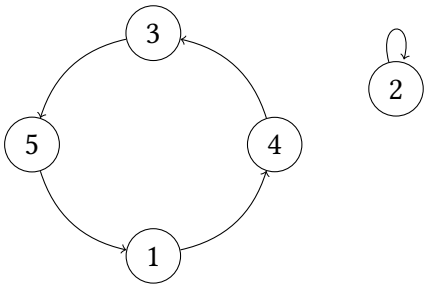
— *Rešenja* —

1 NI-grafovi (Andrej Ivašković)

Posmatrajmo jednu karakterističnu klasu *usmerenih grafova*: čvorovi su dati skupom $[n] = \{1, 2, \ldots, n\}$ i svaki čvor ima ulazni stepen 1 i izlazni stepen 1, odnosno za svaki čvor u postoji tačno jedna grana sa početkom u u i tačno jedna grana sa krajem u u, pri čemu je grana (u, u) moguća. U ovom zadatku ćemo ih zvati **NI-grafovima**. Primer jednog NI-grafa je dat na slici.



NI-grafovi su veoma značajni u računarstvu i neka njihova svojstva su u tesnoj vezi sa nedavnim rezultatima o *izomorfizmu grafova*, kao i u oblasti kriptografije.

Jedno značajno svojstvo NI-grafova je *periodičnost*: čvor u u NI-grafu G ima **vreme povratka** t ukoliko postoji put dužine t od u do u u G. Primetimo da ova definicija dozvoljava da čvor ima *više mogućih* vremena povratka. Definišimo da je NI-graf G ***k*-periodičan** ukoliko svi čvorovi u G imaju vreme povratka k. Tako, na primer, NI-graf sa slike je 4-periodičan.

U primenama ove teorije nas često zanima periodičnost nekog NI-grafa. Tema ovog zadatka je k-periodičnost kada je k *prost broj*.

(a) Odrediti broj NI-grafova koji mogu da se formiraju nad skupom čvorova $[n]$.

Odgovor:

Ovakvih grafova ima $n! = 1 \cdot 2 \cdot 3 \cdot \ldots \cdot n$. Ovo može da se pokaže na više načina. Ovde predstavljamo tri.

(i) Ovaj graf ima n grana i one se daju predstaviti kao uređeni parovi $(1, x_1)$, $(2, x_2)$, $\ldots$, (n, x_n) jer je izlazni stepen svih čvorova jednak 1. Pošto je ulazni stepen 1, svaki od brojeva u $[n]$ mora da se javi tačno jednom kao druga koordinata ovih uređenih parova. Dakle, ukupan broj ovih grafova je jednak broju načina da se elementi $[n]$ "poređaju" (prvi u redu odgovara x_1, drugi x_2 itd), a ovo je broj permutacija skupa $[n]$. Prema tome, odgovor je $n!$.

(ii) Ovaj graf predstavlja neku funkciju σ na skupu $[n]$, gde prisustvo (u, v) znači $\sigma(u) = v$. Ovo je omogućeno zbog toga što je izlazni stepen 1. Kako je ulazni stepen 1, to je za svako x jedinstveno definisano neko y takvo da je $\sigma(y) = x$. Ovo znači da σ ima inverz, pa je samim tim i bijekcija. Bijekcije na konačnom skupu su permutacije, a njih u ovom slučaju ima $n!$.

(iii) Neka je a_n broj NI-grafova nad $[n]$. Trivijalno važi $a_1 = 1$. Kada je $n \geqslant 2$, posmatramo element 1. NI-grafova takvih da je $(1, 1)$ grana ima a_{n-1} jer smo se samo "prebacili" na manji problem. Ako $(1, 1)$ nije grana, neka $(1, x)$ jeste, gde x može da se odabere na

$n-1$ način. Za svaki od ovih načina posmatramo načine da se formira NI-graf od svih čvorova sem čvora 1, pri čemu "novu" ulaznu granu za čvor x, neku (y,x), zamenjujemo granom $(y,1)$. Time se dobije rekurentna formula $a_n = a_{n-1} + (n-1)\,a_{n-1} = na_{n-1}$. Sledi $a_n = n!$.

Ovo pokazuje važnu vezu NI-grafova sa permutacijama.

(b) Odrediti eksplicitnu formulu za $T_k(n)$, broj k-periodičnih NI-grafova sa skupom čvorova $[n]$, gde je k prost broj. Odgovor dati u vidu iteracije (sume ili proizvoda).

Odgovor:

Ovde ćemo izvesti $T_k(n)$ na samo jedan način (drugi su takođe mogući).

Prvo primetimo da se NI-grafovi sastoje od *ciklusa* i dužine tih ciklusa određuju periodičnost NI-grafa. Ako je NI-graf k-periodičan, dužine svih ciklusa moraju da budu delioci k. Kada je k prost broj, jedine mogućnosti su 1 i k.

