

**Maksimum drsečega okna**

Timen Bobnar

Ljubljana, 2023

# **Povzetek**

Na spletni strani LeetCode ([1]) je predstavljena naloga maksimuma drsečega okna. Zanimajo nas vsi maksimumi v okencu dolžine *k*, ko se sprehajamo po seznamu števil. Ta problem želimo rešiti z uporabo vrste. Najprej si bomo ogledali način predstavitve vhodnih podatkov problema, ter s pomočjo primera predstavili problem. V nadaljevanju si bomo ogledali algoritem za reševanje našega problema. Algoritem nato prikažemo še na primeru. Na koncu si bomo ogledali še programsko rešitev, ter naredili še analizo časovne zahtevnosti.

# **Problem**

Kot vhodni podatek dobimo seznam števil dolžine med 1 in 105, števila so iz intervala od -104 do 104. Dobimo tudi podatek *k*, ki predstavlja širino našega drsečega okna. Parameter *k* je iz intervala od 1 do dolžine seznama.

Naš problem izgleda tako: Premikamo se z drsečim oknom po našem sezamu. Na vsakem koraku se premaknemo za eno število naprej in izračunamo kakšen je maksimum trenutnega okenca ter ga shranimo v seznam. Na koncu vrnemo naš novi seznam.

Primer:

[3, 2, 1, 2, 4, 3, 5, 5, 3, 2], 3

V našem primeru je *k* enak 3, torej bo naše drseče okno dolžine 3.

|  |  |
| --- | --- |
| Pozicije drsečega okna | Maksimumi |
| [3 2 1] 2 4 3 5 5 3 2 | 3 |
| 3 [2 1 2] 4 3 5 5 3 2 | 2 |
| 3 2 [1 2 4] 3 5 5 3 2 | 4 |
| 3 2 1 [2 4 3] 5 5 3 2 | 4 |
| 3 2 1 2 [4 3 5] 5 3 2 | 5 |
| 3 2 1 2 4 [3 5 5] 3 2 | 5 |
| 3 2 1 2 4 3 [5 5 3] 2 | 5 |
| 3 2 1 2 4 3 5 [5 3 2] | 5 |

Kot rezultat vrnemo [3, 2, 4, 4, 5, 5, 5, 5].

# **Ideja rešitve**

Da se izognemo nepotrebnemu premetavanju podatkov, bomo uporabili vrsto v kateri se bodo nahajali podatki trenutnega okna. Vrsta je v ta namen zelo primerna, saj ko okno premaknemo v desno, odstranimo element iz začetka vrste in na koncu vrste dodamo nov element. Potrebujemo tudi števec, ki pove, kolikokrat se je v našem trenutnem oknu pojavil maksimum.

Najprej v vrsto prestavimo *k* (širina drsečega okna) podatkov in ob tem določimo njihov maksimum ter kolikokrat je ta maksimum dosežen. Pomožna vrsta predstavlja naše drseče okno. Sedaj vzamemo naslednji element iz začetnih podatkov (torej k+1.), ter pogledamo, kakšna je njegova vrednost. Iz pomožne vrste odstranimo podatek, ter pogledamo, kakšna je njegova vrednost. V primeru, ko je odstranjen podatek maksimum, števec zmanjšamo za 1.

Sedaj nastopi nekaj pogojev:

1. V okno (vrsto) vstopi element večji od maksimuma

|  |  |  |
| --- | --- | --- |
|  | prej | potem |
| okno | [2,6,5,6] | [6,5,6,12] |
| Maksimalen element | 6 | 12 |
| Števec | 2 | 1 |

V tem primeru ni važno kakšen je števec, saj bomo vstavili element, ki je večji od prejšnjega maksimuma, torej je to naš novi maksimum.

1. V okno vstopi element manjši od maksimuma in števec ni enak 0.

|  |  |  |
| --- | --- | --- |
|  | prej | potem |
| okno | [6,4,5,6] | [4,5,6,3] |
| Maksimalen element | 6 | 6 |
| Števec | 2 | 1 |

V tem primeru se števec zmanjša za 1, saj izstopi ena šestica, vendar še zmeraj ni 0. Torej vemo, da je vsaj ena 6 še zmeraj v oknu, torej je 6 še zmeraj maksimum.

