Racket: Problema mariajelor stabile

- Data publicării: 28.02.2023
- Data ultimei modificări: 28.02.2023 (changelog)
- Tema (o arhivă .zip cu toate fișierele .rkt folosite în etapa curentă) se va încărca pe vmchecker [https://vmchecker.cs.pub.ro/ui/#PP]

Descriere generală și organizare

Tema constă în rezolvarea problemei mariajelor stabile și este împărțită în **4 etape**:

- una pe care o veți rezolva după laboratorul 2 (cu deadline în ziua laboratorului 3, la ora 23:59)
- una pe care o veți rezolva după laboratorul 3 (cu deadline în ziua laboratorului 4, la ora 23:59)
- una pe care o veți rezolva după laboratorul 4 (cu deadline în ziua laboratorului 5, la ora 23:59)
- una pe care o veți rezolva după laboratorul 5 (cu deadline în ziua laboratorului 6, la ora 23:59)

Așa cum se poate observa, **ziua deadline-ului variază în funcție de semigrupa în care sunteți repartizați**. **Restanțierii care refac tema și nu refac laboratorul beneficiază de ultimul deadline** (deci vor avea deadline-uri în zilele de 23.03, 30.03, 06.04, 13.04).

Rezolvările tuturor etapelor pot fi trimise până în ziua laboratorului 6, dar orice exercițiu trimis după deadline (și până în ziua laboratorului 6) se punctează cu **jumătate** din punctaj. Orice exercițiu trimis după ziua laboratorului 6 nu se mai punctează deloc. Nota finală pe etapă se calculează conform formulei: **n = (n1 + n2) / 2** (n1 = nota obținută înainte de deadline; n2 = nota obținută după deadline). Când toate submisiile sunt înainte de deadline, nota pe ultima submisie este și nota finală (întrucât n1 = n2).

În fiecare etapă, veți folosi ce ați învățat în săptămâna anterioară pentru a dezvolta aplicația.

Pentru fiecare etapă veți primi un **schelet de cod** (dar rezolvarea se bazează în mare măsură pe rezolvările anterioare). Enunțul din această pagină este menit să descrie problema mariajelor stabile (pe care o vom denumi SMP, conform prescurtării în limba engleză) și să vină cu exemple de rulare a funcțiilor din schelet. Dacă preferați, odată ce ați înțeles problema, puteți rezolva tema utilizând doar indicațiile din schelet.

Etapa 1

În această etapă vă veți familiariza cu problema mariajelor stabile (SMP) și felul în care modelăm datele sale de intrare in Racket, apoi veți implementa câteva funcții de manipulare a acestor date.

În SMP se dau un număr egal de bărbați și femei și, pentru fiecare dintre aceștia, un "clasament" al tuturor persoanelor de sex opus, și se cere să se căsătorească fiecare bărbat cu câte o femeie astfel încât să nu existe un bărbat și o femeie în căsătorii diferite care s-ar prefera unul pe altul în detrimentul soților lor. Dacă ar exista o asemenea pereche, cei doi ar fi motivați să își părăsească partenerii pentru a fi împreună, așadar căsătoriile lor nu ar fi stabile (de aici numele problemei).

Vom modela o instanță SMP în Racket ca pe două liste (egale ca lungime) de preferințe masculine, respectiv feminine, ca în exemplul următor:

```
(define men-preferences
  '([adi  ana  bia cora]
    [bobo cora ana bia ]
    [cos  cora bia ana ]))
(define women-preferences
  '([ana  bobo adi cos ]
    [bia  adi  cos bobo]
    [cora bobo cos adi ]))
```

Astfel, lista de preferințe masculine/feminine este o listă de liste, fiecare listă interioară având pe prima poziție un bărbat/o femeie, iar pe următoarele poziții persoanele de sex opus în ordinea preferințelor acestui bărbat/acestei femei. De-a lungul întregii teme, o listă de preferințe masculine/feminine se va referi la o listă cu acest format. Când, în schimb, ne vom referi la lista preferințelor unei persoane p, ne vom referi la felul în care p a clasat persoanele de sex opus, așadar o listă care conține doar persoane de sex opus, nu și pe p pe prima poziție. Deși în exemplu bărbații/femeile apar în ordine alfabetică, nu presupuneți că acest lucru este necesar să se întâmple.