Napisaćemo odgovor u vidu sume čiji će brojač i predstavljati broj ciklusa dužine k (maksimalno $\lfloor n/k \rfloor$, gde je $\lfloor\cdot\rfloor$ označava donji ceo deo). Ovo znači da će $n-ki$ elemenata biti u ciklusima dužine 1, a ki elemenata u ciklusima dužine k. Broj načina da se odabere ki elemenata od n je jednak $\binom{n}{ki} = n!/\left((ki!)\,(n-ki)!\right)$. Da bi se ovih ki rasporedili u i podskupova od po k elemenata, najpre biramo koji elementi su u prvom podskupu na $\binom{ki}{k}$ načina, potom elemente drugog biramo na $\binom{ki-k}{k}$ načina itd, s tim što nam "redosled" podskupova nije bitan, te delimo rezultat sa $i!$. Na kraju, ovo treba da se pomnoži sa $(k-1)!$, i to i puta: postoji $(k-1)!$ načina da se napravi ciklus u svakom od i podskupova. Dakle:

$$T_k(n) = \sum_{i=0}^{\lfloor n/k \rfloor} \frac{1}{i!}\binom{n}{ki}\binom{ki}{k}\binom{k\,(i-1)}{k}\ldots\binom{k}{k}(k-1)!^k$$

Međutim, ovaj proizvod se (na osnovu definicije faktorijela i binomnog koeficijenta) lako uprosti:

$$T_k(n) = \sum_{i=0}^{\lfloor n/k \rfloor} \frac{n!}{(n-ki)!i!k^i}$$

(c) Odrediti vrednosti $T_2(n)/n!$, $T_3(n)/n!$, $T_5(n)/n!$ za sve prirodne brojeve n ne veće od 100. Skicirati grafike ovih funkcija za $1 \leqslant n \leqslant 100$. [*Hint:* Izraz u prethodnom delu nije numerički pogodan za ovaj račun, neophodno je naći adekvatne *rekurentne formule*.]

Odgovor:

Najpre odredimo rekurentnu formulu za $T_2(n)$. Početne vrednosti su $T_2(0) = T_2(1) = 1$, a rekurentna veza je $T_2(n) = T_2(n-1) + (n-1)\,T_2(n-2)$. Rezon je naredni: element 1 se slika ili u sebe (u tom slučaju rešavamo ekvivalentan problem "veličine" $n-1$) ili u jedan od preostalih $n-1$ (nakon čega imamo $T_2(n-2)$ načina za postizanje 2-periodičnosti). Ako je $x_n = T_2(n)/n!$, tada se rekurentna veza može zapisati u obliku $x_n = \frac{1}{n}(x_{n-1} + x_{n-2})$. Slične jednostavne veze važe i za ostale nizove.

(d) Odrediti neke dve funkcije $d(n)$ i $g(n)$ takve da za sve prirodne brojeve n važi:

$$d(n) \leqslant T_3(n) \leqslant g(n)$$

Dokazati dobijeno tvrđenje. Kvalitet rešenja će zavisiti i od kvaliteta procene i od tačnosti dokaza (trivijalne granice se neće vrednovati). U izrazima za $d(n)$ i $g(n)$ ne smeju da se nađu iteracije (sume, proizvodi) i dozvoljene su isključivo elementarne funkcije i $\lceil\cdot\rceil$, $\lfloor\cdot\rfloor$ (gornji i donji ceo deo).

Odgovor:

Ovaj deo zahteva kreativnost. Chowla, Hernstein, Scott dokazuju u *The solutions of* $x^d = 1$ *in symmetric groups* (1952) da je $T_3(n) \sim (n/e)^{2n/3} e^{\sqrt[3]{n}}/\sqrt{3}$, ali je izvođenje ovog rezultata jako komplikovano. Ovde se očekuje dokaz primenom matematičke indukcije, prevashodno zbog formule koja je izvedena u prethodnom delu. Dovoljno dobre granice su $n^{\lceil 2n/3\rceil}$ i $n^{\lfloor n/3\rfloor}$, tako da ove, slične ili bolje granice donose sve poene.

(e) Kako bi se računalo $T_k(n)$ ako bi k bio složen broj? Nije neophodno dati eksplicitan izraz, kratko objašnjenje je dovoljno.

Odgovor:

Jasno je da ovde moramo da pokrijemo sve delioce broja k na neki način. Jedna mogućnost je da se suma iz (a) zameni nekom u kojoj se sumira po $i_1, i_2, \ldots, i_t$, gde je $t = \tau(k)$ broj delilaca k i ti delioci su $1 = d_1, d_2, \ldots, d_t = n$ i važi $i_1 d_1 + i_2 d_2 + \ldots + i_t d_t = n$, a izraz u sumi je sličan onom iz (b). Ne očekuje se detaljniji odgovor od ovog.

Izvan onoga što se očekuje od odgovora, zanimljiva je naredna činjenica koja važi za sve $k \in \mathbb{N}$ i za $x \in (0,1)$:

$$\sum_{n=0}^{\infty} T_k(n) \frac{x^n}{n!} = \exp\left(\sum_{d|k} \frac{x^d}{d}\right)$$

uz dodatno definisanje $T_k(0) = 1$.

Ono što bi trebalo da predate kao rešenje ovog zadatka je `.pdf` fajl u kom se nalaze rešenja podzadataka (zbog matematičke prirode zadatka se prihvataju i rukom ispisana rešenja dokle god su čitljiva!). U njemu bi trebalo da se nalaze i grafici i vrednosti funkcija iz (c). Matematički postupak bi trebalo da bude jasan, uz razumnu količinu detalja u obrazloženjima.

Za skiciranje grafika u (c) se preporučuje korišćenje programa poput MS Office Excel, Matlab, GNU Octave, gnuplot.

2 JDRUŠTVO (David Davidović)

JDRUŠTVO, zloglasna, misteriozna i ambiciozna hakerska grupa sa grandioznim motivima, zatražila je kontakt sa vama, sa očiglednim ciljem ugovaranja saradnje.

