Ređanje novčića na polici

Opis algoritma i implementacije u okviru kursa
Geometrijski algoritmi
Matematički fakultet

Tatjana Kunić
tanja.kunic@gmail.com

31. januar 2023.

Sažetak

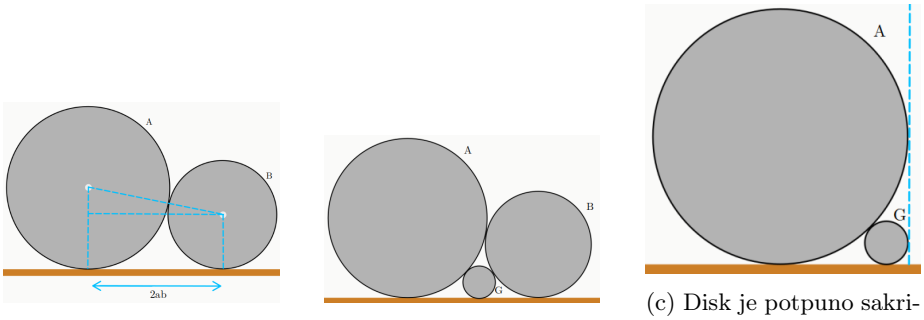
U ovom tekstu je opisan problem ređanja novčiča na polici (eng. *Placing your coins on a shelf*), naivni pristup kao i pohlepni algoritam za rešavanje problema i njegova implementacija.

Sadržaj

1 Opis problema

Problem ređanja novčića na polici ubraja se u probleme pakovanja (eng. *packing problems*) - klasu optimizacionih problema u kojoj su mnogi problemi NP - teški. Problem se može definisati na sledeći način: Potrebno je poređati novčiće različitih veličina na policu tako da je dužina rasporeda minimalna. Novčiće predstavljamo kao diskove, dok polica predstavlja horizontalnu x-osu. Veličina diska definiše se kao koren njegovog poluprečnika. Razlikujemo tri odnosa između diskova 1:

- Diskovi se dodiruju 1a - donje rastojanje (eng. *footpoint distance*) između dva diska koja se dodiruju gde su a i b redom veličine diskova A i B jednako je $2ab$
- Disk staje u prazan prostor između dva diska 1b - veličina najvećeg takvog diska G može da se izračuna kao $1/g = 1/a + 1/b$
- Disk G je potpuno sakriven ispod diska A 1c - njegova veličina jednaka je barem $g = (\sqrt{2} - 1) \cdot a$



(a) Donje rastojanje (b) Disk staje u prazninu (c) Disk je potpuno sakriven ispod drugog

Slika 1: Odnosi diskova

2 Naivni algoritam

Najjednostavniji pristup rešavanju problema jeste da se isprobaju svi mogući rasporedi i da se na kraju kao rešenje uzme onaj čija je dužina najmanja. Što se tiče implementacije naivnog algoritma, za određivanje sledeće permutacije niza diskova može da se koristi funkcija `std::next_permutation` . Neophodno je da niz prethodno bude sortiran rastuće pre korišćenja ove funkcije. S obzirom na to da se ispituje svaka permutacija niza diskova i pritom računa dužina rasporeda svakog niza, složenost naivnog pristupa će biti $O(n \cdot n!)$. Jasno je da ovaj pristup brzo postaje izuzetno neefikasan, ali s druge strane, on će uvek dati tačno rešenje za razliku od pohlepnog pristupa koji će sledeći biti opisan.

3 Pohlepni algoritam sa faktorom aproksimacije $4/3$

Diskovi koji su dovoljno mali da stanu u prazan prostor između dva diska koja se dodiruju neće uticati na ukupnu dužinu konačnog rasporeda. Upravo je ovo jedna od glavnih ideja pohlepnog algoritma za ređanje

novčića na polici - uvek se prvo proverava da li je disk dovoljno mali da stane u praznine između diskova.

