In-place a in situ algoritmy

I.Kolingerová

Obsah:

1. In-place algoritmus
2. In situ algoritmus
3. Výhody in-place a in situ
4. Příklady
5. In place sorting

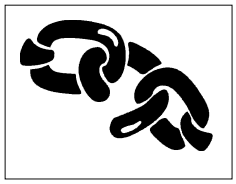
1. In-place algoritmus

- Narozdíl od data stream algoritmu se předp. vstup v poli
- Transformuje datové struktury s užitím malé, O(1) dodatečné paměti
- Vstup během chodu programu obvykle přepsán výstupem – není nutný ani postačující rys – výstup může být konst.
- Struktura vlastně zakódována jako permutace vstupu

2

Př.: Otočení řetězce

```
function reverse (a[0..n])
        allocate b[0..n]
        for i from 0 to n
            b[n-i] = a[i]
return b
x
function reverse_in_place (a[0..n])
        for i from 0 to floor(n/2)
            swap (a[i],a[n-i])
```



3

2. In situ algoritmus

- Někdy do in-place řazeny také alg., které potřebují kromě vstup. pole O(log n) extra paměť
- <= pro omez. data sice pointer O(1) paměť, ale pro lib. velká data potřebuje O(log n) bitů pro specifikaci indexu do seznamu délky n – obvykle se toto ale ignoruje
- Někdy tyto alg. ozn. jako in situ

4

3. Výhody in-place a in situ

- Jde zpracovat větší dat. množiny
- Větší lokalita reference – vhodné pro paměť. hierarchie
- Méně náchylné k selhání – nevyžaduje velké objemy paměti, kt. nemusí v době běhu být dostupná

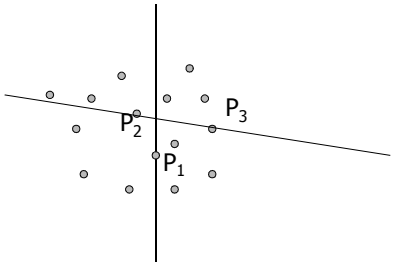
5

4. Příklady

- Jeden z hlavních triků: ukládat pointery implicitně permutací bloku dat
- Např. Willardův 2D dělicí strom - n static. bodů rekurzivně rozděleno do 4 oblastí 2 přímkami, 1. svislá bodem p_1 s mediánem v x, 2. body p_2, p_3 voleny tak, aby počet bodů v každé oblasti byl přesně (n-3)/4 (lze - tzv. ham-sandwich teorém), zabírá O(n)

6

Willardův dělicí strom



7

Willardův 2D dělicí strom

- Prostorově efektivní verze: p_1, p_2, p_3 v čele pole, zbytek rozdělen na 4 podpole stejné velikosti, na nich rekurze => 0 prostor navíc
- Prohledávání v v sublineárním čase, O(1) místo, předzprac. O(n log n)
- Možná i semidynamizace (=vkládání)

p_1	p_2	p_3	...	...										

8

Př.: Prohoďte 2 posloupnosti v lineárním čase, s užitím O(1) dodatečné paměti

A=LSP, LSP ->PSL
(posloupnosti L,S,P mohou mít různou délku => problém)

1. Zrcadlově obrať L,S,P
2. Zrcadlově obrať celé A

Př.: 1 2 3 4 ¦ 5 6 7 ¦ 8 9 10 11 12
Ad 1) 4 3 2 1 ¦ 7 6 5 ¦ 12 11 10 9 8
Ad 2) 8 9 10 11 12 ¦ 5 6 7 ¦ 1 2 3 4

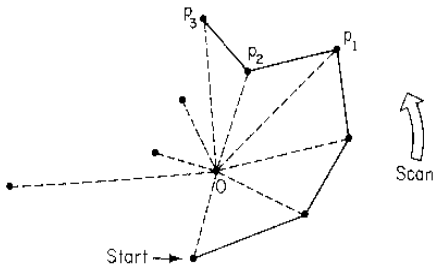
9

Konvexní obálky

- Aplikace prost. efekt. verze - vrcholy *CH(S)* - prefix téhož pole
- Úprava Grahamova algoritmu: potřebuji In-place sort