Ovaj zadatak je interaktivnog tipa. Od vas će se očekivati da pronađete bezbednosne rupe u sistemu (zle?) megakorporacije, kako biste ispunili svoje zaduženje i pomogli **JDRUŠTVU**.

Znanja koja očekujemo da posedujete su osnovi *web*-a. Sve ostalo što vam je neophodno za uspešan prelazak ovog zadatka je moguće pronaći na Internetu i primeniti uz malo pametnog razmišljanja.

Vaši uspesi u ovom zadatku se automatski beleže; ipak, kao rešenje je potrebno poslati u kratkim crtama opisane metode koje ste koristili i sve dodatne informacije koje objašnjavaju kako ste probili bezbednost sistema.

Kako biste započeli ovaj zadatak, morate ga prvo odabrati u procesu prijave. Ukoliko niste sigurni šta od zadataka želite da odaberete, a želite da se okušate u ovom zadatku, prijavite one koje mislite da ćete najverovatnije rešavati. Ako se predomislite, možete nam se javiti na kontakt email adresu kako bismo promenili odabrani spisak zadataka. Ako niste sigurni kako ćete se snaći, možete odabrati i dodatne zadatke kao "rezervu" u slučaju da niste zadovoljni svojim uspehom na zadatku **JDRUŠTVO**.

Nakon odabira zadatka na stranici za prijavu, posetite odmah link za slanje rešenja koji budete dobili. Na ovoj stranici naći ćete informacije o tome kako da započnete interaktivni izazov **JDRUŠTVO**.

U slučaju da ne znate kako dalje, pripremili smo za vas i određene nagoveštaje (*hints*) na raznim nivoima izazova. Korišćenje pomoći donosi određen broj negativnih poena, ali vas ohrabrujemo da zatražite pomoć ukoliko ocenjujete da sami nećete preći neki određen nivo. Da zatražite nagoveštaj, pošaljite mejl na `jdrustvo@csnedelja.mg.edu.rs`.

Za dodatne poene možete doći (u okviru zadatka) do *zastavice (flag)* skrivene na nekom mestu. Zastavica je *šesnaestocifreni broj* koji nam morate poslati kako bismo potvrdili da ste zaista došli do ovog nivoa i uračunali vam dodatne poene.

Srećno.

Odgovor: Rešenje ovog zadatka biće izloženo u delovima, od kojih svaki deo (osim prvog) donosi određenu količinu bodova. Za tražene nagoveštaje oduzimano je 20% od osvojenih bodova.

(*a*) **Uvod u zadatak.** Nakon što kliknete na dugme *Započni izazov*, prikazuje se simulirani *IRC chat* sa osobom koja vas kontaktira radi saradnje u napadu na korporaciju **Z Biotech**, zli konglomerat koji se bavi biotehnologijom. Naime, **Z Biotech** je skoro bio upleten u skandal u vezi sa izlivom toksičnih materija u Beogradu, u kome su poginuli nedužni ljudi.

Organizacija **JDRUŠTVO** traži od vas da u sistemu za upravljanje dokumentima ove kompanije dođete do poverljivih, internih informacija o ovom skandalu, kao i da učinite neku štetu kompaniji (ako je to moguće). U samom tekstu zadatka ste obavešteni o činjenici da je negde sakrivena i *flag* vrednost, šesnaestocifreni broj koji vam, ako ga priložite uz rešenje, donosi dodatne poene.

Tokom razgovora, vaš kontakt vam daje na raspolaganje *employee ID* i lozinku jednog zaposlenog u kompaniji **Z Biotech**.

(b) **Prvi koraci.** Nakon što se ulogujete na adresi `http://zbiotech.xyz`, prikazuje vam se interfejs *Z Biotech Enterprise Document Management* (ZEDM) sistema. Međutim, takođe dobijate i informaciju o tome da je u toku migracija na novu, savremenu verziju, a da za to vreme možete nastaviti da pristupate staroj, *legacy* verziji. Posetom linka dolazite do sajta `http://cs30.zbiotech.xyz/zedm1/legacy`.

Korišćenjem pristupnog kôda koji vam je prikazan na prvom ZEDM sajtu, možete se ulogovati i na starom sistemu. Međutim, i ovde postoje problemi: naime, migracija je u punom jeku i dokumenti vam u tom trenutku nisu dostupni. Takođe, nije vam dozvoljeno ni da otpremate svoje dokumente (na stranici *Upload*).

Pregledom sajta dolazi se do URL-a `http://cs30.zbiotech.xyz/zedm1/legacy/info.php?id=66`, koji prikazuje informacije o vama (trenutnom korisniku). Ukoliko pokušate da promenite vrednost GET parametra `id`, moguće je doći do interesantnog zaključka: možemo izazvati da sajt vrati poruku o grešci sa MySQL bazom, umesto pravog odgovora.

Ovo je siguran znak da je GET parametar koji menjamo ubačen u SQL upit takav kakav je, bez ikakvih provera. Konkretan SQL upit je

```
SELECT * FROM users WHERE id = <parametar> ORDER BY access_code
```

gde je `<parametar>` upravo tačna vrednost parametra, onakvog kakvog ste ga uneli.

