

**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. V prvem razdelku je predstavljen prikaz vhodnih podatkov, ter s pomočjo primera rešen naš problem. V drugem razdelku je razložena ideja rešitve, v tretjem razdelku pa je ideja rešitve implementirana na primeru. Četrto razdelku je namenjeno predstavitvi programerske implementacije algoritma. V petem razdelku je analiza 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.

Zanimajo nas maksimumi vseh možnih strnjenih podzaporedij dolžine *k*. Rezultat lahko torej izračunamo na sledeč način. Na začetek seznama podatkov postavimo drseče okno dolžine *k*. Na vsakem koraku izračunamo, kakšen je maksimum podatkov v trenutnem okencu ter ga shranimo v seznam maksimumov. Okno premaknemo za eno število naprej 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.

|  |  |
| --- | --- |
| Položaji 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 (najbolj levega v oknu) in na koncu vrste dodamo nov element. Potrebujemo tudi števec, ki pove, kolikokrat se je v našem trenutnem oknu pojavil maksimum. Ta števec bomo potrebovali za učinkovitejše določanje maksimuma v oknu.

Najprej v vrsto prestavimo *k* (širina drsečega okna) podatkov in ob tem določimo njihov maksimum ter kolikokrat je ta maksimum dosežen. Ta vrsta predstavlja naše drseče okno, ki se začne z elementom Tab[0] in konča s Tab[k-1]. Poglejmo kaj se zgodi, ko okno premaknemo za eno mesto v desno. Na levi strani izpade en element, torej v vrsti odstranimo en element iz začetka. Če je ta element enak maksimalnemu, števec zmanjšamo za ena. Okno tudi pridobi nov element- Zato smo vstavili element na konec vrste. Glede na element, ki smo ga odstranili in elementa ki smo ga dodali v vrsto, nastopi nekaj različnih možnosti:

Ogledali si bomo vse možne situacije, ki nastopijo, njihovo delovanje si bomo ogledali na med seboj nepovezanih zgledih. Kot smo že omenili, je lahko velikost okna poljubna. Tu bomo uporabili okno dolžine 4 (k=4). Oglejmo si vse možnosti, ki lahko nastopijo.

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

|  |  |  |
| --- | --- | --- |
|  | prej | potem |
| Okno | [2,6,5,6] | [6,5,6,12] |
| Trenutni maksimalni element | 6 | 12 |
| Števec maksimalnih | 2 | 1 |

Dobili smo element, ki je večji od prejšnjega maksimuma, torej je to naš novi maksimum. Števec ponovitev maksimalnega elementa nastavimo na 1.

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

|  |  |  |
| --- | --- | --- |
|  | prej | potem |
| Okno | [6,4,5,6] | [4,5,6,3] |
| Trenutni maksimalni 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] |
| Trenutni maksimalni 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] |
| Trenutni maksimalni element | 8 | 6 |
| Števec | 1 | 1 |

V tem primeru se števec zmanjša na 0, saj smo 8 odstranili. Vstavili smo 5, torej manjši podatek od trenutnega maksimuma. Ker ne moremo vedeti, kakšen je maksimum v oknu, moramo pregledati celotno okno, element po element.

Ne glede na to, kaj se je zgodilo, trenutni maksimalni element dodamo k rezultatu.

# 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.

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

Iz okna odstranimo 3 (ki ni maksimalni element), torej se števec maksimuma ne zmanjša. Ker je števec maksimuma večji od 0 in vstavljamo element, manjši od maksimuma, maksimum ostane enak (drugi 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 enak.

|  |  |  |
| --- | --- | --- |
| 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] |

Na tem koraku je stanje kot ga opisuje pogoj 1 iz razdelka Ideja rešitve. V okno dodamo 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 razdelka Ideja rešitve. Vstopi element, ki je enak maksimumu. , Zato maksimum ostane enak in števec maksimalnih 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] |