1. V okno vstopi element, enak maksimumu

|  |  |  |
| --- | --- | --- |
|  | prej | potem |
| okno | [2,4,5,6] | [4,5,6,6] |
| Maksimalen element | 6 | 6 |
| Števec | 1 | 2 |

V tem primeru se števec poveča za ena, saj smo vstavili še eno 6 in nobene odstranili.

1. V okno vstavimo element manjši od maksimuma in števec je enak 0

|  |  |  |
| --- | --- | --- |
|  | prej | potem |
| okno | [8,4,5,6] | [4,5,6,5] |
| Maksimalen element | 8 | 6 |
| Števec | 1 | 1 |

V tem primeru se števec zmanjša na 0, saj smo 8 odstranili. Vstavili smo 5, ki je manjša od maksimuma. Sedaj ne moremo sklepati kaj je maksimum, zato moramo pregledati celotno okno, element po element.

# **Uporaba na primeru**

Naj bodo naši osnovni podatki ( [3,5,5,4,3,4,9,7,3,15,90,85,90,63,86,50] , 3).

Preko našega seznama se bomo sprehajali z oknom dolžine 3.

|  |  |
| --- | --- |
| 1.korak | podatki |
| okno | [3,5,5] |
| maksimum | 5 |
| števec | 2 |
| rešitev | [5] |

V prvem koraku pregledamo prvih k členov našega seznama in s tem določimo maksimum in števec. Tu primerjamo element po element, kar ni časovno učinkovito.

|  |  |  |
| --- | --- | --- |
| 2.korak | prej | potem |
| okno | [3,5,5] | [5,5,4] |
| izstopi | 3 | /////// |
| vstopi | /////// | 4 |
| maksimum | 5 | 5 |
| števec | 2 | 2 |
| rešitev |  | [5,5] |

Vidimo, da izstopi 3, torej se števec maksimuma ne zmanjša. Ker je števec maksimuma večji od 0 in vstavljamo element manjši od maksimuma, maksimum ostane isti (2. pogoj iz Ideja rešitve).

|  |  |  |
| --- | --- | --- |
| 3.korak | prej | potem |
| okno | [5,5,4] | [5,4,3] |
| izstopi | 5 | /////// |
| vstopi | /////// | 3 |
| maksimum | 5 | 5 |
| števec | 1 | 1 |
| rešitev |  | [5,5,5] |

V tem koraku izstopi 5, zato se tudi števec zmanjša na 1. Števec je še zmeraj večji od 0 torej maksimum ostane isti.

|  |  |  |
| --- | --- | --- |
| 4.korak | prej | potem |
| okno | [5,4,3] | [4,3,4] |
| izstopi | 5 | /////// |
| vstopi | /////// | 4 |
| maksimum | 5 | 4 |
| števec | 0 | 2 |
| rešitev |  | [5,5,5,4] |

V tem koraku nastopi 4. pogoj iz Ideja rešitve. Ker je izstopila 5, je števec padel na 0. Vstopila je tudi 4, ki je manjša od prejšnjega maksimuma. V tem koraku moramo pregledati celotno okno, da dobimo novi maksimum in njegovo število ponovitev.

|  |  |  |
| --- | --- | --- |
| 5.korak | prej | potem |
| okno | [4,3,4] | [3,4,9] |
| izstopi | 4 | /////// |
| vstopi | /////// | 9 |
| maksimum | 4 | 9 |
| števec | 1 | 1 |
| rešitev |  | [5,5,5,4,9] |

V tem koraku nastopi pogoj 1 iz poglavja Ideja rešitve. Vstopi element večji od maksimuma in postane novi maksimum. Hkrati se števec postavi na 1.

Nekaj korakov bomo preskočili, saj se ponavljajo podobni koraki, kot smo jih že obdelali.

|  |  |  |
| --- | --- | --- |
| 10.korak | prej | potem |
| okno | [15,90,85] | [90,85,90] |
| izstopi | 15 | /////// |
| vstopi | /////// | 90 |
| maksimum | 90 | 90 |
| števec | 1 | 2 |
| rešitev |  | [5,5,5,4,9,9,9,15,90,90] |

V tem koraku nastopi pogoj 3 iz poglavja Ideja rešitve. Vstopi element enak maksimumu, torej maksimum ostane isti in števec se poveča.