(c) **Ubrizgavanje SQL-a.** Čitalac koji je upoznat sa konceptom *SQL injectiona* će u ovom odmah prepoznati priliku: ukoliko možemo da u potpunosti kontrolišemo deo upita koji se izvršava na bazi, vrlo lako možemo da učinimo da taj upit radi nešto sasvim drugo. Na primer, ako stavimo da `id` parametar bude `66 OR 1=1 --`, upit će izgledati ovako:

```
SELECT * FROM USERS WHERE id = 66 OR 1=1 -- ORDER BY access_code
```

Dakle, ovaj upit, umesto da vrati podatke o jednom korisniku, vratiće podatke o *svim* korisnicima, zbog dodatka `OR 1=1` koji čini da `WHERE` uslov uvek bude tačan. Ipak, ako probate ovo, sajt će vam i dalje dati informacije o jednom korisniku, i to prvom u bazi. Primetimo, takođe, da su dve crtice (`--`) u SQL-u oznake za komentar, te smo na vrlo jednostavan način eliminisali ostatak upita (`ORDER BY access_code`), koji bi nam inače možda smetao.

Ovo se dešava jer je sâm sajt napisan tako da vrati podatke o jednom korisniku. Ukoliko želimo da iz baze "izvučeno" neke druge korisne podatke, moramo malo poboljšati naš pristup. Na osnovu prisustva `access_code` kolone, čiji pomen možemo videti u porukama o grešci koje sajt ispisuje ako kao `id` parametar unesemo neto to neće napraviti validan SQL upit, možemo zaključiti da se u bazi drže sirove (*plaintext*) verzije ovih pristupnih šifri, i da bi nam bilo korisno da pronađemo pristupnu šifru nekog korisnika sa većim privilegijama. A ko je bolji kandidat za to od korisnika sa `id`-em 0, gospodina Terija Kolbija (*Terry Colby*), koji je takođe *administrator sistema*?

Čitalac koji se malo više informisao o *SQL injection* napadima, ili ima prethodnog iskustva sa njima, shvatiće da nam ovde može pomoći ključna reč `UNION` u SQL-u, koja omogućava spajanje rezultata više upita. Na primer, ukoliko manipulacijom `id` parametra dobijemo uniju upita koji *ne vraća ništa* sa nekim drugim upitom, rezultat će biti rezultat samo drugog upita! Ovaj drugi upit onda može odabrati bilo šta iz tabele `users`, pa i same pristupne kodove za bilo kog korisnika.

Ovo nas, još uvek, ne dovodi do konačnog rešenja; ukoliko za parametar `id` stavimo `-9999 UNION SELECT access_code FROM users WHERE id = 0 --`, dobijamo sledeću poruku o grešci:

```
The used SELECT statements have a different number of columns
```

Naime, dodatnim ispitivanjem SQL dokumentacije moguće je zaključiti da, kako bi se napravila unija rezultata dva upita, oba moraju vratiti *isti broj kolona*. Na sreću, SQL nam dozvoljava da dodamo koliko god je potrebno praznih kolona u upit, čime možemo postepeno dodavati nove *"dummy"* kolone dok ne dobijemo željeni izlaz. Redom, `id` parametri koje isprobavamo postaju:

(*i*)
```
-9999 UNION SELECT access_code, NULL
FROM users WHERE id = 0 --
```

(*ii*)
```
-9999 UNION SELECT access_code, NULL, NULL
FROM users WHERE id = 0 --
```

(*iii*)
```
-9999 UNION SELECT access_code, NULL, NULL, NULL
FROM users WHERE id = 0 --
```

(*iv*)
```
-9999 UNION SELECT access_code, NULL, NULL, NULL, NULL
FROM users WHERE id = 0 --
```

Konačno, poslednji isprobani upit ima korektan broj kolona (pet), nakon čega će jedno od polja izlaza prikazivati pristupnu šifru korisnika Terija Kolbija (sa ID-em 0), koji je administrator sistema. Sada možemo da se ulogujemo u sistem kao on.

(*d*) **Dokumenti.** Nakon što se ulogujemo kao Teri Kolbi, stvari su drugačije. Možemo videti sve dokumente, a ako u polje *Search* na stranici *Browse* unesemo *Belgrade*, dobićemo samo jedan dokument, čiji naziv izgleda veoma relevantno: *Class action toxic waste leak suit in Belgrade risk assessment report*, sa ID brojem `GZLPYO-851446`. Međutim, iako smo identifikovali traženi dokument, ne možemo ga direktno *preuzeti*—u koloni *View*, umesto linka za pregled ili preuzimanje, stoji samo `false`.

Za dalji napredak na ovom zadatku, korisno je uvideti da neki dokumenti imaju potpuno prazno *View* polje, što ukazuje da se nešto čudno dešava u samom *HTML izvornom kodu stranice*. Ukoliko ga pogledamo, možemo jasno videti i zašto: i ovi dokumenti imaju link u *View* polju, samo što je on *pod komentarom*. Ključna opservacija je da svaki link ima oblik

```
http://cs30.zbiotech.xyz/zedm1/legacy/docs/XYZ/<id-dokumenta>.zed
```

gde su `XYZ` tri cifre. Ovo implicira da je link ka traženom dokumentu najverovatnije oblika

```
http://cs30.zbiotech.xyz/zedm1/legacy/docs/XYZ/GZLPYO-851446.zed
```

Međutim, i dalje ne znamo tačne cifre `X`, `Y` i `Z`, budući da su one *različite* za svaki dokument.