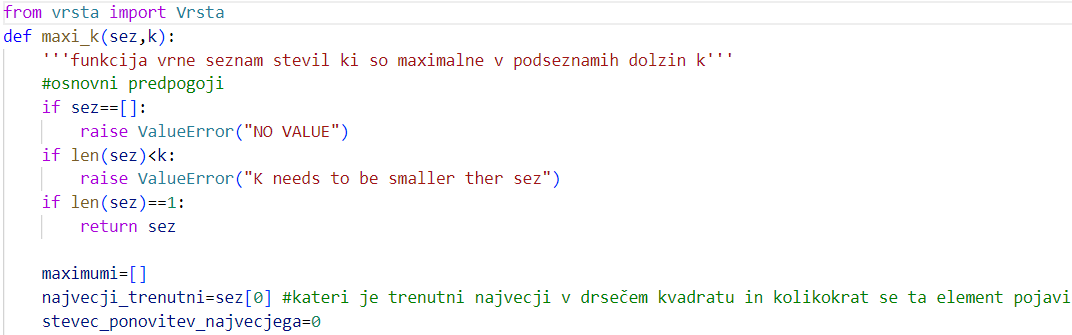
Spet smo prišli do stanja, ko je števec 0 in vstavimo element manjši od maksimuma (pogoj 4 iz razdelka Ideja rešitve), torej moramo pregledati vse elemente okna.

Sedaj smo prišli do konca, saj smo pregledali celotni seznam podatkov. Torej je naša rešitev [5,5,5,4,9,9,9,15,90,90,90,90,86].

# **Programska rešitev**

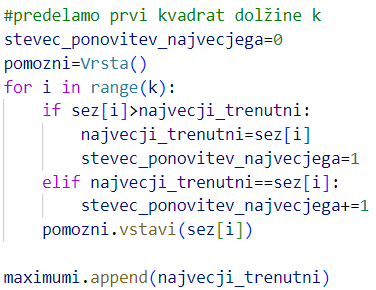
V tem razdelku si bomo ogledali posamezne bolj zanimive dele kode.

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



Slika 1 Osnovni pogoji

Tu obdelamo prvih k elementov oziroma prvo okno:



Slika 2 Obdelava prvega okna

V tem delu preverimo vse štiri pogoje:

* 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 3 Pogoji

# **Analiza časovne zahtevnosti:**

Analiza časovne zahtevnosti je rahlo zapletena. Problem je v tem, da za splošne podatke ne moremo vedeti, kolikokrat bomo lahko maksimum določali le s primerjavo odhajajočega, prihajajočega in trenutnega maksimuma in kolikokrat morali pregledati vse podatke v oknu. Zato si bomo ogledali le najboljši primer in najslabši primeru.

bomo maksimum določili vsakič s pregledom celotnega okna. Naj bo *n* dolžina seznama števil ter *k* širina okna.

Premislek pokaže, da bomo najmanj dela imeli takrat, ko bodo podatki urejeni naraščajoče. . Takrat maksimum določamo s pregledom vseh elementov okna le na začetku Najprej torej opravimo *k* primerjav, nato sledi še *n-k* korakov. Na vsakem koraku opravimpo le dve primerjavi - eno, da vidimo, da je prihajajoči element večji od trenutnega maksimuma in drugo, da ugotovimo, da je novi trenutni maksimum večji od doslej navečjega kandidata. Časovna zahtevnost je torej v najboljšem primeru *O(n)*.

Premislek ponovno pokaže, da bomo največ dela imeli takrat, ko bodo podatki urejeni padajoče. V tem primeru imamo najprej *k* primerjav, da določimu maksimum za prvo okno. Nato vsakič, ko dodamo nov element, odstranimo tudi maksimum. Element, ki ga vstavljamo, 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. Vseh oken je torej *n-k* in za vsako okno imamo *k* primerjav. Opravimo torej *(n-k)\*k* primerjav. V primeru, ko je *k* veliko manjši od *n (ali pa skoraj enak n!!)*, je časovna zahtevnost *O(n)*, sicer pa je *O(n2)*.

# **Viri**

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

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

­­­