Nekaj korakov bomo preskočili, saj se ponavljajo podobni koraki kot smo jih že obdelali.

|  |  |  |
| --- | --- | --- |
| 13.korak | prej | potem |
| okno | [90,63,86] | [63,86,50] |
| izstopi | 90 | /////// |
| vstopi | /////// | 50 |
| maksimum | 90 | 86 |
| števec | 0 | 1 |
| rešitev |  | [5,5,5,4,9,9,9,15,90,90,90,90,86] |

Spet smo prišli do situacije, ko je števec 0 in vstavimo element manjši od maksimuma (pogoj 4 iz poglavja Ideja rešitve), torej moramo pregledati vse elemente okna.

Sedaj smo prišli do konca, saj ni več elementa, ki bi lahko vstopil. Torej je naša rešitev [5,5,5,4,9,9,9,15,90,90,90,90,86].

# **Programska rešitev**

Koda ima nekaj komponent.

V tem delu najprej preverimo, ali so podatki veljavni in deklariramo vse potrebne spremenljivke:

Slika, ki vsebuje besede besedilo, posnetek zaslona, pisava

Opis je samodejno ustvarjen

Tu obdelamo prvih k elementov oziroma prvo okno:

Slika, ki vsebuje besede besedilo, posnetek zaslona, pisava

Opis je samodejno ustvarjen

V tem delu preverimo vse 4 pogoje, ki si sledijo tako:

* Vstavimo element manjši od maksimuma in števec ni enak 0
* Vstavimo element enak maksimumu
* Vstavimo element večji od maksimuma
* Vstavimo element manjši od maksimuma in števec je enak 0

Slika, ki vsebuje besede besedilo, posnetek zaslona

Opis je samodejno ustvarjen

# **Analiza časovne zahtevnosti:**

Analiza časovne zahtevnosti je rahlo zapletena, zato bomo obravnavali le časovno zahtevnost v najboljšem ter najslabšem primeru. Naj bo *n* dolžina seznama števil ter *k* širina okna.

Premislek pokaže, da bomo najmanj dela imeli takrat, ko bodo podatki urejeni naraščajoče. V tem primeru imamo najprej *k* primerjav, nato sledi še *n-k* korakov. V okno vstavimo element večji od maksimuma, torej je zmeraj naš novi maksimum element, ki ga vstavimo. Če je naša osnovna operacija primerjava dveh podatkov (katero je večje oziroma manjše) sledi, da imamo *k* primerjav za prvo okno in *n-k* primerjav za drugo, saj moramo še zmeraj preveriti ali je element, ki ga vstavljamo večji od prejšnjega maksimuma. Časovna zahtevnost je torej v najboljšem primeru *O(k+n-k)*, kar je enako *O(n)*.

Premislek ponovno pokaže, da bomo največ dela imeli takrat, ko bodo podatki urejeni padajoče. V tem primeru imamo najprej k primerjav za prvo okno. Nato pa vsakič, ko dodamo nov element odstranimo tudi maksimum. Element, ki smo ga vstavili je manjši od maksimuma, torej ne vemo, kaj je maksimum (primer 4 iz Ideja rešitve). Posledica tega je, da moramo za vsako na novo ustvarjeno okno (ko dodamo in odstranimo element) izračunati kaj je maksimum na enak način kot smo to naredili za prvo okno. Vseh oken je torej *n-k* in za vsako okno imamo *k* primerjav. Za osnovno operacijo bomo spet uporabili primerjavo. Torej bomo imeli *(n-k)\*k* primerjav. V primeru, ko je *k* veliko manjše od *n*, je časovna zahtevnost še zmerjaj *O(n)*, sicer pa je *O(n2)*. Časovno zahtevnost *O(n2)* dobimo tako:

Predpostavimo, da je *k=n/3*, torej če uporabimo formulo *(n-k)\*k* dobimo:

Ko pogledamo časovno zahtevnost dobimo *O(n2)*.

# **Viri**

[1] LeetCode, (17.11.2023), Sliding Window Maximum(239)

https://leetcode.com/problems/sliding-window-maximum/

­­­