(*e*) **Sirova snaga.** Iako je moguće smisliti mnogo teorija o tome kako se biraju tražene cifre `X`, `Y` i `Z` za svaki dokument (pod uslovom da pravilo *uopšte postoji*!), daleko lakše rešenje podrazumeva pisanje programa koji *isprobava svaku moguću trocifrenu kombinaciju* (kojih sveukupno ima samo 1000). Postoji mnogo načina da se ovo uradi, i moguće je u gotovo svakom programskom jeziku. Jedino o čemu treba voditi računa je proces prijavljivanja na sistem pomoću pristupne šifre—*kako da naš program to uradi*?

Ispostavlja se da nakon prijavljivanja setuje kolačić (*cookie*) na vašem pretraživaču, koji prati stanje vaše prijave na sistemu. Lako možemo doći do ovog kolačića (npr. korišćenjem *Developer Tools* alata u pretraživaču) i slati ga uz svaki pokušaj koji napravi naš program.

Pisanje samog programa ostavljamo kao vežbu čitaocu—uz napomenu da je, za najbrže rezultate, verovatno najbolje koristiti programski jezik *Python* i biblioteku `httplib`, uz čiju pomoć ovakav program broji svega desetak linija. Pri izvršenju programa, primetićemo da jedan link vraća različit rezultat od ostalih; taj link je

`http://cs30.zbiotech.xyz/zedm1/legacy/docs/924/GZLPYO-851446.zed`

i on nam omogućava da skinemo traženi dokument, čime je zadatak (*skoro!*) gotov.

(*f*) **Dodatni bodovi.** Dobijanje *flag*-a o kome je bilo reči u tekstu zadatka nije preterano teško, i zahteva samo osnove rada u *shell*-u. Ukoliko malo pažljivije pogledate izvorni kôd početne stranice (pri korišćenju naloga Terija Kolbija), možete videti zakomentarisano korisničko ime i lozinku za *SSH* pristup.

Posle zanemarljive količine dodatne pretrage moguće je preuzeti *SSH klijenta* (npr. *PuTTy* za *Windows* ili *OpenSSH* za Unix sisteme), pomoću kog je moguće povezati se sa serverom koristeći ovo korisničko ime i šifru, čime smo ubačeni u minimalnu *komandnu liniju*.

Ova komandna linija dozvoljava izvršenje samo nekolicine standardnih komandi. Jedna takva komanda je komanda `ls`, kojom dobijamo pristup svim fajlovima u trenutnom direktorijumu. Njenim pokretanjem dobijamo informaciju o prisustvu sledeća dva fajla:

- `fuxsocy.py`
- `jdrustvo00.dat`

Radoznali mogu da izvrše komandu `cat jdrustvo00.dat` (za ispisivanje sadržaja `.dat` fajla). Međutim, za dobijanje samog *flag*-a, potrebno je izvršiti program sadržan u fajlu `fuxsocy.py`. Ovo se može uraditi na dva načina: `python fuxsocy.py` ili `./fuxsocy.py`.

Nakon što se ovaj program izvrši i učini podatke na serveru potpuno *enkriptovanim* (i samim tim *nečitljivim*), dobijate *flag*. Čestitamo svima koji su došli dovde (ukupno **pet** kandidata)! Ukoliko niste, ne osećajte se loše—ovo su samo bonus poeni: postoje i važnije stvari. :)

NEDELJA INFORMATIKE v3.0 – 1617 – Paper 0

3 Izgubljeni u d3k0d1r4nju (Lazar Mitrović/Nikola Jovanović)

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

Odgovor:

Ovaj zadatak ima neobičan format: umesto teksta data je samo `.zip` arhiva sa tri audio fajla: `tekst.wav`, `2.wav` i `3.wav`. Cilj zadatka je, kao što se odmah da naslutiti, dekodiranje ovih audio fajlova, a njihova imena nam ukazuju na redosled. Svi kodovi napisani za potrebe kreiranja ovog zadatka (enkoderi i dekoderi) kao i svi tekstovi dobijeni dekodiranjem (budući da su u ovom rešenju prikazani samo delovi) su javno dostupni[1].

(a) `tekst.wav`
Prvi audio fajl sadrži tekst zadatka. Očekuje se da kandidati umeju da prepoznaju (inicijalno ili nakon što uspore zapis u nekom programu za obradu zvuka) da je metoda korišćena za enkodiranje **Morzeov kod**[2], i da umeju da napišu dekoder koji će otkriti tekst zadatka (rešenja zasnovana na dobrom sluhu su bodovana polovično). Pri enkodiranju je korišćena samo jedna frekvencija (300 Hz) pa je, kako bi se došlo do rešenja, neophodno podeliti audio zapis na intervale tišine i intervale u kojima se emituje zvuk. Dalje, svakom intervalu možemo na osnovu njegove dužine i prisustva zvuka dodeliti jedan od pet tokena: {dit(tačka), dah(crta), razmak između simbola, razmak između slova, razmak između reči}. Tačan odnos dužina intervala korišćen pri enkodiranju je 1:3:1:3:7, što se može otkriti kraćom analizom. Dobijeni niz tokena se jednoznačno može prevesti u tekst korišćenjem Morzeove azbuke.
Rešenje:

```
STX U VREME...SRECNO ETX EOF
```

