# Problem stabilnih cimera

Konstrukcija i analiza algoritama 2
Matematički fakultet

Lazar Stanojević
lazar01.stanojevic@gmail.com

3. maj 2024.

**Sažetak**

U radu će biti predstavljen opis problema stabilnih cimera, sličnog problemu stabilnog braka, kao i algoritam za njegovo rešavanje. Algoritam je preuzet iz časopisa Journal of Algorithms, iz rada autora Robert W. Irving [1]. Ideja je poslužila za demonstraciju jedne moguće varijante implementacije ovog algoritma u programskom jeziku C++.

## Sadržaj

# 1 Uvod

Problem stabilnih cimera pripada familiji problema uparivanja. Pretpostavimo da imamo skup od $n$ osoba, parne kardinalnosti. Svaka osoba iz skupa, rangira ostale iz skupa, prema svojim preferencijama. Cilj je da nađemo stabilno uparivanje, koje predstavlja particionisanje skupa na $\frac{n}{2}$ parova cimera, tako da ne postoje 2 osobe koje nisu cimeri, a preferiraju jedan drugog više u odnosu na svoje trenutne partnere, u skladu sa poretkom iz listi preferencija.

Srodan ovome je i problem stabilnog braka, ali nasuprot njemu, pokazuje se da postoje instance ovog problema za koje odgovarajuće uparivanje ne postoji, što može da se vidi i na slici 1a. Bilo koja osoba uparena sa osobom 4 će izazvati nestabilnost. Takođe, postoje i instance za koje postoji i više mogućih uparivanja. Primer jedne takve dat je na slici 1b. Mogućih rešenja za ovaj ulaz ima tačno 3, i to su sledeća uparivanja: [1-2, 3-4, 5-8, 6-7], [1-4, 2-3, 5-6, 7-8] i [1-5, 2-6, 3-7, 4-8].
Dalje u radu, prikazaćemo algoritam za ovaj problem, koji radi u polinomijalnom vremenu. Osvrnućemo se i na složenost algoritma, kao i na način na koji može da se realizuje u jeziku C++.

| Osoba | Lista preferencije |
|---|---|
| 1 | 2 3 4 |
| 2 | 3 1 4 |
| 3 | 1 2 4 |
| 4 | bilo šta |

(a) Nepostojeće uparivanje

| | |
|---|---|
| 1 | 2 5 4 6 7 8 3 |
| 2 | 3 6 1 7 8 5 4 |
| 3 | 4 7 2 8 5 6 1 |
| 4 | 1 8 3 5 6 7 2 |
| 5 | 6 1 8 2 3 4 7 |
| 6 | 7 2 5 3 4 1 8 |
| 7 | 8 3 6 4 1 2 5 |
| 8 | 5 4 7 1 2 3 6 |

(b) Postojanje više rešenja

Slika 1: Primeri instanci

# 2 Algoritam stabilnih cimera

Algoritam se sastoji od dve faze, pri čemu obe podrazumevaju redukciju listi preferencija, dok neka ne postane prazna, što implicira da traženo uparivanje ne postoji, ili dok kardinalnost svake ne postane 1, u tom slučaju je uparivanje pronađeno, i svakoj osobi je dodeljena preostala osoba u njenoj listi. Osobe ćemo označavati brojevima od 1 do $n$, dok će liste preferencija biti nizovi, veličine $n-1$, pri čemu za osobu $i$ lista predstavlja jednu permutaciju skupa $\{1 \dots n\} \setminus \{i\}$.

## 2.1 Prva faza

U prvoj fazi, krećemo se redom kroz listu osoba, i svaka od njih predlaže sledećoj osobi u svojoj listi preferencija da budu cimeri. Označimo trenutnu osobu sa $x$, a sa $y$ osobu iz njene liste preferencija. Ukoliko $y$ ima već predlog od neke osobe koju je rangirala više u svojoj listi preferencija, onda odbija $x$, i primorava je da nastavi sa predlozima, tamo gde je stala u svojoj listi preferencija, odnosno odmah nakon $y$. U suprotnom, $y$ nema bolji predlog od $x$, prihvata njen predlog na razmatranje, i omogućava prelazak na osobu $x+1$, koja počinje da šalje predloge osobama iz svoje liste.
Ova faza algoritma se završava ako svaka osoba ima predlog koji čuva, ili ukoliko je neka osoba odbijena od strane svih drugih osoba.
Dalje u tekstu navodimo neka tvrđenja, bez dokaza, koja obezbeđuju ispravnost ove faze algoritma.

