nedeljainformatike

OPŠTI TEST IZ RAČUNARSKIH NAUKA

12. februar 2018. — 26. februar 2018.

Pre nego što počnete, pažljivo pročitajte ove instrukcije:

Pred vama je pet zadataka iz raznih oblasti računarskih nauka, za čiju izradu imate tačno **dve nedelje**. Kao deo prijave za Nedelju informatike, potrebno je da, pored popunjenih osnovnih podataka, pošaljete vaša rešenja za tačno **dva** zadatka. Ukoliko pošaljete rešenje za više od dva problema, **računaće se samo dva najbolje urađena**.

Neki problemi su namerno **nedovoljno definisani** ili nemaju jedinstveno "tačno" rešenje. Potrebno je da, u okvirima koje postavlja zadatak, razvijete što bolje rešenje za takve probleme, sa obrazloženjem za odluke ili pretpostavke koje ste napravili.

Preporučujemo da odabrane probleme rešavate **što je kompletnije moguće**. Radi ilustracije, jedan kompletno urađen problem će biti više vrednovan nego dva problema urađena do pola.

Uzimajući u obzir datu količinu vremena, dati problemi nisu jednostavni, i očekivanje je da će solidan kandidat uspešno rešiti **neznatno više od polovine problema koje je odabrao**.

Molimo vas da probleme rešavate **samostalno**!

Zadatke pripremili:

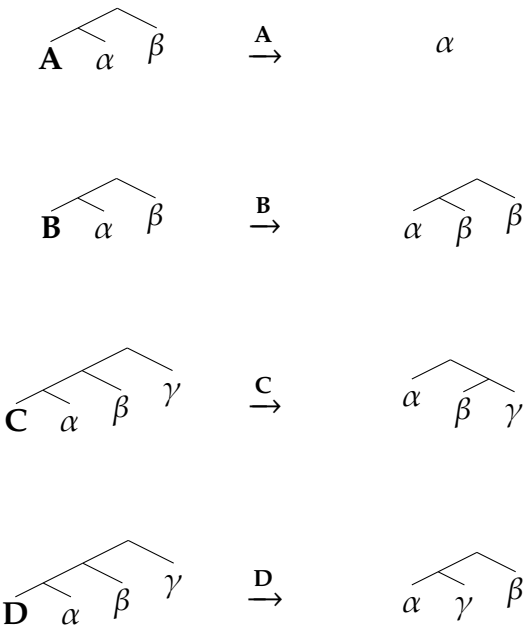
Andrej Ivašković, Univerzitet Kembridž
Dimitrije Erdeljan, Univerzitet Kembridž
Edgar Liberis, Featurespace
Filip Vesović, Elektrotehnički fakultet
Petar Veličković, Univerzitet Kembridž
Vladimir Milenković, Univerzitet Kembridž

1 NI-stabla

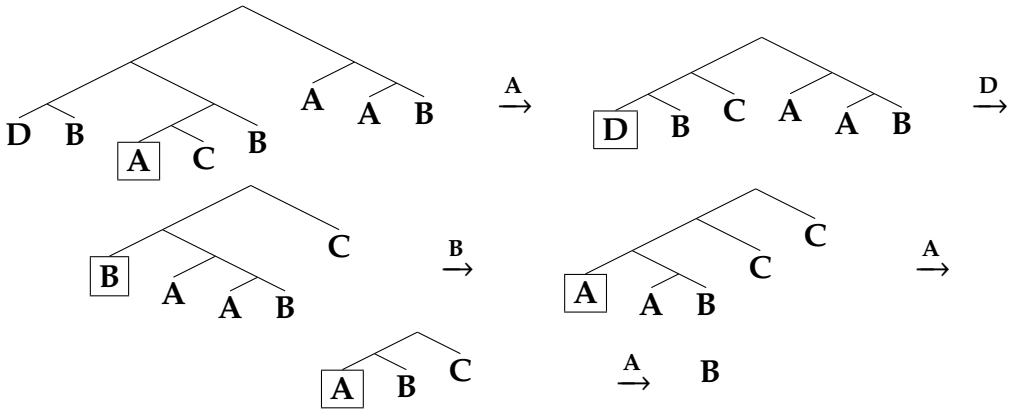
NI-stabla su specijalna vrsta binarnih stabala čije listove označavamo slovima **A**, **B**, **C**, **D** i mogu da podležu **redukcijama** u zavisnosti od svoje strukture. Unutrašnjim čvorovima ne delimo oznake. Redosled podstabala (šta je levo, a šta desno) je bitan.

