვის, რაც ალგორითმს ხდის უფრო მარტივს და თავიდან გვარიდებს განშტოების ზედმეტ ოპერაციებს. მარჯვნივ ტრიალი ხორციელდება მანამ, სანამ კვანძის ჰყავს მარცენა შვილი. ამის შემდეგ ალოგირთმი ჩადის ერთი დონით დაბლა და იმეორებს იგივე ოპერაციას. მიღებულ სტრუქტურას ავტორებმა უწოდეს vine tree აღნიშნულ მეთოდს კი - tree_to_vine.

ალოგირითმის მეორე ფაზაში ხდება მობრუნებების (მარცხნივ მობრუნება) ოპტიმალური K რაოდენობების დათვლა. K არგუმენტად გადაეცემა ფუნქციას, რომელიც K ჯერ მოახდენს vine tree-ს მობრუნებას მარცხნივ, რის შედეგადაც მიიღება დაბალანსებული ორობითი ხე.

ალგორითმის პირველი ფაზა, tree_to_vine ფსევდო კოდი:

-tmp = root // დროებითი ცვლადი, თავიდან დგას ფესვზე.

-while (tmp != NULL)

-თუ tmp ჰყავს მარცხენა შვილი:

    -შეასრულე მარჯვნივ მობრუნება ამ წვეროსთვის.

    -tmp-ს მიანიჭე მარცხენა შვილი, რომელიც ახლა მშობელი გახდა.

-სხვა შემთხვევაში: tmp-ს მიანიჭე მარჯვენა შვილი.

როცა ხეში n ცალი წვეროა, საუკეთესო შემთხვევაში, თუ მონაცემთა სტრუქტურა უკვე გადაგვაერბულია(vine tree), ციკლი სრულდება n-ჯერ და არცერთი კვანძისათვის არ სრულდება, მობრუნების ოპერაცია. ყველაზე ცუდ შემთხვევას წარმოადგენს, ორობითი ხე, რომელის თითოეულ კვანძს მხოლოდ მარცხენა შვილი ჰყავს, ასეთ შემთხვევაში, რეკურსიის სიღრმე არის 2n – 1, კვანძის მარჯვნივ მობრუნების ოპერაციის რაოდენობა კი n – 1. შესაბამისად პირველი ფაზის -გადაგვარებულ ხედ გარდაქმნა, დროითი შეფასება არის წრფივი, O(n).

ალგორითმის მეორე ფაზა, vine_to_tree ფსევდო კოდი:

-m = 2|lg(n+1)| - 1;

-გააკეთე m - n მობრუნება ხის მიმდინარე ფესვიდან.

-while (m > 1)

-m = m / 2;

-შეასრულე m მობრუნება ხის მიმდინარე ფესვიდან.

მეორე ფაზის დროითი, სირთულის შესაფასებლად დავითვალოთ მობრუნებების რაოდენობა:

(2lg(m+1) -1 - 1) + … + 15 + 7 + 3 + 1 = i = 1lg(m+1)-1(2i -1) = m - lg(m+1)

მობრუნებების რაოდენობა შეგვიძლია წარმოვადგინოთ ფორმულით:

n - m + (m - lg(m + 1)) = n - lg(m+1) = n - |lg(n+1)|

DSW ალგორითმის დროითი შეფასება არის წრფივი, O(n).

QBalance

ალგორითმი, ორობითი ხის დასაბალანსებლად იყენებს მისი წარმოდგენას სიის სახით. ორობითი ხის სიად წარმოდგენისათვის, შემუშავებული იქნა იტერაციული მეთოდი -treeToList, რომელიც სტრუქტურის სიად გარდასაქმნელად იყენებს სტეკს, რაც, ცხადია, ითხოვს დამატებით რესურს. აღნიშნული მიდგომის შედარებამ უკვე ცნობილ, tree-to-vine(მცირედი მოდიფიკაციით -ოპერირება, შვილების მასივზე) ვერ აჩვენა უკეთესი შედეგი. შესაბამისად სიად გარდაქმნისათვის ალგორითმი იყენებს tree-to-vine მოდიფიცირებული ვარიანტს. ალგორითმის მეორე ეტაპი, არის პირველ ეტაპზე მიღებული სტრუქტურის დაბალანსება, მეთოდი -qBalance. დაბალანსების მეთოდი რეკურსიულია, რეკურსიის ბაზას წარმოადგენს ერთ ელემენტიანი სია, რომელიც, ცხადია, დაბალანსებულია. ალგორითმის იდეა შემდეგია, ბმული სია იყოფა ორ

