Urša Kumelj

OSNUTEK

# **Povzetek \*\*\* Končno – priporočam, da je ta razdelek VEDNO že končen\*\*\***

*V prispevku so predstavljeni trije problemi in sicer poiskati največjo vsoto podseznama, največji zmnožek podseznama in koliko časa bi potrebovali za izračun najmanjše poti med dvema ogljiščema v zazankanem drevesu. Najprej si bomo ogledali način predstavitve vhodnih podatkov problema ter s pomočjo primera predstavili, zakaj pri posameznem problemu točno gre. Posamezen algoritem prikažemo še na primeru. Prispevek prvih dveh problemov zaključimo s prikazom kode rešitve.*

# **Problem največje vsote in največjega produkta podseznama**

Tukaj bom napisala navodilo naloge in od kod je naloga.

# **Največja vsota podseznama**

## **Naivni način**

Dodala bom sliko in opisala kako pristopimo k problemu na prvo žogo.

## **Kadanov algoritem\*\*\*končno\*\*\***

Ideja je imeti dve spremenljivki. Ena, ki hrani največjo vsoto podseznama do trenutnega indeksa (recimo ji trenutnaVsota) in drugo, ki hrani največjo vsoto podseznamov najdenih do sedaj (recimo ji maxVsota). Torej trenutnaVsota opredelimo kot maksimalno vsoto tistih podzaporedij, ki se končajo na indeksu, kjer smo. Ko smo na koraku i, vemo kakšna je trenutnaVsota podzaporedij, ki se končajo na (i-1) mestu. Ko pa gledamo i-ti element, novo tranutnaVsota naračunamo tako, da:

* Če je element pozitiven, bo trenutnaVsota stara povečana za element na i-tem mestu.
* Če je element negativen, pa imamo dve možnosti:
  + Element se nam morda splača vzeti, ker bo služil kot 'most' do nadaljnjih novih maksimalnih trenutnihVsot. To se splača vzeti, če stara trenutnaVsota s to prišteto negativno vrednostjo ne pade pod 0.
  + Če povečana vsota pade pod 0, pa se nam elementa ne splača vzeti. Bolje je vzeti podzaporedje na (i+1) mestu, zato trenutnaVsota nastavimo na 0.

Hkrati preverjamo tudi največjo vsoto do sedaj (maxVsoto), ki je lahko sedaj trenutnaVsota ali pa ostane maxVsota od prej.

Razumevanje zgoraj opisanega si lahko morda pomagamo s spodnjo skico.

A whiteboard with writing on it

Description automatically generated

## **Uporaba na primeru\*\*\*končno\*\*\***

Recimo, da imamo dan seznam [-3,10,-9,80,-75,3,-10,20,41,3,-7,81,203,17,-306,-33].

Rešitev bomo pokazali na določenih indeksih, držali pa se bomo poimenovanj opisanih zgoraj.

Na začetku nastavimo trenutnaVsota in maxVsota na 0.

* i = 0:
  + - * trenutnaVsota = max(-3, 0 + (-3)) = -3
      * maxVsota = max(0, -3) = 0
* i = 1:
  + - * trenutnaVsota = max(10, 10 + (-3)) = 10
      * maxVsota = max(0, 10) = 10
* i = 6:
  + - * trenutnaVsota = max(-10, -10 + 9) = -1
      * maxVsota = max(81, -1) = 81
* i = 11:
  + - * trenutnaVsota = max(81, 81 + 57) = 138
      * maxVsota = max(81, 138) = 138
* i = 12:
  + - * trenutnaVsota = max(203, 203 + 138) = 341
      * maxVsota = max(138, 341) = 341
* i = 13:
  + - * trenutnaVsota = max(17, 17 + 341) = 358
      * maxVsota = max(341, 358) = 358

Kar je z zeleno obrvano prikazuje enako število. Enako velja za modor, rumeno in vijolično obravna števila.

Ko postopek nadaljujemo do konca, dobimo, da je največja vsota enaka 358, njej pripadajoč podseznam pa je [20,41,3,-7,81,203,17].