U daljem tekstu grčka slova α, β, γ predstavljaju proizvoljna podstabla.

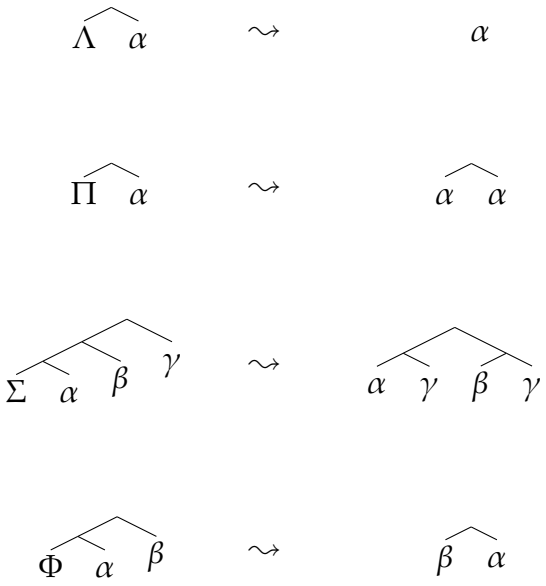
Moguće transformacije (u bilo kom podstablu koje odgovara nekoj od narednih struktura) su:



NI-stablo može da podlegne nizu redukcija i postoji mogućnost da može da se svede na **ireducibilno** NI-stablo (formu u kojoj nijedna od prethodno navedenih redukcija ne može da se primeni). Naredni primer pokazuje kako se jedno NI-stablo svodi na ireducibilno:

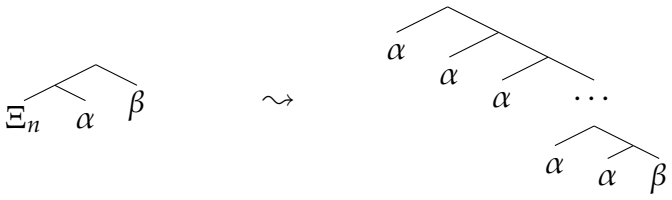


(*a*) Označimo sa $\alpha \rightsquigarrow \beta$ iskaz "NI-stablo α može da se svede na β primenom osnovnih redukcija". Osmisliti podstabla (ili dokazati da ne postoje) Λ, Π, Σ i Φ takva da za sve α, β, γ važi:



(*b*) Postoji li NI-stablo Ω koje ne može da se svede na ireducibilnu formu? Obrazložiti odgovor (primerom ili dokazom).

(*c*) Odrediti sve brojeve $n \in \mathbb{N}_0$ za koje postoji stablo Ξ_n koje zadovoljava:



gde se α u desnom stablu pojavljuje n puta. Obrazložiti odgovor (najbolje rešenje je dokaz).

Vaše odgovore na ova pitanja predajte u vidu `.pdf` fajla sa rešenjima i obrazloženjima. Zbog količine crtanja (i grčkih slova) je dozvoljeno da predate *skenirana* ili *fotografisana* rukom napisana rešenja *dokle god su čitljiva* (ukoliko se odgovor ne vidi jasno usled lošeg rukopisa ili kvaliteta slike, neće biti priznat).