ნაწილად, ინდექსით შუა ელემენტის მიმართ, რომელიც უნდა გახდეს ფესვი, მის მარცხნივ და მარჯვნივ მყოფი, დაუბალანსებელი სიისათვის. გაყოფის შემდეგ ალგორითმი რეკურსიულად ეშვება გაყოფის შემდეგ მიღებული სეგმენტებისათვის. ალგორითმი რეკურსიის ბაზიდან, დაბრუნებისას იყენებს წინასწარ დამახსოვრებულ კვანძს, რომლის შვილის უნდა გახდეს მის მერ დაბალანსებული ქვეხის ფესვი. ალგორითმის პროგრამული რეალიზაცია შემდეგია:

```
qBalance(int st, int fin){

        if (fin < st)  return NULL;
        if (fin == st)
        {
                Node<T>* tmp = head;
                head = head->child[1];
                tmp->child[0] = tmp->child[1] = NULL;
                return tmp;
        }
        int q = (st + fin) / 2;
        Node<T>* a = qBalance(st, q - 1);
        Node<T>* b = head;
        b->child[0] = a;
        head = head->child[1];
        a = qBalance(q + 1, fin);
        b->child[1]
        return b;
}
```

კვანძის სტრუქტურა

კვანძის სტრუქტურა მინიმალისტურია, ამოღებულია მშობელ კვანძზე მიმთითებელი. კვანძის შვილები წარმოდგენილია ორ ელემენტიანი მასივის სახით, რომლის უპირატესობა, შესაბამისი ტესტებით არის ნაჩვენები. შვილების მასივად წარმოდგენა აგრეთვე ამარტივებს ალგორითმის იმპლემენტაციას, პროგრამულ რეალიზაციას უფრო ელეგანტურს ხდის.

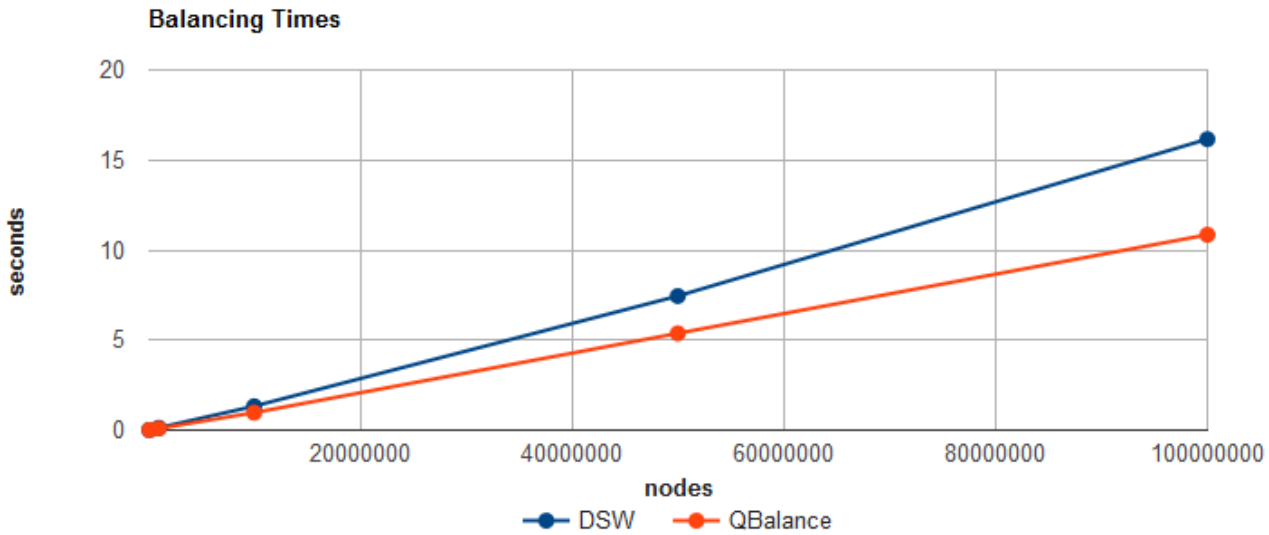
ორობითი ხის სტრუქტურა

ძებნის ორობითი ხის სტრუქტურა, დაბალნსების QBalance ალგორითმის მოთხოვნებიდან გამომდინარე, შეიცავ ინფორმაციას კვანძების რაოდენობაზე, რადგან აღნიშნული თვისების დადგენა არ მოხდეს დინამიურად, წრფივ დროში. აგრეთვე

სტრუქტურა შეიცავს, სტანდარტულ, ხის ფესვს და დროებით ფესვს, ალგორითმის სპეციფიკიდან გამომდინარე.