Aceste liste vor fi definite în checker, însă este necesar să le înțelegeți structura pentru a putea implementa funcțiile din cerință.

În etapa 1, veți exersa lucrul cu:

- liste și operatorii acestora (datorită modului de reprezentare a preferințelor masculine/feminine)
- perechi și operatorii acestora (pentru că fiecare logodnă este reprezentată ca o pereche cu punct; de-a lungul temei vom folosi termenul de logodnă pentru că algoritmul pentru SMP creează multe perechi temporare până a ajunge la soluția finală care reprezintă căsătoriile)
- funcții recursive pe stivă, respectiv pe coadă (observați tipul de recursivitate al fiecărei funcții implementate, și atenție la cazurile în care vi se solicită un anumit tip de implementare - chiar dacă obțineți punctaj pe checker, punctajul va fi anulat în cazul în care funcțiile nu sunt implementate conform specificației)
- operatori logici și condiționali (din nou, atenție la cazurile în care vi se cere să folosiți astfel de operatori)

Funcțiile pe care va trebui să le implementați sunt:

```
(get-men mpref)
(get-women wpref)
```

- get-men primește o listă de preferințe masculine și determină lista tuturor bărbaților din problemă
- get-women primește o listă de preferințe feminine și determină lista tuturor femeilor din problemă
- ex: `(get-men men-preferences)` (pentru men-preferences definit mai sus) ⇒ `'(adi bobo cos)`

```
(get-pref-list pref person)
```

- get-pref-list primește o listă de preferințe (masculine sau feminine) și o persoană person ale cărei preferințe apar (garantat) în listă și întoarce lista preferințelor lui person
- ex: `(get-pref-list women-preferences 'bia)` ⇒ `'(adi cos bobo)`

```
(preferable? pref-list x y)
```

- preferable? primește lista pref-list a preferințelor unei persoane și două persoane (x, y) care apar în pref-list și întoarce true dacă x apare înaintea lui y în pref-list, altfel false
- ex: `(preferable? '(adi cos bobo) 'bobo 'cos)` ⇒ `#f`

```
(get-partner engagements person)
```

- get-partner primește o listă de logodne (perechi cu punct) și o persoană și, în cazul în care această persoană apare pe prima poziție într-o logodnă, este întors partenerul său, altfel se întoarce false.
- ex: `(get-partner '((ana . cos) (bia . adi) (cora . bobo)) 'cora)` ⇒ `'bobo`
- ex: `(get-partner '((ana . cos) (bia . adi) (cora . bobo)) 'adi)` ⇒ `#f`

```
(better-match-exists? p1 p2 p1-list pref2 engagements)
```

- better-match-exists? primește 2 persoane logodite (p1, p2), lista preferințelor lui p1, lista preferințelor tuturor persoanelor de același gen cu p2, respectiv lista tuturor logodnelor (în care fiecare bărbat/femeie din problemă apare într-un cuplu) și întoarce true dacă există un partener mai potrivit pentru p1, și false altfel
- logodnele din listă au pe prima poziție persoana de același gen cu p2, pentru ca funcția get-partner să poată căuta doar după prima persoană din pereche (este responsabilitatea voastră ca, atunci când apelați better-match-exists? în etapele următoare, să vă asigurați că perechile au elementele în această ordine)
- p' este mai potrivit pentru p1 decât p2 dacă p1 și p' se preferă reciproc în detrimentul persoanelor cu care s-au logodit
- ex: `(better-match-exists? 'cos 'ana '(cora bia ana) women-preferences '((ana . cos) (bia . adi) (cora . bobo)))` ⇒ `#f`, pentru că, deși ana este ultima alegere a lui cos, atât cora cât și bia își preferă partenerii actuali
- ex: `(better-match-exists? 'cos 'ana '(cora bia ana) women-preferences '((ana . cos) (bia . bobo) (cora . adi)))` ⇒ `#t`, pentru că, de exemplu, cos o preferă pe cora anei, iar cora îl preferă pe cos lui adi

Etapa 2

În această etapă veți reimplementa majoritatea funcțiilor din etapa anterioară folosind ce ați învățat în utima săptămână. Apoi veți implementa verificarea stabilității unei liste de căsătorii.