2 Kriptovane slike

U arhivi `http://csnedelja.mg.edu.rs/static/resources/crypto-images.zip` nalazi se direktorijum `input/` sa četiri fajla: `01.png`, `02.png`, `03.png` i `04.png`. Svaki od njih je dobijen enkriptovanjem fotografije, koja (između ostalog) sadrži papir sa tekstom oblika `flag{...}`. Algoritmi koji su korišćeni za enkripciju su opisani kasnije u tekstu.

Vaš zadatak je da dekriptujete slike i za svaku sliku pošaljete: tekst sa papira (`flag{...}`), kod korišćen za dekripciju, i `.txt` ili `.pdf` fajl sa detaljnim opisom vašeg algoritama.

U direktorijumu `scramblers/` nalaze se implementacije algoritama korišćenih za enkripciju. Za njihovu kompilaciju potrebni su fajlovi koji se nalaze u `include/`, a to su: biblioteka *lodepng* (fajlovi `lodepng.h` i `lodepng.cpp`) i `image.h`, koji definiše strukture `Image` i `Pixel`, kao i funkcije za čitanje i pisanje `.png` fajlova (koristeći *lodepng*). Ovaj header vam može biti od koristi ako algoritme za dekripciju implementirate u C++-u (naravno, nije ga obavezno koristiti).

Ukoliko ručno kompajlirate ove fajlove, neophodno je u isto vreme kompajlirati i relevantni algoritam i biblioteku *lodepng*, npr. sledećom komandom:

```
g++ -std=c++11 -O2 scramblers/01.cpp include/lodepng.cpp
```

a ukoliko ih kompajlirate koristeći okruženje poput *Code::Blocks*, neophodno je ubaciti sve relevantne fajlove u vaš projekat (uključujući i `lodepng.cpp`).

Vaš algoritam može da ima ljudsku "pomoć" (da neke odluke donosi čovek), ali za maksimalan broj poena rešenje bi trebalo da bude u što većoj meri automatizovano (ukoliko je moguće, u potpunosti, a u suprotnom sa što manje pomoći od korisnika).

U datim opisima algoritama, `D` i `E` su originalna (ulazna) i enkriptovana (izlazna) slika. Obe su matrice sa `n` redova i `m` kolona, gde su elementi uređene trojke celih brojeva iz intervala $[0, 255]$ – redom vrednosti crvenog, zelenog i plavog kanala za taj piksel. `K` označava ključ za enkripciju. Operator `^` je bitovna operacija *xor*. Svi nizovi su indeksirani počev od nule.

(*a*) Slika `01.png` je enkriptovana sledećim algoritmom, gde je ključ `K` ceo broj iz intervala $[0, 255]$:

```
for i = 0 .. n-1:
    for j = 0 .. m-1:
        let (r, g, b) = D[i,j]
        E[i,j] = (r^K, g^K, b^K)
```

(*b*) Slika `02.png` je enkriptovana sledećim algoritmom, gde je ključ `K` niz od tri elementa iz $[0, 255]$:

```
for i = 0 .. n-1:
    for j = 0 .. m-1:
        let (r, g, b) = D[i,j]
        E[i,j] = (r^K[0], g^K[1], b^K[2])
```

(*c*) Slika `03.png` je enkriptovana sledećim algoritmom, gde je ključ `K` ceo broj iz intervala $[0, 2^{20} - 1]$:

```
for i = 0 .. n-1:
    for j = 0 .. m-1:
        let (r, g, b) = D[i,j]
        r' = r ^ next()
        g' = g ^ next()
        b' = b ^ next()
        E[i,j] = (r', g', b')
```

Funkcija `next()` modifikuje `K` i definisana je na sledeći nacin, gde je `&` bitovna operacija *and*:

```
next():
    K = 19997 * K + 31
    res = (K >> 20) & 0xFF
    K = K & 0xFFFFF
    return res
```

(*d*) Slika `04.png` je enkriptovana sledećim algoritmom, gde je K niz od n celih brojeva iz intervala $[0, m - 1]$:

```
for i = 1 .. n:
    for j = 1 .. m:
        E[i,j] = D[i, (j + K[i]) mod m]
```

U ovom podzadatku, nije neophodno da "idealno" dekriptujete sliku – da biste osvojili bar deo poena, dovoljno je da pročitate tekst sa papira (`flag{...}`). Slike bliže originalu nose više bodova, ali idealno rešenje nije neophodno za maksimalan broj poena.