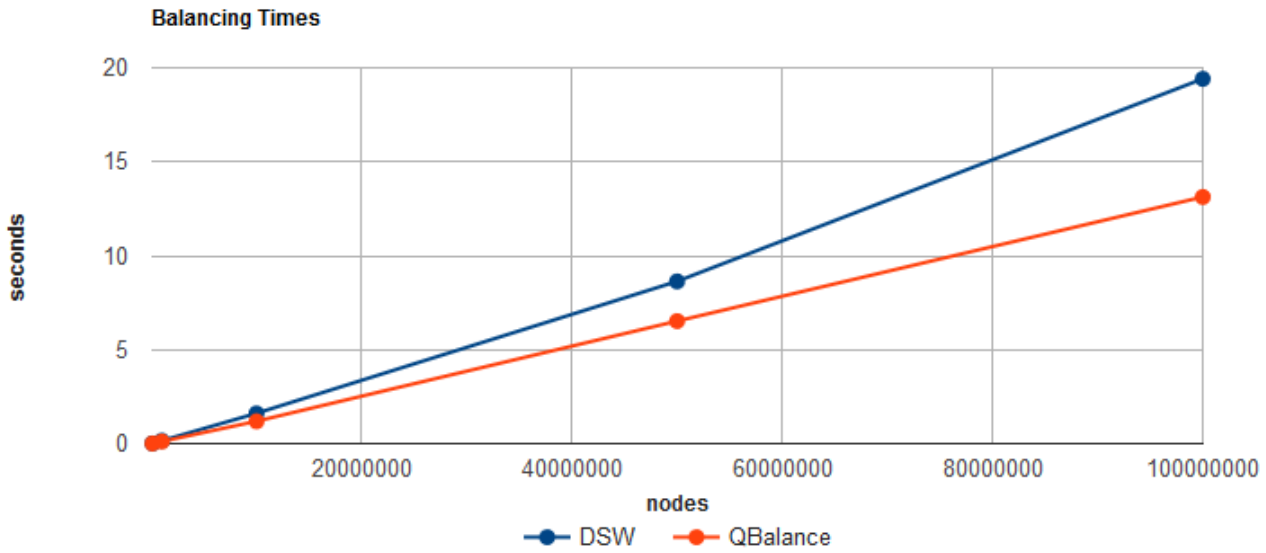
ტესტირების შედეგები

ზრდადობით შემომავალი მონაცემები



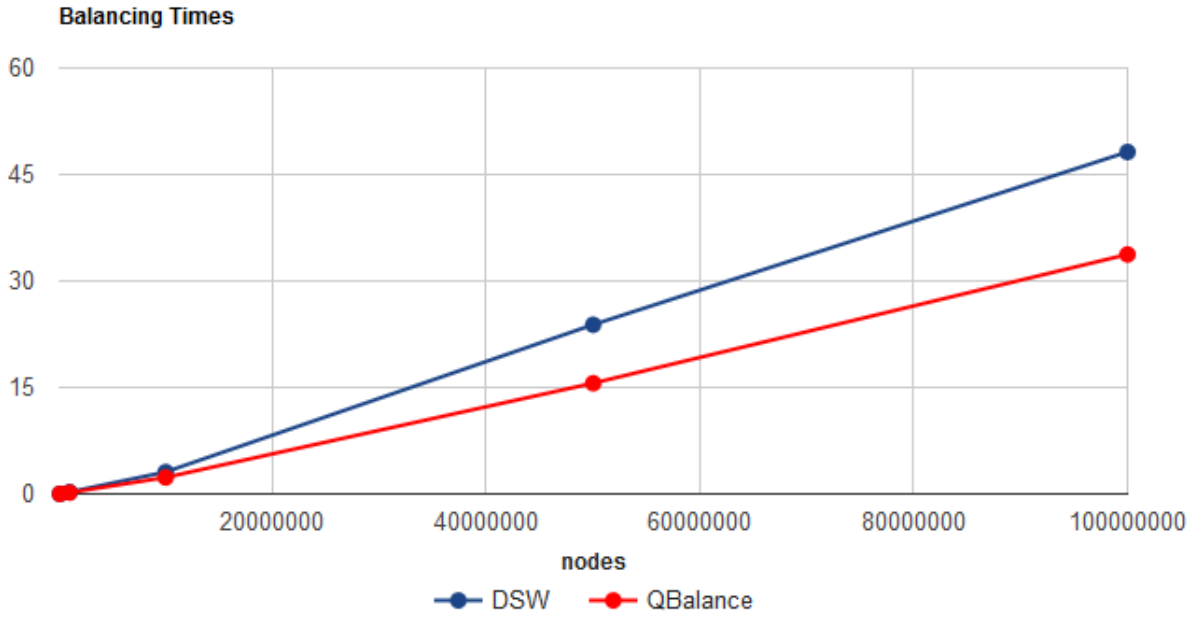
Tree size	100000	1000000	10000000	50000000	100000000
DSW	0.008	0.13	1.329	7.446	16.16
QBalance	0.009	0.091	0.97	5.375	10.85