Scopul etapei este consolidarea cunoștințelor legate de:

- funcționale (faptul că funcțiile sunt valori de ordinul întâi oferă atât posibilitatea de a lucra cu funcționalele din limbaj, cât și pe aceea de a implementa propriile funcționale, pentru orice șablon util pe care îl observați - din acest motiv, aveți inclusiv sarcina de a implementa două funcționale și a le folosi ulterior)
- funcții anonime
- programare de tip "wishful thinking" - veți implementa funcția stable-match? folosindu-vă de funcția better-match-exists?, întrucât noțiunea de stabilitate este definită ca "nu există un partener mai potrivit pentru niciunul dintre soți"; dacă nu ați fi avut deja funcția better-match-exists?, ar fi fost o idee foarte bună să programați stable-match? ca și cum ați fi implementat deja conceptul de partener mai potrivit (wishful thinking), și apoi să aveți grijă chiar să implementați această funcție ajutătoare

Funcțiile noi pe care va trebui să le implementați sunt:

```
(find-first p L)
```

- find-first primește un predicat p și o listă L și întoarce primul element din L care satisface p (sau false dacă nu există un asemenea element)
- ex: `(find-first odd? '(0 1 2 3))` ⇒ `1`
- ex: `(find-first odd? '(0 2 4))` ⇒ `#f`

```
(change-first p L val)
```

- change-first primește un predicat p, o listă L și o valoare val și întoarce lista L în care primul element care satisface p a fost înlocuit cu valoarea val (sau lista L nemodificată dacă nu există un asemenea element)
- ex: `(change-first odd? '(0 1 2 3) 'unu)` ⇒ `'(0 unu 2 3)`
- ex: `(change-first odd? '(0 2 4) 1)` ⇒ `'(0 2 4)`

```
(update-engagements engagements p1 p2)
```

- update-engagements primește o listă de logodne engagements și două persoane p1 și p2 și întoarce lista engagements în care partenerul lui p1 a fost actualizat la valoarea p2
- se garantează că p1 apare în engagements și că toate cuplurile din engagements au pe prima poziție persoana de același gen cu p1
- ex: `(update-engagements '((ana . cos) (bia . adi)) 'bia 'bobo)` ⇒ `'((ana . cos) (bia . bobo))`

```
(stable-match? engagements mpref wpref)
```

- stable-match? primește o listă completă de logodne engagements, o listă de preferințe masculine mpref și o listă de preferințe feminine wpref și întoarce true dacă toate cuplurile din engagements sunt stabile
- un cuplu este stabil dacă pentru niciunul dintre soți nu există un alt partener mai potrivit
- fiecare logodnă este de forma '(femeie . bărbat) (femeia apare pe prima poziție)
- ex: `(stable-match? '((ana . cos) (bia . adi) (cora . bobo)) men-preferences women-preferences)` ⇒ `#f` întrucât, de exemplu, pentru ana există un partener mai potrivit:
  - ana îl preferă pe adi soțului său cos
  - adi o preferă pe ana soției sale bia
- ex: `(stable-match? '((ana . adi) (bia . cos) (cora . bobo)) men-preferences women-preferences)` ⇒ `#t`, întrucât nu putem găsi o situație ca cea din exemplul anterior

Etapa 3

În această etapă veți rezolva problema mariajelor stabile folosind algoritmul Gale-Shapley și veți implementa câteva funcții necesare etapei următoare. Observați că la începutul fișierului etapa3.rkt apare linia `(require "etapa2.rkt")`, ceea ce înseamnă că va fi necesar să aduceți rezolvarea etapei 2 în același folder în care rezolvați etapa 3, pentru a beneficia de funcțiile implementate anterior.