3 Prevare sa kreditnim karticama

U ovom pitanju ćete se baviti dizajnom jednostavnog sistema za detekciju neuobičajenih transakcija sa kreditnim karticama (i potencijalnim prevarama).

Date transakcije se izvršavaju u realnom vremenu, a sistem za detekciju prevare koji treba da se napravi je odvojen od postojećeg sistema koji se bavi stanjem na računu klijenata. Sistemu za detekciju prevare se zadaju, jedna po jedna, informacije o svakoj transakciji (u redosledu kojim su zahtevi stigli banci). Relevantni podaci o transakciji su *vreme* (zadato u obliku Unix *timestamp*), *broj kartice* (šesnaestocifreni prirodan broj, predstavljen kao string), *ID vlasnika kartice* (jedinstveni 64-bitni broj), *ID prodavca* (takođe jedinstveni 64-bitni broj), *vrednost transakcije*, kao i *da li je banka prihvatila transakciju* (1 za *da* ili 0 za *ne*).

Cilj ovog zadatka je da se izgradi sistem koji će automatski da označi neobične transakcije *onog trenutka kada se pojave*, nakon čega će ljudski ekspert izvršiti inspekciju. Odlučeno je da se ovom problemu pristupi polazeći od pretpostavke da su sumnjive velike transakcije koje znatno odstupaju od tipične transakcije vlasnika kartice.

Resursi za ovaj zadatak mogu da se nađu na sledećoj adresi:

`http://csnedelja.mg.edu.rs/static/resources/fraud.zip`

Dat vam je primer fajla koji sadrži ove podatke. Takođe vam je dat i jednostavan C++ program, `fraud.cpp`, koji može da ih obrađuje. U `main` funkciji se učitava fajl sa ulaznim podacima i prosleđuje relevantnoj funkciji. Ovaj kod možete da menjate po svojoj volji (dokle god ulazni podaci ostaju isti).

(*a*) Za detektovanje neobičnih transakcija je neophodno sakupiti neke podatke o svakom klijentu.

(*i*) Odlučeno je da se detekcija vrši isključivo preko broja transakcija nekog klijenta N, srednje vrednosti uzorka $\overline{X}$ i procene standardne devijacije $\widehat{\sigma}$, gde je:

$$\overline{X} = \frac{\sum_{k=1}^{N} X_k}{N}$$

$$\widehat{\sigma} = \sqrt{\frac{1}{N-1} \sum_{k=1}^{N} \left(X_i - \overline{X}\right)^2}$$

ako su $X_1, X_2, \ldots, X_N$ vrednosti transakcija. Implementirati i objasniti efikasan način da se, po prijemu nove transakcije, ažuriraju svi neophodni podaci za odgovarajućeg klijenta.

(*ii*) Dovršiti kod koji detektuje sumnjivost transakcije i objasniti logiku, pri čemu funkcija `sumnjivaTransakcija(Transakcija)` ne sme da koristi atribut `prihvaceno`. Po kojim kriterijumima je ovo rešenje dobro?

(*b*) Uočeno je da klijenti troše različite količine novca u zavisnosti od meseca i dana. Odlučeno je da bi zato ovaj sistem trebalo da uzima isključivo skorije transakcije – konkretno, u obzir bi trebalo da se uzmu isključivo one iz otprilike prethodnih L dana. Na primer, dve konkretne bitne vrednosti za L koje bi mogle da se koriste su $L = 1$ i $L = 30$. Osmisliti način da sistem vodi računa o ovome. Nije neophodno pisati kod, detaljno objašnjenje je dovoljno.