(b) `2.wav`
Drugi audio fajl sadrži niz bitova moduliran **BFSK**[3] metodom, nastao mapiranjem karaktera u tekstu u njihove ASCII vrednosti u binarnom sistemu sa širinom 8 bita. U ovom slučaju je svakako teže pogoditi metodu kodiranja, ali su sve neophodne informacije date u tekstu iz prethodnog dela zadatka. Prvi deo rešenja se sastoji u pretvaranju niza audio frejmova u niz frekvencija. Neke naivne metode koje mogu da to reše su računanje talasne dužine na osnovu razmaka između dva maksimuma/minimuma amplitude ili između npr. dva preseka sa nultom amplitudom. Dobijeni niz frekvencija će, zbog nesavršenosti korišćenih metoda, sadržati više od samo dve frekvencije. Pri enkodiranju korišćene su frekvencije 300Hz (0) i 1kHz (1), što kandidati naravno ne znaju, pa se očekuje da do te informacije dođu naprednom analizom zvuka (Brza Furijeova transformacija (FFT)), plotovanjem dobijenih frekvencija, ili da prosto bez te informacije programski podele sve frekvencije u dve grupe: "niske" (0) i "visoke" (1). Najprostiji način za ovo sastoji se u biranju *threshold*-a tj. granične vrednost iznad koje sve frekvencije bivaju tretirane kao binarna jedinica, a jedan od načina za odabir ispravnog thresholda je prosek svih frekvencija. Ovo se oslanja na pretpostavku da će u binarnom zapisu teksta biti približno jednak broj nula i jedinica, što nije najispravnija pretpostavka, ali je dovoljno dobra da se frekvencije pretvore u bitove. Nakon toga sledi grupisanje u blokove od po 8 i prosto pretvaranje binarnih blokova u karaktere.
Rešenje:

```
#Rand0m joined as R
...
EOT
```

[1] `https://github.com/mgcsweek/such-wav-very-wow`
[2] `https://en.wikipedia.org/wiki/Morse_code`
[3] `https://en.wikipedia.org/wiki/Frequency-shift_keying`

(*c*) `3.wav`

Treći i poslednji audio fajl sadrži, kao što upravo otkriveni tekst sugeriše, sliku enkodiranu korišćenjem **SSTV**[4] metode. Osnovni pristup je sličan kao u prethodnom delu zadatka: izdvojimo frejmove i identifikujemo postojeće frekvencije. U ovom slučaju, značenje frekvencija je određeno SSTV standardom čiji su ključni delovi opisani u rešenju prethodnog dela zadatka (kompletan opis standarda može se naći u odgovarajućem kodu u repozitorijumu datom na početku ovog rešenja). Pored toga, neophodno je istražiti informacije zajedničke za sve SSTV standarde (uglavnom konkretna trajanja i frekvencije određenih signala). Kada imamo sve potrebne informacije, nakon dodatne obrade dobijamo vrednosti piksela i otkrivamo da je enkodirana slika zapravo QR kod.

Slika 1: **Levo:** originalna slika QR koda. **Desno:** Slika dobijena iz priloženog audio fajla korišćenjem dekodera autora zadatka.

Skeniranjem QR koda dobijamo link ka `pastebin` stranici na kojoj se nalazi sledeći tekst:

```
{
 "greeting": "Hello friend!",
 "mailto": "tyrellwellick59@gmail.com",
 "subject": "IZGUBLJENI U D3K0D1RANJU",
 "body": "whiter0se_stage2"
}
```

Ovo predstavlja sam kraj zadatka. Jedino što je preostalo je poslati e-mail na datu adresu sa datim sadržajem.

Na veliku radost autora zadatka, stigla su **četiri** ispravna mejla. Najbržem rešavaocu (za koga je spremna obećana nagrada) je trebalo 9 dana, a preostala 3 kandidata su poslala mejl nakon 11, 16, odnosno 17 dana.

Bodovanje

Isti princip bodovanja je korišćen za sve delove zadatka. Neki opšti principi kojima su se pregledači vodili:

- Kandidati koji uspešno dolaze do rešenja ali uz korišćenje gotovih softverskih alata ne osvajaju poene. Ovo je izričito naglašeno u tekstu zadatka koji stoji iza fajla `tekst.wav` pa i takmičari koji do teksta dođu na ovaj način imaju šansu da to isprave i dobiju poene na prvom delu.
- Nezavisno od metode i uspeha u rešavanju, korektne ideje i tekstualni opisi metoda nagrađivani su minimalnim brojem poena.
- Svi funkcionalni programi koji uspešno dekodiraju konkretan audio fajl uz izvorni kod i kratko objašnjenje nagrađivani su maksimalnim brojem poena. Na ovo ne utiče izbor programskog jezika, količina „ružnih hakova" i broj eksternih biblioteka koje su korišćene. Ni u jednom delu zadatka nije bilo neophodno napisati „savršen" dekoder i od kandidata se suštinski očekivalo da u ovom zadatku prepoznaju probleme sa svojim rešenjima, koja uvek nastaju kako zbog sopstvenih grešaka tako i zbog nesavršenosti digitalnih audio zapisa i samih enkodera, i samostalno dođu do ideja za prevazilaženje tih problema.
- Nesavršena rešenja koja ne vrše dekodiranje u potpunosti pravilno ili imaju očigledne nedostatke u okviru zadatka su penalizovana, doduše malim brojem poena.