**Lema 1.** *Ako $y$ odbije $x$ u nekom trenutku, onda $x$ i $y$ ne mogu da budu par u stabilnom uparivanju.*

**Posledica 1.1.** *Ako u nekom trenutku, $x$ predloži $y$ da budu cimeri, onda, u stabilnom uparivanju, $x$ ne može imati boljeg partnera od $y$, a $y$ ne može imati goreg od $x$, u kontekstu listi preferencija.*

**Posledica 1.2.** *Ako je jedna osoba odbijena od strane svih, onda ne postoji stabilno uparivanje.*

**Posledica 1.3.** *Ako se prva faza završi tako što svaka osoba čuva predlog, onda listu preferencija osobe $y$, koja čuva predlog od $x$, možemo da redukujemo tako što brišemo sve one koji su rangirani gore od $x$, kao i one koji čuvaju predlog bolje rangirane osobe od $y$, u njihovoj listi preferencija.*

**Lema 2.** *Ako u redukovanoj listi preferencija, svaka lista sadrži tačno 1 osobu, onda je to stabilno uparivanje.*

Dakle, u prvoj fazi izvodimo cikluse slanja predloga, nakon čega, na osnovu Posledice 1.3, redukujemo liste preferencija. Ukoliko svaka lista sadrži tačno jedan element, našli smo stabilno uparivanje, u suprotnom prelazimo na drugu fazu algoritma.

## 2.2 Druga faza

U drugoj fazi treba nekako da se pozabavimo redukovanim listama preferencija, od kojih neke liste imaju više od jednog elementa. U ovoj fazi ponovo redukujemo liste, ali sada malo na drugačiji način, pri čemu kriterijum zaustavljanja ostaje isti, dok nekoj osobi ne ponestane osoba kojima može da predloži da budu cimeri, ili su sve liste redukovane na tačno jednu osobu.
Ključna stvar u drugoj fazi je prepoznavanje ciklusa $a_1, a_2, ..., a_r$, različitih

osoba, takvih da je druga osoba u trenutno redukovanoj listi preferencija osobe $a_i$, prva u listi preferencija osobe $a_{i+1}$, pri čemu $i+1$ gledamo po modulu $r$. Ovakav ciklus zovemo ***sve li ništa ciklus***.
Ispostavlja se da je lako naći jedan ovakav ciklus. Označićemo sa $p_1$ proizvoljnu osobu, čija lista preferencija ima više od jednog elementa. Koristimo niz pomoćnih promenljivih, $q_i$, pri čemu je $q_i$ druga osoba u listi preferencija od $p_i$. Na osnovu $q_i$ generišemo $p_{i+1}$, uzimajući poslednji element iz liste osobe $q_i$. Ponavljamo postupak dok niz $p$ ne napravi ciklus, odnosno dok se neka osoba ne javi dva puta. Označimo sa $p_s$ osobu koja se prva ponovila u sekvenci $p$. Niz $p_1, p_2, ..., p_s$ zovemo rep ciklusa, dok sve ili ništa ciklus dobijamo tako što $a$ uzima vrednosti na sledeći način, $a_i = p_{s+i-1}$.
Faza dva, primenjena na skup redukovanih listi preferencija, za određeni sve ili ništa ciklus $a_1, ..., a_r$, podrazumeva primoravanje svakog $b_i, 1 \leq i \leq r$ da odbije predlog koji čuva od $a_i$, i samim tim tera $a_i$ da pošalje predlog osobi $b_{i+1}$, drugoj osobi u njegovoj trenutnoj, redukovanoj listi preferencija. Samim tim, svi sukcesori osobe $a_i$, u redukovanoj listi preferencija osobe $b_{i+1}$ mogu biti obrisani, a takođe i $b_{i+1}$ može biti uklonjen iz njihove liste preferecija.
Značaj redukovnih listi preferencija leži u tome da, ukoliko početna instanca problema sadrži stabilno uparivanje, onda postoji i stabilno uparivanje u kojem su sve osobe uparene sa nekom iz njihove redukovane liste preferencija. Za ovakvo uparivanje kažemo da je sadržano u redukovanim listama preferencija, i ovo je posledica sledeće leme.