(*c*) Objasniti dve slabosti ovakvog sistema koje omogućavaju prevaru.

(*d*) Očekuje se da ovaj sistem vodi procesira $2,5$ milijardi transakcija koje obuhvataju 700 miliona klijenata. Ne bi trebalo memorisati više od 2 KB podataka po korisniku.

(*i*) Uz ova ograničenja, navesti nedostatke aktuelnog sistema i kako bi mogli da se poprave.

(*ii*) Traži se da izgrade podaci o $2,5$ milijarde starih transakcija aktuelnih klijenata pre nego što se sistem pusti u rad. Predložiti način da se ovaj posao uradi na jednoj jedinoj (modernoj) mašini što je moguće efikasnije. Pretpostaviti da mašina ima 32 GB RAM memorije, klasičan hard disk i da podaci o transakcijama zauzimaju veći deo hard diska. Eksplicitno navesti sve ostale pretpostavke.

Vaš odgovor treba da bude predat u vidu arhive koja sadrži `.txt/.pdf` fajl sa odgovorima, kao i izvorni kod programa koji vrši klasifikaciju koja se traži u zadatku (on bi trebalo da bude zasnovan na `fraud.cpp` – može da bude napisan i u jeziku koji nije C++, dokle god je jasno da je osnova ekvivalentna).

4 Pokemon

Pokemoni (ili *Pocket Monsters*) su čudovišta kao što svi dobro znamo. Rešavanjem ovog zadatka skupljaćete Pokemone, a vaš cilj je skupiti ih sve.

Na linku `http://csnedelja.mg.edu.rs/static/resources/pokemon.zip` nalazi se arhiva u kojoj se nalazi zadatak.

NAPOMENA: Arhiva `pokemon.zip` je ažurirana *17. februara*, zbog problema sa ključem u jednom problemu. Molimo vas da preuzmete novu arhivu.

NAPOMENA: U *Charizard* podzadatku, traži se broj **uređenih**, a ne neuređenih parova (i, j).

Zadatak se sastoji iz više algoritamskih problema (sličnim onima koji se susreću na takmičenjima), podeljenih u tri grupe. Zadaci unutar jedne grupe su sortirani po težini, a na samom početku vidljivi su vam samo početni problemi iz svake grupe. Rešavanjem tih problema vi otključavate naredne (teže) probleme.

Svaki zadatak se nalazi u zasebnom folderu koji sadrzi: tekstualni fajl sa postavkom zadatka i ograničenjima, fajl sa `.in` ekstenzijom koji predstavlja test primer i zaključanu arhivu ka narednom problemu.

Da biste generisali šifru za otključavanje narednog zadatka, potrebno je da kada rešite zadatak i otkučate ga u jeziku po izboru, generišete izlazni fajl koji odgovara test primeru. Nakon toga, na sajt:

`https://www.browserling.com/tools/remove-all-whitespace`

kopirajte celokupan sadržaj izlaznog fajla i uklonite blanko znakove, a onda na sajtu

`http://passwordsgenerator.net/sha1-hash-generator`

generišite SHA heš dobijenog izlaza (kome su uklonjeni blanko znakovi). Dobijeni SHA heš predstavlja šifru kojom možete pristupiti narednom problemu.

Napomena. Vodite računa da prilikom kopiranja izlaza kome su uklonjeni blanko znakovi ne dodate neki blanko znak (kao sto je prelaz u narednu liniju).

Za predaju ovog zadatka neophodno je poslati:

- generisane ključeve kojim su otključani naredni problemi;
- kratka objašnjenja rešenja i procene vremenskih složenosti;
- implementacije rešenja.

5 Zamalo tačno

Pri izradi nekih delova ovog zadatka, potencijalno će vam biti korisni podsetnici sa prošlogodišnjeg predavanja o *Izračunljivosti i složenosti* i *NP-kompletnosti*. Oba podsetnika možete pronaći na stranici:

`http://csnedelja.mg.edu.rs/materijali`

U ovom zadatku je potrebno predati `.zip` arhivu koja u sebi sadrži `.pdf` fajl sa vašim rešenjima za sve delove ovog zadatka, kao i izvorni kod (npr. `.cpp` fajl ukoliko ste koristili C++) vaše implementacije dela *(b) (i)*.