კლებადობით შემომავალი მონაცემები



Tree size	100000	1000000	10000000	50000000	100000000
DSW	0.014	0.155	1.612	8.634	19.414
QBalance	0.009	0.114	1.19	6.52	13.126

შემთხვევით შემომავალი მონაცემები



Tree size	100000	1000000	10000000	50000000	100000000
DSW	0.015	0.264	3.092	23.841	48.175
QBalance	0.015	0.188	2.32	15.6	33.742

კვლევის მიზანი

სადოქტორო დისერტაციის ფარგლებში იგეგმება სხვადასხვა ჰიბრიდული მონაცემთა სტრუქტურის განსაზღვრა, იმპლემენტირება და გამოყენება პრაქტიკულ ამოცანებში. ერთ-ერთ მათგანში ვფიქრობთ დაბალანსებული ძებნის ორობითი ხის და ჰეშ-ცხრილის ფუნქციების გაერთიანებას.

ასეთი სტრუქტურა ძალიან სასარგებლო იქნება მეჩხერი (sparse) მარტიცების გადამრავლებითვის და ასეთი მატრიცის ხარისხების განსაზღვრისთვის.

თვითონ მეჩხერი მატრიცების გადამრავლება საკმაოდ იშვიათად გამოიყენება პრაქტიკაში, ბევრად უფრო ხშირად გამოყენებადია მეჩხერი მატრიცის (ჩვეულებრივ) ვექტორზე გადამრავლების ოპერაციები, რის გამოც შესაბამისი ალგორითმები კარგადაა დამუშავებული.

თუმცა, ჩვენ შეგვიძლია მივუთითოთ რამდენიმე პრაქტიკულად საინტერესო ამოცანა, სადაც ასეთი აუცილებლობა ჩნდება.

ცნობილია, რომ მატრიცების ახარისხების საჭიროება ჩნდება, როდესაც გვინდა ორიენტირებული გრაფის ყველა წვეროებს შორის უმოკლესი მანძილების განსაზღვრა.

მატრიცების ახარისებაზე დაფუძნებული ალგორითმი არაა ყველაზე სწრაფი, თუმცა საკმაოდ ცნობილია და მისი აჩქარება შესაძლებელია ახალი სტრუქტურის გამოყენეით ისეთი გრაფებისთვის, რომლებშიც წიბოების რაოდენობა შორსაა შესაძლო მაქსიმუმისგან (მაგალითად, პლანარული გრაფები).
მეორე, ბევრად უფრო ბუნებრივი გამოყენება შეგვიძლია მოვიყვანოთ ბიოლოგიიდან. ესაა ლესლის მატრიცა, ძალიან მპოპულარული მოდელი პოპულაციათა ეკოლოგიაში. ამ მატრიცის ხარისხები აღწერენ პოპულაციის მნიშვნელოვანი მახასიათებლების ცვლილებას დროში. მატრიცა უმეტესწუილად ნულებისგან შედგება, ამიტომ ჩვენი მიდგომა აქ პერსპექტიოული ჩანს.
მოკლედ აღვწეროთ თუ რა განასხვავებს ჩვენს მიდგომას უკვე არსებული მეთოდებისგან, რომლებიც გამოიყენება მეჩხერი, ანუ გაიშვიალლებული მონაცემების დამუშავებისთვის.
ერთ-ერთი ყველაზე სანდო ბიბლიოთეკა არის boost -ბიბლიოთეკა. იგი ამ მიზნიტ იყენებს წითელ-შავი ხის საფუძველზე შექმნილ მეჩხერი მატრიცის ადაპტერს, რომელიც მარტიცის ვექტროზე გამრავლებისთვის, მატრიცის სტრიქონებს გაივლის წრფიც დროში იტერატორის გამოყენებით, ხოლო სტრიქონის ელემენტების ამოღება დამოუკიდებლად ხდება. როდესაც მარტიცებს ვამრავლებთ ერთმანეთზე, მატრიცის სტრიქონები წრფივ დროში გაისინჯება (და ნულოვანი ელემენტები გამოიტოვება), მაგრამ ამავე დროს, სვეტების ელემენტების ამოსაღებად საჭიროა შენახული ელემენტების როადენობის ლოგარითმის პროპორციული დრო. შევნიშნოთ, რომ აქ ეკონომიას ვერ ვაღწევთ, ყველა ელემენტი უნდა მოვძებნოთ, არის ჩაწერილი თუ არა. იმ შემთხვევაში, როდესაც ვიყენებთ ჰიბრიდს, ელემენტის ძებნა ხორციელდება მუდმივ დროში, რაც გვაძლევს სასურველ საგრძნობ განსხვავებს პროცედურის მრავლალჯერ განხორციელების შემთხვევაში. ჰიბრიდის სტრუქტურა საკმაოდ რთულია და ამიტომ მასზე აქ არ შევჩერდებით.