თავი 8. გადათვლით დახარისხება

განხილული საკითხები:

- დახარისხების (sorting) ამოცანა
- გადათვლით დახარისხების ალგორითმი >>>
- ლიტერატურა >>>

დახარისხების (sorting) ამოცანა

ამ ამოცანაში, **შემავალ მონაცემებს** წარმოადგენენ: დახარისხების სახე (ზრდადობა ან კლებადობა), ჩანაწერები $R_0, R_1, \ldots, R_{n-1}$ და ამ ტიპის ჩანაწერის ერთ-ერთი (წინასწარ მოცემული) რიცხვითი ველი-გასაღები (უფრო რთულ შემთხვევებში, ჩანაწერითან ასოცირებული რიცხვითი სიდიდე (გასაღები), ან ბინარული პრედიკატი, რომელიც გვეუბნება ნებისმიერი ორი ასეთი ჩანაწერიდან რომელია უფრო ნაკლები). ვთქვათ, k_i აღნიშნავს იმ რიცხვითი ველის მნიშვნელობას R_i ჩანაწერისთვის, რომლის მიმართაც გვინდა ჩანაწერების დახარისხება.

გამომავალ მონაცემებს წარმოადგენენ ისეთნაირად გადაადგილებული ჩანაწერები, რომ მათი შესაბამისი k_i სიდიდეები დალაგებულია ზრდადობის (ან კლებადობის) მიხედვით.

მაგალითად, წარმოვიდგინოთ, რომ შემდეგი ფაილი შედგება ბანკის მეანაბრეთა (გამარტივებული) მონაცემებისაგან:

გვარი	სახელი	ასაკი	ანაბარი ლარებში	ანაბარი დოლარებში
გეგეშიძე	გიორგი	75	3000	6000
მესხი	პეტრე	40	25000	2500
საბაძე	პავლე	35	2000	15000

და ა. შ.. თითოეული სტრიქონი არის ჩანაწერი, რომელიც შედგება ხუთი განსხვავებული ველისგან. სხვადასხვა შემთხვევაში, შესაძლოა საჭირო გახდეს ჩანაწერების დახარისხება ისე, რომ ანაბარი დოლარებში დალაგდეს ზრდადობის მიხედვით (მესხი, გეგეშიძე, საბაძე) და ამ შემთხვევაში ჩანაწერის გასაღებს წარმოადგენს ველი "ანაბარი დოლარებში". სხვა შემთხვევაში, მაგალითად ბანკთან არსებული სადაზღვევო ფირმისთვის შესაძლოა საჭირო გახდეს ჩანაწერების გადალაგება ისე, რომ მეანაბრეთა ასაკი დალაგდეს კლებადობის მიხედვით (ამ შემთხვევაში უკვე დალაგებულია ამ გასაღების მიხედვით).

ქვემოთ განვიხილავთ გამარტივებულ შემთხვევებს, როდესაც ყოველი ჩანაწერი გაიგივებულია თავის გასაღებთან და წარმოადგენს ნამდვილ რიცხვს. ამ შემთხვევაში ჩვენს ამოცანას წარმოადგენს რიცხვების დალაგება ზრდადობით ან კლებადობით.

<<< გადათვლით დახარისხების (Counting Sort) ალგორითმი

დახარისხების ეს მეთოდი მხოლოდ კერძო შემთხვევებშია გამოყენებადი, როდესაც დასალაგებელი კონტეინერის ელემენტები არის მთელი რიცხვები, და დიაპაზონი არის კონტეინერის ელემენტების რაოდენობის რიგის. მეთოდის სისწრაფე პროპორციულია კონტეინერის ელემენტების მნიშვნელობათა დიაპაზონისა და კონტეინერის ელემენტების რაოდენობის ჯამისა. მეთოდის გამოყენებისთვის საჭიროა დამხმარე კონტეინერი, რომლის ელემენტების რაოდენობა ტოლია დასახარისხებელი კონტეინერის დიაპაზონის (ანუ, უდიდესი და უმცირესი ელემენტის სხვაობის). ჩვენ განვიხილავთ კიდევ უფრო გამარტივებულ შემთხვევას, როდესაც კონტეინერის ელემენტები არაუარყოფითებია (თუმცა ამ შეზღუდვის მოხსნა ძალიან ადვილია).

გადათვლით დახარისხების დროს საწყისი კონტეინერის ელემენტები განიხილება როგორც სხვა (დამხმარე) კონტეინერის ინდექსები. ალგორითმის იდეა შემდეგია: თუ

კონტეინერის ნებისმიერი x ელემენტისათვის წინასწარ დავთვლით ამ კონტეინერის რამდენი ელემეტია x-ზე ნაკლები (ვთქვათ m), მაშინ ის შეგვიძლია საბოლოო კონტეინერში პირდაპირ ჩავწეროთ (m+1)-ე ადგილზე, ანუ ინდექსით m. თუ შემავალ კონტეინერში გვხვდება ერთმანეთის ტოლი რიცხვები, მაშინ დამატებით უნდა ვიზრუნოთ, რომ ტოლი რიცხვები ერთმანეთის მეზობლად და ძველი რიგის შენარჩუნებით განლაგდეს.

განვიხილოთ პროგრამული კოდი. ვიგულისხმოთ, რომ კონტეინერი ვექტორია.

ვთქვათ, საწყისი ვექტორი არის a, დამხმარე ვექტორი არის c, ხოლო b ვექტორი არის საბოლოო, ანუ პასუხი.