(*a*) Problem **Hamiltonovog ciklusa** podrazumeva određivanje da li u datom neusmerenom i netežinskom grafu, postoji ciklus koji posećuje svaki od njegovih čvorova tačno jednom. Ovo je jedan od najpoznatijih *NP-kompletnih problema*. Navedite kratku, neformalnu, definiciju NP-kompletnosti, kao i implikacije na efikasnu "rešivost" ovog problema.

(*b*) Mnogi problemi koji su NP-kompletni su i dalje previše važni da bi bili napušteni—ponekad ćemo biti zadovoljni i *približnim rešenjima*! Definisaćemo algoritam sa *stopom približnosti* ρ kao algoritam za koji uvek važi:

$$\max\left(\frac{C}{C^*}, \frac{C^*}{C}\right) \leq \rho$$

gde je C^* optimalno rešenje za neki ulaz, a C rešenje koje vraća algoritam.

Pretpostavimo da imamo skup od n objekata, i da i-ti objekat ima veličinu s_i, tako da važi $0 < s_i < 1$. Želimo da smestimo sve objekte u najmanji mogući broj kutija kapaciteta 1. Svaka kutija može držati arbitrarno mnogo objekata, dokle god njihova ukupna veličina ne premašuje 1.

(*i*) Posmatrajmo algoritam koji razmatra objekte redom i svaki smešta u **nasumičnu kutiju koja može da ga prihvati** (stvarajući novu kutiju ukoliko ni jedna ne može). Implementirajte ovaj algoritam u jeziku po vašem izboru, *što je efikasnije moguće*, i analizirajte njegovu vremensku složenost. [*Hint:* Algoritam ne mora da bude *uniformno* nasumičan pri biranju kutije, ali treba da postoji neka (ne-nula) verovatnoća da će odabrati svaku od mogućih kutija.]

(*ii*) Ovaj algoritam ne nalazi uvek optimalno rešenje—priložite kontraprimer (sa detaljnim objašnjenjem) koji ovo pokazuje.

(*iii*) Odredite **što strožiju moguću** stopu približnosti za ovaj algoritam, i priložite dokaz za ovu stopu. [*Hint:* Izrazite donje ili gornje granice za optimalan broj kutija, C^*, i broj kutija koje koristi algoritam, C, preko ukupne veličine svih objekata, $S = \sum_{i=1}^{n} s_i$.]

(*c*) Problem **putujućeg trgovca** (*Travelling Salesman Problem/TSP*) podrazumeva nalaženje ciklusa u datom neusmerenom težinskom grafu koji posećuje svaki od njegovih čvorova tačno jednom, i ima minimalnu ukupnu težinu. Ovo je jedan od najpoznatijih *NP-teških* problema. Međutim, ovaj problem je težak i sa stanovišta *približnih rešenja*: za bilo koju konstantu ρ, postojanje polinomnog TSP algoritma sa stopom približnosti ρ bi značio $P = NP$! Priložite dokaz ove tvrdnje. [*Hint:* Kako bi nam ovakav algoritam pomogao za nalaženje Hamiltonovih ciklusa?]

6 Slot mašine

Našli ste se u zapuštenoj prostoriji u kojoj se nalazi $N = 26$ slot mašina, svaka označena sa jednim velikim slovom engleske abecede (`'A'`, `'B'`, ..., `'Z'`). Prilikom povlačenja ručica jedne od mašina, dobijate nazad neku *nagradu*, $r \in \mathbb{R}$. Ova nagrada može biti i *negativna*, i neće neophodno uvek biti ista za istu mašinu! Svaka od ovih mašina poseduje različite karaktersitike, i jedini način na koji saznajete informacije o mašinama je povlačenjem njihovih ručica, tj. kroz skupljene nagrade.