Veți continua să folosiți recursivitate și funcționale (facilități fundamentale în programarea funcțională), dar scopul principal al etapei este lucrul cu expresii de legare statică a variabilelor:

- let, let* - pentru evitarea calculelor duplicate
- named let, letrec - pentru implementarea ad-hoc a proceselor recursive, fără a apela la funcții ajutătoare

Funcțiile pe care va trebui să le implementați sunt:

```
(get-unstable-couples engagements mpref wpref)
```

- get-unstable-couples primește o listă engagements de logodne (unde fiecare cuplu are pe prima poziție o femeie), respectiv două liste mpref/wpref de preferințe masculine/feminine și întoarce lista tuturor cuplurilor instabile din engagements
- ex: `(get-unstable-couples '((ana . cos) (bia . adi) (cora . bobo)) men-preferences women-preferences)` ⇒ `'((ana . cos) (bia . adi))` (întrucât ana și adi se preferă reciproc în dauna partenerilor lor actuali) (ordinea cuplurilor în rezultat nu contează, checker-ul sortează singur returul funcției get-unstable-couples)

```
(engage free-men engagements mpref wpref)
```

- engage primește o listă free-men de bărbați încă nelogodiți, o listă engagements de logodne parțiale (unde fiecare cuplu are pe prima poziție o femeie), și două liste mpref/wpref de preferințe

masculine/feminine și întoarce lista completă de logodne stabile, obținută conform algoritmului Gale-Shapley
- algoritmul Gale-Shapley funcționează astfel:
  - inițial nimeni nu este logodit, și toate cererile în căsătorie vor fi inițiate de bărbați
  - la o iterație oarecare din timpul algoritmului, vom avea situația surprinsă de funcția engage: o submulțime de bărbați încă nelogodiți, respectiv o mulțime de logodne încheiate până în prezent
  - în această iterație (ca și în toate celelalte), un bărbat actualmente nelogodit o cere în căsătorie pe prima femeie din lista sa de preferințe pe care nu a cerut-o încă
    - dacă ea este liberă sau preferă acest bărbat celui cu care este logodită, cei doi se logodesc (și eventualul logodnic anterior devine liber)
    - altfel, această femeie nu este o opțiune, și bărbatul va încerca să o ceară pe următoarea femeie din lista sa, etc.
- ex: `(engage '(adi bobo) '((cora . cos)) men-preferences women-preferences)` ⇒ `'((ana . adi) (bia . cos) (cora . bobo))` (ordinea cuplurilor în rezultat nu contează, checker-ul sortează singur returul funcției engage)
  - în acest exemplu, bărbații nelogodiți sunt adi și bobo, iar singurul cuplu logodit este compus din cora și cos
  - adi (primul bărbat nelogodit) își consultă lista de preferințe - `'(ana bia cora)`, și o cere în căsătorie pe ana
  - întrucât ana este liberă, ea acceptă
  - doar bobo rămâne nelogodit, iar cuplurile actuale sunt `'((ana . adi) (cora . cos))`
  - bobo (primul bărbat nelogodit) își consultă lista de preferințe - `'(cora ana bia)`, și o cere în căsătorie pe cora
  - întrucât cora (a cărei listă este `'(bobo cos adi)`) îl preferă pe bobo lui cos (cu care este momentan logodită), ea acceptă
  - cos devine singurul bărbat nelogodit, iar cuplurile actuale sunt `'((ana . adi) (cora . bobo))`
  - cos (primul bărbat nelogodit) își consultă lista de preferințe - `'(cora bia ana)`, și o cere în căsătorie pe bia (întrucât pe cora a cerut-o deja, au fost logodiți; ar putea să o ceară și pe cora din nou, care ar refuza - rezultatul nu ar fi afectat, este doar o chestiune de eficiență)
  - întrucât bia este liberă, ea acceptă
  - în acest moment, nu mai există bărbați nelogodiți, ceea ce înseamnă că lista completă de logodne stabile este `'((ana . adi) (bia . cos) (cora . bobo))`

```
(gale-shapley mpref wpref)
```

- gale-shapley primește două liste mpref/wpref de preferințe masculine/feminine și folosește funcția engage pentru a determina o listă completă de logodne stabile corespunzătoare acestor preferințe
- ex: `(gale-shapley men-preferences women-preferences)` ⇒ `'((ana . adi) (bia . cos) (cora . bobo))` (ca mai sus, ordinea cuplurilor nu contează)

```
(get-couple-members pair-list)
```