[4] `https://en.wikipedia.org/wiki/Slow-scan_television`

4 Tematski problemi (Nenad Bauk)

Mladi Nenad iz Atomske banje želi da napravi svoju internet enciklopediju, po uzoru na popularnu *Wikipediu*. Po njegovom modelu, drugi korisnici bi trebalo da šalju nove članke (na engleskom jeziku) na njegov server, nakon čega bi njegov softver automatski razvrstao tekstove u jednu od sledećih deset kategorija, sa sledećim indeksima:

0 — *Culture* (Kultura)
1 — *Geography* (Geografija)
2 — *Health* (Zdravlje)
3 — *History* (Istorija)
4 — *Mathematics* (Matematika)
5 — *Nature* (Priroda)
6 — *Philosophy* (Filozofija)
7 — *Religion* (Religija)
8 — *Society* (Društvo)
9 — *Technology* (Tehnologija)

Zamolio vas je za pomoć pri implementaciji ovog sistema. Potrebno je da implementirate program u jeziku po vašem izboru, koji će preko standardnog ulaza učitati ime fajla u kome se nalazi članak (napisan na engleskom jeziku), a na standardni izlaz ispisati jedan ceo broj između 0 i 9, koji označava kategoriju unetog članka. Radi ilustracije, na raspolaganju su vam po tri primerka članaka iz svake od deset kategorija, na sledećem linku:
`http://csnedelja.mg.edu.rs/static/resources/tematski-problemi.zip`.

Vaše rešenje je potrebno predati kao `.zip` arhivu koja sadrži kôd i, ukoliko je potrebno, sve neophodne instrukcije za njegovo pokretanje, kao i `.pdf/.txt` fajl koji sadrži detaljan opis vašeg algoritma. Vaši programi će prevashodno biti ocenjivani na osnovu njihove tačnosti na (tajnom) probnom skupu članaka, ali je moguće osvojiti dosta poena na korektan i dobro obrazložen pristup, čak i ukoliko ne bude dovoljno efikasan.

Odgovor: U ovom zadatku se mogu izdvojiti *tri* osnovne kategorije pristupa:

- Sistemi zasnovani samo na *pravilima*, dakle potpuno program-driven pristupi. Od ovakvih pristupa se očekuje najslabiji učinak, osim ukoliko su izrazito složeni;
- Sistemi zasnovani na jednostavnim metrikama nad podacima, npr. *bag-of-words*, *Naïve Bayes* ili *bigram* modeli. Od ovakvih pristupa se očekuje razumna tačnost, u zavisnosti od količine treniranja;
- Sistemi zasnovani na učenju apstraktnih svojstava podataka; npr. PCA/LDA/deep learning. Ovo, za razliku od prethodne stavke, zahteva dodatno čitanje i treba dobro vrednovati čak i ukoliko ne donese najpovoljnije rezultate (iako ima najveći potencijal).

Osim ocenjivanja osnovnog pristupa, posebnu pažnju obratiti na prisustvo *augmentacije podataka*, tj. da li su kandidati pokušali da samostalno preuzmu članke sa interneta radi dodatnog treninga. Ovo treba značajno da razdvoji prosečne od jako kvalitetnih kandidata.

---

5 Uzastopni podnizovi i Acko kamiondžija (Nikola Nedeljković)

Zadatke rešiti u što boljoj vremenskoj i memorijskoj složenosti. Što detaljnije i jasnije izložiti metodu rešavanja i diskutovati složenost rešenja. U ovom zadatku nije neophodna implementacija, pa rešenja treba da budu u obliku `.pdf` ili `.txt` fajlova.

(*a*) Definišimo alternirajuću sumu niza $a[x..y]$ kao $S[a[x..y]] = a[x] - a[x+1] + a[x+2] + ... + (-1)^{(y-x)}a[y]$. Za niz $a[1..N]$ potrebno je naći maksimalnu alternirajuću sumu svih uzastopnih podnizova **dužine K**. Npr. za niz $a[1..5] = [2, 3, -1, 4, -3]$ uzastopni podnizovi dužine 3 su $[2, 3, -1]$, $[3, -1, 4]$, $[-1, 4, -3]$, a najveću alternirajuću sumu ima podniz $[3, -1, 4]$ i iznosi 8.

Odgovor: Označimo sa $S[x]$ alternirajuću sumu $a[x] - a[x+1] + ... + (-1)^{K-1}a[x+K-1]$. Primetimo da važi $S[x+1] = -S[x] + a[x] + (-1)^{K-1}a[x+K]$. Prvo računamo $S[1]$, a zatim u jednom prolazu kroz niz $a[1..N]$ u složenosti $O(N)$ koristeći prethodnu formulu nalazimo $S[i+1]$ pomoću $S[i]$, beležimo maksimum i nalazimo rešenje.

(*b*) Za niz $a[1..N]$ naći maksimalnu alternirajuću sumu svih uzastopnih podnizova **bilo koje** dužine. Npr. za niz $a[1..5] = [2, 3, -1, 4, -3]$ najveću alternirajuću sumu ima podniz $[3, -1, 4, -3]$ i iznosi 11.