**Lema 3.** *Neka je $a_1, a_2, ..., a_r$ sve ili ništa ciklus, i neka je $b_i$ prva osoba u redukovanoj listi preferencija osobe $a_i$, onda:*

*1) U bilo kom uparivanju sadržanom u redukovanim listama preferencija, ili su $a_i$ i $b_i$ partneri za sve vrednosti i, ili ni za jedno.*

*2) Ako postoji stabilno uparivanje u kojem su $a_i$ i $b_i$ partneri, onda postoji još jedno u kojem nisu.*

**Posledica 3.1.** *Ako postoji stabilno uparivanje u početnoj instanci problema, onda postoji i stabilno uparivanje sadržano u bilo kojim redukovanim listama preferencija.*

**Posledica 3.2.** *Ako je bilo koja od redukovanih listi preferencija prazna, onda ne postoji stabilno uparivanje.*

I poslednja i finalna lema kaže:

**Lema 4.** *Ukoliko svaka redukovana lista preferencije sadrži tačno jednu osobu, onda postoji stabilno uparivanje za tu instancu problema.*

Faza dva redukcije se sprovodi sve dok postoji lista preferencije koja ima više od jednog elementa, ili dok se neka ne isprazni, što znači da ćemo potencijalno više puta tražiti sve ili ništa cikluse i vršiti redukcije listi preferencija.

# 3 Implementacija

Algoritam je implementiran u programskom jeziku C++, a kako bi korišćenje bilo olakšano, integrisan je u aplikaciju napravljenu pomoću QtCreator-a. U ovom poglavlju biće opisane samo najbitnije funkcije za rad algoritma, dok će opisivanje integracije algoritma u Qt aplikaciju biti zanemareno.

Za opisivanje matrice preferencija, koju zovemo *preferences* koristimo ugrađenu strukturu podataka jezika C++, vector, tačnije vektor vektora. Na samom početku, tokom unosa instance problema, odnosno matrice preferencija, formiramo i matricu *rankings*, koja predstavlja rankove osoba u listama preferencija. Ovo nam olakšava posao, jer će nam rankovi biti potrebni, a njihovi računanje tokom algoritma bi oduzimalo bespotrebno vreme.

**do_proposals** - Ovo je pomoćna funkcija koja služi za realizaciju prve faze algoritma, kako bi kod bio čitljiviji. Argumenti funkcije su *preferences, rankings, favorites, proposals_pos, roommate, n.* Vektor *favorites* pamti za svaku osobu njenog trenutnog favorita, odnosno najbolje rangiranu osobu u njenoj listi preferencija, od koje je dobila predlog za uparivanje. Vektor *proposals_pos* čuva poziciju u listi preferencija do koje se stiglo sa slanjem predloga, za svaku osobu. *Roommate* označava osobu za koju se vrši slanje predloga do prihvatanja, a *n* dimenziju problema. Funkcija ide redom kroz listu preferencija osobe *roommate*, i šalje predloge za uparivanje, sve dok neko ne prihvati, a to bi značilo da *roommate* ima bolji rank u listi preferencija osobe koja prihvata predlog, od njenog trenutnog favorita. Samim tim, favorit se ažurira, a za bivšeg favorita se rekurzivno poziva funkcija. Ukoliko se stiglo do kraja liste preferencija neke osobe, znači da je ona odbijena od strane svih ostalih, i algoritam nema stabilno uparivanje, pa se prekida izvršavanje.

**stage_one** - Funkcija realizuje prvu fazu redukcija listi preferencija, pozivajući *do_proposals* za svaku osobu, kako bi odredila favorite. Nakon toga, na osnovu favorita vrši redukciju listi preferencija na način opisan ranije u tekstu. Argumenti funkcije su isti kao za prethodnu, osim što ne prima određenu osobu.

**phase_two_condition** - Proverava da li je neki od uslova za zaustavljanje druge faze algoritma zadovoljen. Takođe, i ovi uslovi za navedeni ranije u tekstu.