- get-couple-members primește o listă de cupluri (perechi cu punct) și întoarce lista persoanelor din aceste cupluri (în alte cuvinte, aplatizează lista de perechi cu punct)
- ex: `(get-couple-members '((ana . adi) (cora . bobo)))` ⇒ `'(ana adi cora bobo)` (ordinea elementelor nu contează)

Etapa 4

În această etapă veți rezolva SMP pentru cazul în care preferințele evoluează în timp, așa cum se întâmplă adesea în realitate. Orice schimbare poate afecta stabilitatea căsătoriilor obținute anterior, dar - având în vedere că preferințele nu se schimbă radical de la un moment la altul - pare mai puțin costisitor ca în loc să rulăm algoritmul Gale-Shapley de la zero de fiecare dată când se schimbă ceva, să pornim de la soluția anterioară și să o actualizăm pe aceasta în cazul în care identificăm în ea cupluri instabile.

Observați că la începutul fișierului etapa4.rkt se solicită să aduceți în folderul curent rezolvările etapelor 2 și 3, pentru a beneficia de funcțiile implementate anterior.

Veți exersa în continuare stilul funcțional de a programa studiat în săptămânile anterioare, plus conceptul de flux introdus la ultimul laborator. Întrucât o instanță SMP este reprezentată prin două liste de preferințe masculine, respectiv feminine, vom reprezenta SMP cu preferințe dinamice ca pe un flux de instanțe SMP, fiecare element din flux reprezentând preferințele la un anumit moment în timp. Soluția SMP cu preferințe dinamice trebuie calculată tot sub formă de flux, unde fiecare element este o listă de căsătorii stabile conform preferințelor la momentul respectiv. Veți obține primul element din fluxul rezultat aplicând algoritmul Gale-Shapley asupra primului element din fluxul de intrare, iar pe următoarele prin aplicarea succesivă a algoritmului de actualizare prezentat mai jos.

Algoritmul de actualizare este bazat pe o metodă introdusă de Roth și Vande, și funcționează în felul următor:

- pornește de la soluția anterioară - căsătoriile stabile la momentul anterior
- dintre acestea, elimină cuplurile care au devenit instabile în urma schimbărilor de preferințe
- consideră că toate cuplurile rămase (care sunt stabile între ele) se găsesc împreună într-o cameră, în timp ce toate celelalte persoane din problemă așteaptă la coadă la intrarea în cameră
- prima persoană din coadă intră în cameră și încearcă să se cupleze cu cineva care este deja acolo, în ordinea dată de lista sa de preferințe; procesul de potrivire (descris în detaliu în scheletul de cod) continuă până când situația din cameră este stabilă (toate cuplurile sunt stabile între ele, și - în caz că există persoane libere - acestea nu sunt preferate de o persoană logodită în dauna partenerului său)
- repetă pasul anterior până când se termină coada iar în cameră sunt numai cupluri stabile, reprezentând noua soluție

Toate funcțiile din această etapă întorc una sau mai multe liste de cupluri. Așa cum v-ați obișnuit în etapa anterioară, ordinea cuplurilor în listă nu contează (dar pe prima poziție a fiecărui cuplu se va găsi un bărbat/o femeie, în funcție de cum cere fiecare funcție în parte). Aveți de implementat următoarele funcții:

```
(match person engagements pref1 pref2 queue)
```

- match primește o persoană person care intră în cameră, o listă engagements de logodne parțiale ale celorlalte persoane din cameră, două liste pref1/pref2 de preferințe ale persoanelor de același gen/gen opus lui person, și o coadă queue a persoanelor din afara camerei și întoarce lista engagements actualizată astfel încât situația din cameră să fie stabilă
- ex: `(match 'ana '((bobo . cora) (adi . bia)) women-preferences-0 men-preferences-0 '(cos))` ⇒ `'((#f . bia) (bobo . cora) (adi . ana))`
  - observație: cuplurile au pe prima poziție persoanele de gen opus anei
  - ana intră în cameră, își consultă lista de preferințe - `'(bobo adi cos)` și, întrucât bobo este în cameră, încearcă să se cupleze cu acesta
  - bobo o preferă pe cora, astfel încât rămâne logodit cu ea
  - atunci, ana încearcă să se cupleze cu adi (următorul pe listă care se află în cameră)
  - adi o preferă pe ana, așadar cei 2 se logodesc
  - noile logodne din cameră sunt `'((bobo . cora) (adi . ana))`, iar bia devine liberă și încearcă să își găsească un partener în cameră conform aceluiași algoritm
  - bia își consultă lista de preferințe - `'(adi cos bobo)`