სტრიქონ 1-ში განისაზღვრება დამხმარე ვექტორის უდიდესი ინდექსი, - რადგან დამხმარე ვექტორის ინდექსებს წარმოადგენენ საწყისი ვექტორის ელემენტები. შემდეგ, ელემენტების რაოდენობის მისაღებად, ეს სიდიდე უნდა გაიზარდოს ერთით, რადგან უდიდიესი ინდექსი ელემენტების რაოდენობაზე ერთით ნაკლებია.

დამხმარე ვექტორის ინიციალიზაციის დროს (სტრიქონი 3), დამხმარე c ვექტორი იქმნება და ივსება ნულებით. დამხმარე ვექტორში ზუსტად იმდენი ნულია, რამდენიც არის ელემენტების დიაპაზონი საწყუის ვექტორში.

სტრიქონებში 4-5, დამხმარე c ვექტორის c[i] ელემენტი ხდება a ვექტორის იმ ელემენტების რაოდენობის ტოლი, რომლებიც i ინდექსის ტოლია.

სტრიქონებში 6 და 7, დამხმარე c ვექტორში c[i] ელემენტად იწერება a ვექტორის იმ ელემენტების რაოდენობა, რომლებიც არ აღემატებიან i ინდექსს.

ბოლოს, ყოველი ელემენტი თავსდება b ვექტორში მის ადგილზე (სტრიქონები 8 და 9). ეს ადგილი განისაზღვრება შემდეგნაირად. თუ შემომავალ ვექტორში ყველა ელემენტი განსხვავებულია, მაშინ დალაგებულ ანუ გამომავალ ვექტორში $a[i]$ მოთავსდება c[a[i]]-1 ინდექსით, რადგან ზუსტად ამდენი ელემენტი არის $a[i]$- ზე ნაკლები; თუ შემავალი მასივი შეიცავს ერთმანეთის ტოლ ელემენტებს, მაშინ ელემენტები მოთავსდება ისევ c[a[i]]-1 ინდექსით, მაგრამ ყოველი ელემენტის ჩაწერისას c[a[i]] უნდა შევამციროთ 1-ით (ერთმანეთს რომ არ გადაეწეროს).

მიუხედავად თავისი კერძოობისა, ამ მეთოდს ორი უპირატესობა აქვს ბევრ სხვა მეთოდთან შედარებით: მისი შესრულების დრო, როგორც ჩანს ცხრილიდან, არის $\Theta(n+k)$ რიგის და ეს არის მდგრადი ალგორითმი, რაც ნიშნავს შემდეგს: საწყის მასივში ერთმანეთის ტოლი ელემენტები საბოლოო (უკვე დალაგებულ) მასივში ინარჩუნებენ თავიანთ ფარდობით რიგს ერთმანეთის მიმართ.

	countingSort(vector<**int**>& a, vector<**int**>& b) ალგორითმის სტრიქონები	დრო	სტრიქონის განმეორებათა რიცხვი
1	`int k = *max_element(a.begin(), a.end());`	$\Theta(n)$	1
2	`k++;`	c_1	1
3	`vector<int> c(k);`	$\Theta(k)$	1
4	`for (int i=0; i< a.size(); i++)`	c_2	(n +1)
5	`    c[a[i]]++;`	c_3	(n)
6	`for (int i=1; i<k; i++)`	c_4	(k)
7	`    c[i] += c[i-1];`	c_5	(k - 1)

8	`for (int i= a.size()-1; i>=0; i--)`	c_6	`( n +1 )`
9	`b[c[a[i]]-1] = a[i];` `c[a[i]]--;`	c_7	`( n )`

განვიხილოთ მაგალითი.

ინდექსი	0	1	2	3	4	5	6	7	8
საწყისი ვექტორი a	5	2	7	4	4	2	1	9	2

შემდეგ ცხრილში, ქვედა სტრიქონში არის დამხმარე ვექტორი c, 4-5 სტრიქონების შესრულლების შემდეგ. c არის 10 ელემენტიანი ვექტორი, რადგან a ვექტორის უდიდესი ელემენტი არის 9 – ის ტოლი. ამ ეტაპზე, c ვექტორში აღრიცხულია a ვექტორის ელემენტების რაოდენობები ცალ-ცალკე.

0	1	2	3	4	5	6	7	8	9
0	1	3	0	2	1	0	1	0	1

შემდეგ ცხრილში, ქვედა სტრიქონში არის დამხმარე ვექტორი c, 6-7 სტრიქონების შესრულების შემდეგ. ამ მომენტისათვის, c ვექტორი უკვე შეიცავს ინფორმაციას იმის შესახებ, თუ რამდენ ელემენტს არ აღემატება a ვექტორის თითოეული წევრი.

0	1	2	3	4	5	6	7	8	9
0	1	4	4	6	7	7	8	8	9

საბოლოო b ვექტორის რომლის ზომა ტოლი უნდა იყოს საწყისი ვექტორის ზომის, და თავიდან ის ნულებითაა შევსილი. შემდეგ იგი ივსება პროგრამის 8-9 სტრიქონების მიხედვით.

<<< ლიტერატურა

1. დ. მასერი. ბმული სიის კოდი Hewlett-Packard Company-ის STL -ის მიხედვით: http://www.cs.rpi.edu//~musser/gp/List/
2. T.Cormen, C. Leiserson, R. Rivest, C. Stein. Introduction to Algorithms, Third Edition. The MIT Press, 2009, გვ. 104-196