10

Grahamovo prohledávání



Start: extrémní bod (nejpravější bod s nejmenší souř.*y*)

1. *p*1*p*2*p*3 je pravotočivý: Eliminuj vrchol *p*2 a zkontroluj *p*0*p*1*p*3

2. *p*1*p*2*p*3 je levotočivý: Zkontroluj *p*2*p*3*p*4

Kritérium vyřazení bodu:
Úhel *p*1*p*2*p*3 ≥ π (pravotočivost)

11

Graham_In_Place_Scan

S - vstup. body, n - počet, d = 1 - sort rostoucí, horní *CH(S)*,
-1 - sort klesající, dolní *CH(S)*

```
1.  Graham_In_Place_Sort (S,n,d)
2.  h = 1
3.  for i := 1 to n-1 do
4.      while (h >= 2) and not right_turn(S[h-2],S[h-1],S[i])
5.          h := h-1  { pop top element from the stack }
6.      end while
7.      swap S[i] and S[h]
8.      h := h + 1
9.  end for
10. return h
```

Vrací dolní nebo horní *CH(S)* v S[0] až v S[h-1],
horní *CH(S)* uložena CW zleva doprava,
dolní CW zprava doleva

12

Graham_In_Place_Hull (S,n)

S - vstup. body, n - počet, d = 1 - sort rostoucí, horní *CH(S)*,
-1 - sort klesající, dolní *CH(S)*

1. h := Graham_In_Place_Scan(S,n,1) // O(n log n)
2. **for** i := 0 **to** h-2 **do** // O(h)
3. swap S[i] and S[i+1]
4. **end for**
5. h´ := Graham_In_Place_Scan (S+h-2,n-h+2,-1)
6. **return** h+h´- 2

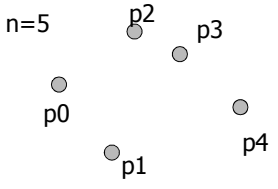
Celkem O(n log n) čas, O(1) extra paměť

Vylepšení: nalézt extrémy, rozdělit pole na 2 části -
- pro horní a dolní *CH(S)*, pak každý bod
jen v 1 volání Scan
Větší vylepšení: pomocí MergeSort

13

Konvexní obálka - příklad (1)

Po setřídění podle x:



S:

p0	p1	p2	p3	p4	...
p0	p2	p1	p3	p4	...
p0	p2	p3	p1	p4	...
p0	p2	p3	p4	p1	...

h	i	Akce
1	1	Swap S1,S1 ☺
2	2	kontrola S0,S1,S2 (p0,p1,p2), dec(h)
1	2	Swap S2,S1
2	3	kontrola S0,S1,S3 (p0,p2,p3), OK, swap S2,S3
3	4	kontrola S1,S2,S4 (p2,p3,p4),OK, swap S3,S4

14

Konvexní obálka - příklad (2)

Po výpočtu horní CHS:

levý bod	horní konvexní obálka	pravý bod	...

Pak přesun levého bodu doprava:

horní konvexní obálka	pravý bod	levý bod	...

Po výpočtu dolní konvexní obálky:

horní konvexní obálka	pravý bod	dolní konvexní obálka (uložena obráceně)	levý bod	...

Výstup:

Konvexní obálka	

15

5. In-place a in situ sorting

a) spojité pole – konst. čas na přístup a prohození, dlouhý čas na posuvy

Pokud ignorujeme O(log n) na pointery:

Heapsort – ano, konst. dodatečná paměť

Quicksort – ano, O(log n) (v nejhorším případě O(n)) dodat. paměť na D&C rekurzi

Mergesort – ne

b) zřetěz. seznamy – vyhled. podle indexu O(n)

⇒ quicksort a heapsort – nutné drastické modif., aby alesp. $O(n^2)$

Mergesort – O(n log n) čas, O(log n) dodat. paměť

16