- ea încearcă pe rând să se cupleze cu adi, respectiv bobo (nu și cu cos, pentru că el nu este în cameră), dar atât adi cât și bobo își preferă partenerele actuale, așadar bia rămâne momentan singură, ceea ce înseamnă că o logodim fictiv cu valoarea false
- rezultatul final este `'((#f . bia) (bobo . cora) (adi . ana))`

```
(path-to-stability engagements mpref wpref queue)
```

- path-to-stability primește o listă engagements de logodne parțiale ale persoanelor din cameră, două liste mpref/wpref de preferințe masculine/feminine, și o coadă queue a persoanelor din afara camerei și întoarce lista completă de logodne stabile, obținută în urma intrării pe rând a tuturor persoanelor din coadă în cameră, fiecare intrare generând un nou proces de match
- ex: `(path-to-stability '((cora . bobo) (bia . adi)) men-preferences-0 women-preferences-0 '(ana cos))` ⇒ `'((bia . cos) (cora . bobo) (ana . adi))`
  - observație: cuplurile au pe prima poziție o femeie
  - ana este prima care intră în cameră și trece prin procesul de match
  - în urma acestuia, situația din cameră este: `'((bia . #f) (cora . bobo) (ana . adi))`
  - apoi cos intră în cameră și trece prin procesul de match
  - în urma acestuia, situația din cameră este: `'((bia . cos) (cora . bobo) (ana . adi))`
  - întrucât toate persoanele au intrat în cameră, acesta este și rezultatul final

Pentru exemplificarea următoarelor două funcții, vom folosi următoarele liste de preferințe reflectând situația la două momente de timp t_0 și t_1 (singura diferență apare la preferințele lui adi):

```
(define men-preferences-0       (define men-preferences-1
  '([adi  ana  bia cora]          '([adi  cora  bia ana]
    [bobo cora ana bia ]            [bobo cora ana bia]
    [cos  cora bia ana ]))          [cos  cora bia ana]))
(define women-preferences-0     (define women-preferences-1
  '([ana  bobo adi cos ]          '([ana  bobo adi cos ]
    [bia  adi  cos bobo]            [bia  adi  cos bobo]
    [cora bobo cos adi ]))          [cora bobo cos adi ]))
```

```
(update-stable-match engagements mpref wpref)
```

- update-stable-match primește o listă completă de logodne engagements (soluția SMP de la momentul anterior de timp), respectiv două liste mpref/wpref de preferințe masculine/feminine (de la momentul actual) și întoarce o nouă listă completă de logodne stabile (conform cu preferințele actuale)
- ex: `(update-stable-match '((ana . adi) (bia . cos) (cora . bobo)) men-preferences-1 women-preferences-1)` ⇒ `'((ana . cos) (bia . adi) (cora . bobo))`
  - se determină cuplurile care au devenit instabile în urma modificării preferințelor: `'((bia . cos) (ana . adi))` (întrucât acum bia și adi se preferă reciproc în dauna partenerilor lor)
  - în consecință, la început vom avea cuplul `'(cora . bobo)` în cameră, iar coada va fi `'(ana adi bia cos)` (nu neapărat în această ordine)
  - după intrarea anei în cameră, situația din cameră devine `'((ana . #f) (cora . bobo))`
  - după intrarea lui adi, situația devine `'((ana . adi) (cora . bobo))`
  - după intrarea biei, situația devine `'((ana . #f) (bia . adi) (cora . bobo))`
  - după intrarea lui cos, situația devine `'((ana . cos) (bia . adi) (cora . bobo))`, care este și rezultatul final, întrucât nu mai există persoane la coadă

```
(build-stable-matches-stream pref-stream)
```