## **Analiza časovne in prostorske zahtevnosti Kadanovega algoritma\*\*\*končno\*\*\***

* Velikost problema: velikost seznama
* Karakteristična operacija: seštevanje in primerjanje velikosti
* Časovna zahtevnost: , ker iteriramo samo enkrat čez seznam
* Prostorska zahtevnost: , ker uporabljamo samo dve spremenljivki in nič dodatnega prostora npr. dodatnega seznama.

## **Programska rešitev Kadanovega algoritma**

Dodala in morda še kaj pokomentirala bom kodo.

# **Največjegi produkt podseznama**

## **Naivni način**

Podobno bom objasnila pristop k temu problemu na prvo žogo kot pri zgornjem primeru.

## **Ideja algoritma\*\*\*končno\*\*\***

Na mestu i imamo od prej poračunan največji in najmanjši produkt podseznama. Sedaj imamo dve možnosti za nov največji produkt. Lahko je to vrednost na mestu i ali pa je vrednost prejšnjega podseznama z največjim zmnožkom pomnoženega, z vrednostjo na mestu i. To označujemo z lokalni\_max. Enak razmislek je za najmanjšo vrednost produkta podseznama, ki ga označujemo z lokalni\_min. Pri seštevanju nimamo 'problema' s predznaki števila, tukaj pa lahko naletimo na tri možnosti, v primeru, da je število na mestu i negativno in sicer:

* Naj bosta lokalni\_max > 0 in lokalni\_min < 0. Ko ju pomnožimo z vrednostjo na mestu i, bo lokalni\_max \* i < 0 in lokalni\_min \* i > 0. Sedaj bo nov lokalni\_min večji od novega lokalnega\_max.
* Naj bosta lokalni\_max > 0 in lokalni\_min > 0. Ko ju pomnožimo z vrednostjo na mestu i, bo lokalni\_max \* i < 0 in lokalni\_min \* i < 0. Sedaj bo nov lokalni\_min večji od novega lokalnega\_max.
* Naj bosta lokalni\_max < 0 in lokalni\_min < 0. Ko ju pomnožimo z vrednostjo na mestu i, bo lokalni\_max \* i > 0 in lokalni\_min \* i > 0. Sedaj bo nov lokalni\_min večji od novega lokalnega\_max.

Od tod ideja, da ko naletimo na negativno vrednost, zamenjamo lokalni\_min in lokalni\_max.

Razumevanje zgoraj opisanega si lahko morda pomagamo s spodnjo skico.

A whiteboard with text and numbers

Description automatically generated

## **Uporaba na primeru**

Na podoben način bom dodala primer kot pri največjem produktu.

## **Analiza časovne in prostorske zahtevnosti algoritma\*\*\*končno\*\*\***

* Velikost problema: velikost seznama
* Karakteristična operacija: množenje in primerjanje velikosti
* Časovna zahtevnost: , ker iteriramo samo enkrat čez seznam
* Prostorska zahtevnost: , ker uporabljamo samo tri spremenljivki in nič dodatnega prostora npr. dodatnega seznama.

## **Programska rešitev algoritma**

Dodala in morda še kaj pokomentirala bom kodo.

# **Problem najkrajše poti med dvema vozliščema v zazankanem drevesu (angl. Looped tree)**

V knjigi (Erickson, 2019) na strani 297, 1. naloga, je predstavljen problem najkrajše poti med dvema vozliščema v zazankanem drevesu. Poglejmo, zakaj gre.

*Zazankano drevo je utežen, usmerjen graf, zgrajen iz binarnega drevesa z dodajanjem povezave od vsakega lista nazaj do korena. Vsaka povezava ima nenegativno težo.*

*A diagram of a tree

Description automatically generated*

1. *Koliko časa bi Dijkstrov algoritem potreboval za izračun najkrajše poti med dvema vozliščema u in v v zankanem drevesu z n vozlišči?*
2. *Opišite in analizirajte hitrejši algoritem.*

## **Dijkstrov algoritem**

Opisala bom idejo Dijkstrovega algoritma.