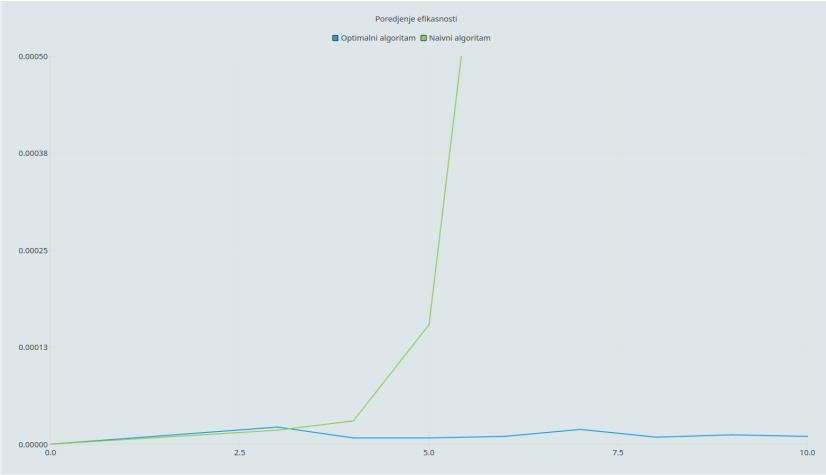
Algorithm 1: Pohlepni algoritam za ređanje novčića na polici

Ulaz: Niz diskova sortirani nerastuće po veličini
Izlaz: Raspored diskova koji je najviše 4/3 optimalnog rešenja
za *svaki disk D veličine d iz sortiranog niza* **radi**
  ako $d \leq \frac{1}{1/a+1/b}$ *gde su a i b redom veličine diskova A i B koji su već na polici* **onda**
    postavi D u prazninu između A i B
  u suprotnom
    Neka je A disk koji je trenutno na levom kraju niza diskova koji su već na polici, a neka je Z disk na desnom kraju. Neka je onda d_A dužina rasporeda i D_A kandidat pozicija za disk D ukoliko bi se on postavio ispred diska A. Slično, D_Z je kandidat pozicija ukoliko bi se disk D postavio iza diska Z i d_Z odgovarajuća dužina rasporeda
    ako $d_A < d_Z$ **onda**
      postavi D na poziciju D_A
    u suprotnom
      postavi D na poziciju D_Z
    kraj
  kraj
kraj

Složenost pohlepnog algoritma će zavisiti samo od složenosti sortiranja, jer prolazimo tačno jedanput kroz sortiran niz diskova. Dakle, složenost ovog pristupa će biti $O(nlogn)$. Iako je ovaj algoritam značajno efikasniji od naivnog, on ne daje uvek tačno (optimalno) rešenje. Algoritam garantuje da će rešenje biti najviše 4/3 puta veće od optimalnog.

4 Implementacija pohlepnog algoritma

Klasa `Disk` sadrži sve neophodne informacije o jednom disku - veličina, poluprečnik kao i metod `isOvershadowedByOther(const Disk&)` kojim se proverava da li je potpuno sakriven ispod drugog diska. Klasom `Gap` opisan je prazan prostor između dva diska. Kako se diskovi postavljaju na policu tako se dodaje nova praznina u red sa prioritetima gde su praznine sortirane po veličini opadajuće. Ovim se garantuje da ukoliko trenutni disk ne staje u prazninu sa najvećim prioritetom, odmah se prelazi na sledeći korak algoritma. S obzirom na to da je neophodno imati informaciju o diskovima sa levog i desnog kraja rasporeda na polici, dobra ideja je da se konačan raspored čuva u redu sa dva kraja (`std::deque`)



Slika 2: Vreme ivršavanja algoritama u zavisnosti od veličine ulaza

5 Zaključak

Problem ređanja novčića na polici je NP-težak i ne postoji efikasan algoritam koji bi uvek vraćao optimalno rešenje. Oba algoritma obrađena u ovom tekstu imaju svoje prednosti i mane. Kao što je već spomenuto, naivni algoritam će brzo postati izuzetno neefikasan iako uvek daje tačno rešenje, dok je pohlepni efikasan ali ne garantuje da će naći optimalno rešenje. Na slici 2 je grafik poređenja vremena izvršavanja ova dva pristupa u zavisnosti od veličine ulaza.