- build-stable-matches-stream primește un flux de instanțe SMP (reprezentând preferințele acelorași persoane la momente succesive de timp) și întoarce fluxul de soluții SMP corespunzătoare acestor instanțe
- pentru exemplificare, considerăm că am definit fluxul pref-stream-a care are pe prima poziție perechea `(cons men-preferences-0 women-preferences-0)` și pe a doua (și ultima) poziție perechea `(cons men-preferences-1 women-preferences-1)`
- ex: `(build-stable-matches-stream pref-stream-a)` ⇒ un flux care are pe prima poziție căsătoriile `'((ana . adi) (bia . cos) (cora . bobo))` și pe a doua poziție căsătoriile `'((ana . cos) (bia . adi) (cora . bobo))`
  - primul element Sol_0 din fluxul rezultat se obține aplicând algoritmul Gale-Shapley asupra preferințelor de la momentul t_0
  - al doilea element din fluxul rezultat se obține aplicând update-stable-match asupra lui Sol_0 și a preferințelor de la momentul t_1

Precizări

- Exercițiile din fiecare etapă se rezolvă în fișierul care poartă numele etapei (**etapa1.rkt**, **etapa2.rkt**, etc.). Pentru testare, veți rula codul din fișierul **checker.rkt**.
- Fiecare etapă (o arhivă .zip cu fișierul care poartă numele etapei plus eventualele fișiere care sunt solicitate de acesta cu "require") se va încărca pe vmchecker. Testele de vmchecker sunt aceleași cu cele din checker.rkt. Pentru etapele 3 și 4 testele folosesc funcția stable-match? implementată în etapa 2. Pentru a preveni teste care trec sau nu trec din cauză că funcția stable-match? este implementată greșit, pe vmchecker vom folosi un stable-match? implementat de noi (deci în această situație puteți obține punctaje diferite local față de vmchecker).
- Dacă doriți să rezolvați exerciții din etapa curentă care depind de exerciții din etapele anterioare pe care nu le-ați rezolvat, puteți semnala acest lucru asistentului, care vă va pune la dispoziție o rezolvare pentru acele exerciții astfel încât să puteți continua tema. Odată ce alegeți această variantă, nu mai puteți primi punctaj pe exercițiile pentru care ați primit rezolvarea, chiar daca deadline-ul lor nu a expirat. Puteți solicita rezolvări doar pentru exercițiile din etapele anterioare, nu și pentru cele din etapa curentă. Dacă implementați un exercițiu din etapa curentă pe baza unui alt exercițiu din etapa curentă pe care nu l-ați rezolvat, se va lua în calcul punctajul dat de checker, chiar dacă implementarea ar funcționa în caz că exercițiul nerezolvat ar funcționa.
- Tema este în primul rând o temă de programare funcțională - pentru care folosim Racket. Racket este un limbaj multiparadigmă, care conține și elemente "ne-funcționale" (de exemplu proceduri cu efecte laterale), pe care **nu** este permis să le folosiți în rezolvare.
- Pentru fiecare etapă, checker-ul vă oferă un punctaj între 0 și 120 de puncte. Pentru a obține cele 1.33p din nota finală cu care este creditată tema de Racket, este suficient să acumulați 400 de puncte de-a lungul celor 4 etape. Un punctaj între 400 și 480 se transformă într-un bonus proporțional.
- Veți prezenta tema asistentului, care poate modifica punctajul dat de checker dacă observă nereguli precum răspunsuri hardcodate, proceduri cu efecte laterale, implementări neconforme cu restricțiile din enunț.

Resurse

- etapa 1
- etapa 2

- etapa 3
- etapa 4

Changelog

- 28.02 (ora 23:25) - Am publicat etapa 1 (enunț + schelet), și etapele 2-4 (momentan doar enunț).
- 11.03 (ora 12:25) - Am publicat etapa 2.
- 17.03 (ora 21:50) - Am publicat etapa 3.
- 25.03 (ora 09:50) - Am publicat etapa 4.

Referinţe

- Matching-Updating Mechanism: A Solution for the Stable Marriage Problem with Dynamic Preferences [https://www.ncbi.nlm.nih.gov/pmc/articles/PMC8871443/]

pp/23/teme/racket-smp.txt · Last modified: 2023/03/25 09:51 by mihaela.balint