**find_all_or_nothing_cycle** - Nalazi traženi ciklus, prema opisanoj proceduri, tako što početni element bira na slučajan način, i smešta ga u vektor *cycle*, a vraća veličinu repa kao povratnu vrednost. Ukoliko funkcija mora biti pozvana više puta, odnosno izvršava se više redukcija druge faze, onda za početnu osobu se bira poslednja osoba iz repa prethodnog ciklusa, zbog toga što njene liste preferencija nisu obuhvaćene redukcijom druge faze, pa i dalje ima više od jedne osobe u listi, što ubrzava traženje odgovarajuće osobe za generisanje ciklusa.

**phase_two_reduction** - Vrši redukciju druge faze na osnovu jednog ciklusa, krećući se po generisanom ciklusu, i uklanjajući odgovarajuće osobe iz listi preferencija, na već ranije opisan način.

**stable_roommates_algorithm** - Kombinuje obe faze u jednu celinu, vršeći redukciju prve faze, i potom dok god nije ispunjen uslov za zaustavljanje druge faze, poziva funkciju *phase_two_reduction*, izvršavajući jednu redukciju druge faze.

# 4 Složenost

Analiziraćemo posebno složenost prve i druge faze.

## 4.1 Prva faza

Prva faza nam se zasniva na pozivanjima funkcije **do_proposals**. Kako ona uvek nastavlja izvršavanje od mesta na kom je zaustavljeno slanje predloga u nekom prošlom pozivu, i nikad se ne vraća unazad, najgori slučaj je da je neko odbijen od strane svih ostalih, a ostali su stigli da pretposlednje osobe u svojim listama preferencija. Znači da je ovo reda veličine matrice, a kako su ostale operacije složenosti $O(1)$, znači da je složenost ove faze algoritma reda $O(n^2)$.

## 4.2 Druga faza

Druga faza se sastoji od generisanja sve ili ništa ciklusa, i redukcije liste preferencija na ranije opisani način, dok neki kriterijum zaustavljanja ne bude zadovoljen. Proveru uslova vršimo uz pomoć tehnike 2 pokazivača, održavajući 2 vektora, *leftmost* i *rightmost*, koji pokazuju na najlevlji i najdešnji element u listi preferencija svake osobe. Ovako je provera veličine liste preferencija neke osobe, odnosno zadovoljenja nekog uslova zaustavljanja, složenosti $O(1)$. Nakon svake redukcije druge faze ažuriramo na odgovarajući način ova dva vektora. Kako je, hipotetički, maksimalan broj elemenata koji mogu biti izbačeni tokom druge faze $n^2$, a složenost ostalih operacija, i generisanja ciklusa $O(1)$, sledi da je maksimalan broj koraka reda $O(n^2)$, samim tim je i složenost celog algoritma $O(n^2)$.

# 5 Rezultati

U ovom delu biće ukratko predstavljeni rezultati rada algoritma, dobijeni testiranjem algoritma na slučajno odabranim instancama problema. Sva testiranja sprovedena su na 1000 različitih instanci, za različite dimenzije ulaza. Dobijeni rezultati su predstavljeni u Tabeli 1.

| Dimenzija problema | Broj instanci | Broj sa rešenjem |
|---|---|---|
| 4 | 1000 | 966 |
| 6 | 1000 | 924 |
| 8 | 1000 | 879 |
| 10 | 1000 | 823 |
| 12 | 1000 | 776 |
| 14 | 1000 | 706 |
| 16 | 1000 | 626 |
| 18 | 1000 | 583 |
| 20 | 1000 | 500 |
| 22 | 1000 | 441 |
| 24 | 1000 | 408 |

Tabela 1: Rezultati rada algoritma

# 6 Zaključak

Videli smo definiciju zanimljivog problema stabilnih cimera, kao i algoritam za njegovo rešavanje. Nakon predstavljanja implementacije u C++u, analizirali smo složenost i zaključili da je u najgorem slučaju ona reda $O(n^2)$, što je efikasan i praktično upotrebljiv algoritam. Veći problem od samog vremena prave memorijski zahtevi za veliko $n$, pošto treba skladištiti veći broj dodatnih struktura podataka.
Kako postoji efikasan algoritam, ovaj problem bi mogao da nađe i praktične primene, kao na primer kod raspoređivanja studenata u domovima, ili kod raznih društvenih mreža i aplikacija za upoznavanje, gde bi ljudi bili povezivani prema njihovim interesima, hobijima i preferencijama.

# Literatura

[1] Robert W. Irving. An efficient algorithm for the "stable roommates" problem. *Journal of algorithms 6*, pages 577–595, 1985.