U narednih nekoliko podzadataka, na vama će biti da smislite strategiju povlačenja ručica tako da se maksimizira ukupna osvojena nagrada posle M povlačenja (gde je M "dovoljno" veliki broj, koji vam nije poznat). Da biste ovo postigli, na raspolaganju vam je C++ fajl, `slots.cpp`, koji možete preuzeti sa sledeće adrese: `http://csnedelja.mg.edu.rs/static/resources/slots.cpp`.

Unutar fajla `slots.cpp` nalazi se kompletna logika kojom se inicijalizuju mašine, i prati strategija koju ste definisali određen broj koraka (na kraju se prijavljuje ukupna osvojena nagrada). Ovaj fajl će biti korišćen i za testiranje vaših strategija (naravno, sa parametrima mašina koji vam neće biti poznati).

Potrebno je prikladno izmeniti implementaciju klase `Player`:

- U ovoj klasi možete deklarisati, inicijalizovati i menjati sve potrebne promenljive—**ne koristiti globalne promenljive**! Specijalno, u priloženom fajlu vam je data promenljiva `name`—obavezno u konstruktoru (`Player()`) postaviti ovu promenljivu na vaše ime i prezime!

- Metoda `pull()` treba da vrati jedno veliko slovo engleske abecede (između `'A'` i `'Z'`), koje predstavlja mašinu čiju ručicu trenutno želite da povučete.

- Metoda `observe(machine, reward)` se poziva nakon svakog povlačenja ručice. U promenljivoj `machine` nalazi se slovo koje predstavlja mašinu koja je odabrana (tj. slovo koje je prethodno odabrala `pull` metoda), dok se u promenljivoj `reward` nalazi osvojena nagrada. Ovu metodu možete iskoristiti da prikladno ažurirate vaše promenljive, na osnovu očitane nagrade.

Kao vaše rešenje, potrebno je predati `.zip` arhivu koja u sebi sadrži prikladno izmenjen `slots.cpp` fajl (sa implementiranom vašom strategijom), kao i `.txt/.pdf` fajl koji sadrži detaljno objašnjenje vaše strategije.

N.B. Potrebno je predati samo *jednu* strategiju!

Ovo je ***izuzetno* važno**: prikladno objašnjenje vaše strategije donosi 50% bodova u ovom zadatku, dok se ostatak bodova odlučuje na osnovu uspešnosti vaše strategije na sledećim podzadacima:

(*a*) Postoji tačno jedna ručica koja daje konstantnu, pozitivnu nagradu—sve ostale daju pretežno negativne (nasumične) nagrade. Od vaše strategije se očekuje da, nakon velikog broja povlačenja ručica, ima ukupnu nagradu koja je *pozitivna*.

(*b*) Postoji tačno jedna ručica koja daje pretežno pozitivne (nasumične) nagrade—sve ostale daju pretežno negativne (nasumične) nagrade. Od vaše strategije se očekuje da, nakon velikog broja povlačenja ručica, ima ukupnu nagradu koja je *pozitivna*.

(*c*) 13 ručica daju pretežno pozitivne (nasumične) nagrade, dok preostalih 13 daju pretežno negativne (nasumične) nagrade. Međutim, pretežno negativne ručice imaju daleko nesigurniju prosečnu nagradu (tako da je moguće vrlo lako zbuniti se da su pretežno pozitivne)! Od vaše strategije se očekuje da, nakon velikog broja povlačenja ručica, ima ukupnu nagradu koja je *pozitivna*.

(*d*) Ne postoje nikakva ograničenja na svojstva ručica—vaš zadatak je da osvojite što veću nagradu, nakon velikog broja povlačenja ručica. Bodovi na ovom podzadatku će se odrediti na osnovu uspešnosti vašeg rešenja u odnosu na nekoliko naših strategija, kao i strategija ostalih kandidata.

KRAJ TESTA