## **Uporaba Dijsktrovega algoritma na primeru**

Na primeru bom obrazložila Dijkstrov algoritem, če na začetku podamo začetno vozlišče u in končno vozlišče v.

## **Psevdokoda Dijsktrovega algoritma**

Dodala bom psevdokoto algoritma.

## **Analiza časovne in prostorske zahtevnosti Dijsktrovega algoritma**

Zapisala bom časovne zahtevnosti v odvisnosti od podatkovne strukture izbrane za Q.

## **Analiza časovne zahtevnosti na zazankanem drevesu z uporabo Dijkstrovega algoritma\*\*\*končno\*\*\***

Opazimo lahko, da ima vsako vozlišče največ dve povezavi, ki gresta iz njega. Torej v najslabšem primeru, če gremo skozi vse povezave, gremo dvakrat skozi vsa vozlišča. Časovna zahtevnost za povezave je , saj je . Skupna časovna zahtevnost za Dijakstrov algoritem je .

## **Analiza časovne zahtevnosti in opis boljšega algoritma na zazankanem drevesu\*\*\*končno\*\*\***

Če pogledamo sliko lahko ugotovimo, da obstaja samo eno vozlišče, ki ima več kot eno povezavo, ki kaže vanj. To je koren. Najkrajša pot med vozliščema in bo lahko šla čez koren ali pa ne. Poglejmo si oba primera.

1. Vozlišči in **ne gresta čez koren**. Rešitev je tako samo ena in sicer seštevek cen povezav od do . To lahko opravimo v z uporabo DFS (Depth-first search) algoritma oz. algoritma iskanja v globino.

A diagram of a tree

Description automatically generated

1. Vozlišči in **gresta čez koren**. Od korena do vozlišča bo pot enolična in jo lahko naračunamo enako kot v primeru a). Potrebujemo izračunati le najkrajšo pot od do korena. Od do korena pa imamo usmerjeni acikličen graf, torej graf, ki je usmerjen in brez ciklov. Zanj prav tako obstaja algoritem za računanje najkrajših poti (angl. Directed Acyclic Graph), ki ima časovno zahtevnost Skupen čas za algoritem je torej

A diagram of a tree

Description automatically generated

## **Prikaz boljšega algoritma na nekaj primerih**

Dodala bom slike različnih primerov za izbran u in v na malo večjem zazankanem drevesu.

# Viri

Erickson, J. (2019). *Algorithms.*

*Maximum subarray problem*. (2023). Pridobljeno iz Wikipedia: https://en.wikipedia.org/wiki/Maximum\_subarray\_problem

*Kadanes algorithm explained*. (2023). Pridobljeno iz Algodaily: https://algodaily.com/lessons/kadanes-algorithm-explained

*Largest sum contiguous subarray*. (2023). Pridobljeno iz Geeksforgeeks: https://www.geeksforgeeks.org/largest-sum-contiguous-subarray/

Abhishek, S. w. (8. June 2021). *Max Product Contiguous Subarray*. Pridobljeno iz You Tube: https://www.youtube.com/watch?v=bLHrFVx-OJQ

*Maximum product subarray in an array*. (2023). Pridobljeno iz TakeUForward: https://takeuforward.org/data-structure/maximum-product-subarray-in-an-array/

*Dijkstra time complexity*. (2023). Pridobljeno iz Baeldung: https://www.baeldung.com/cs/dijkstra-time-complexity

*Priority queue*. (2023). Pridobljeno iz Baeldung: https://www.baeldung.com/cs/priority-queue

*Shortest path for directed acyclic graphs*. (2023). Pridobljeno iz Geeksforgeeks: https://www.geeksforgeeks.org/shortest-path-for-directed-acyclic-graphs/

*H21 solution*. (2023). Pridobljeno iz Bowdoin: https://tildesites.bowdoin.edu/~ltoma/teaching/cs231/duke\_cps130/Homeworks/Solutions/H21-solution.pdf

Person, C. t. (15. january 2022). *Dijkstra's Algorithm Visualized and Explained*. Pridobljeno iz You Tube: https://www.youtube.com/watch?app=desktop&v=71Z-Jpnm3D4