Odgovor: Za rešavanje ovog zadatka koristimo dinamičko programiranje. Označimo sa $S_p[i]$ vrednost maksimalne alternirajuće sume parne dužine koja se završava u indeksu i, a sa $S_n[i]$ istu vrednost sume neparne dužine. Primetimo da je $S_n[1] = a[1]$, a $S_p[1]$ nedefinisano, dok je $S_p[2] = a[1] - a[2]$ i $S_n[2] = a[2]$. Počevši od indeksa 3, $S_p[i]$ računamo kao $S_n[i-1] - a[i]$, a $S_n[i]$ računamo kao $\max(S_p[i-1] + a[i], a[i])$. Na kraju prolazimo kroz nizove S_p i S_n i nalazimo maksimalnu vrednost. Složenost: $O(N)$.

(*c*) Za niz $a[1..N]$ naći uzastopni podniz koji ima minimalnu apsolutnu vrednost zbira. Npr. za niz $a[1..5] = [7, 4, -1, -6, 4]$ minimalnu apsolutnu vrednost zbira ima podniz $[-1]$, dok za niz $a[1..4] = [6, 3, -6, 2]$ minimalnu apsolutnu vrednost zbira ima podniz $[3, -6, 2]$ i iznosi $|3 + (-6) + 2| = |-1| = 1$.

Odgovor: Možemo izračunati prefiksne sume, tj. $S[1] = a[1], S[i] = S[i-1] + a[i]$, a zatim za svaki uzastopni podniz (npr. za $a[2..5]$ imamo $S[2..5] = S[5] - S[1]$ u $O(1)$) nađemo minimalnu apsolutnu vrednost zbira. Složenost ovog rešenja je onda broj podnizova, dakle $O(N^2)$. Ovo rešenje nosi 1p.

Primetimo da je svaki uzastopni podniz određen sa dve prefiksne sume, tako da ako bismo **sortirali** niz $S[i]$ prefiksnih suma (zajedno sa indeksima), dve najbliže prefiksne sume bi nam direktno odredile koji uzastopni podniz ima najmanju apsolutnu vrednost zbira i ona bi bila jednaka apsolutnoj razlici te dve prefiksne sume. Za nalaženje dve najbliže prefiksne sume dovoljan je jedan prolaz kroz već sortirani niz, tako da je složenost rešenja zapravo složenost sortiranja i iznosi $O(N \log N)$. Ovo rešenje donosi maksimalan broj poena na ovom zadatku.

(*d*) Acko je odlučio da poseti rejv u Kragujevcu. Takođe treba da preveze gomilu drugara iz Beograda, pa će ovog puta voziti kamion, dok će oni praviti mini-rejv u

prikolici. Na tom putu nalazi se **veliki** broj gradova i da pređe svaki put između neka dva direktno povezana grada Acku treba određen broj minuta. Nazovimo težinom puta između neka dva grada maksimalno vreme potrebno Acku da pređe između neka dva susedna grada na tom putu. Kako će Acko posetiti razne rejvove na putu do Kragujevca pomozite mu da nađe put najmanje težine iz Beograda za Kragujevac. Sprovesti dokaz korektnosti.

Odgovor: Traži se put iz grada A do grada B u grafu $G(V, E)$ koji ima minimalnu maksimalnu ivicu na tom putu. Za rešavanje ovog zadatka koristićemo modifikovanu varijantu *Dijkstrin*-og algoritma. Za svaki grad $v \in V$ održavaćemo minimalnu težinu puta iz grada A do grada v u nizu $d[]$. Održavaćemo dva skupa S - skup obrađenih gradova i skup $Q = V \backslash S$. Pri svakoj iteraciji algoritma u skup S prebacićemo grad $v \in Q$ koji ima najmanju vrednost $d[v]$. Zatim ćemo za sve susede $u \in Q$ grada v proveriti da li je put preko grada v bolji od dosadašnjeg kao $d[u] = \min(d[u], \max(d[v], e[v][u]))$, gde $e[v][u]$ predstavlja vreme potrebno za prelazak iz grada v u grad u. Na kraju Acku ćemo proslediti vrednost $d[B]$.

Pri prebacivanju grada v sa minimalnim $d[v]$ u skup S primetimo da $d[v]$ zaista predstavlja najmanju težinu puta do grada v, jer svi gradovi u koji su susedi grada v ili već pripadaju S a u tom slučaju je put preko u do v već ažuriran, ili pripadaju Q i samim tim je $d[u] \geq d[v]$, tj. postoji ivica veće ili jednake dužine na putu preko grada u nego ivica maksimalne dužine na već selektovanom putu. Složenost algoritma je jednaka složenosti *Dijkstrin*-og algoritma.

(*e*) Acko je poslednjih dana probudio *party*-čudovište u sebi, pa će vam postaviti Q različitih pitanja gde se od vas traži da nađete put najmanje težine između gradova A i B. Kako bi Acko posetio što više rejvova, na vama je da omogućite što brže odgovore na njegova pitanja. Za rešavanje ovog zadatka smatrati da je ukupan broj puteva između gradova srazmeran broju gradova ($E \sim cV$). Rešenja koja nalaze puteve najmanje težine za sve parove gradova nosiće manji broj poena.

Odgovor: Jedno od mogućih rešenja je puštanje *Dijkstrin*-og algoritma koji koristi prioritetni red za traženje minimuma i ažuriranje, iz svakog čvora grafa $G(V, E)$ u složenosti $O((|V| + |E|) \log |V|)$ i beleženje minimalne težine puta za svaki par čvorova u matrici. Kako je $E \sim cV$ ukupna vremenska složenost ovog algoritma je $O(|V|^2 \log |V|)$, dok je memorijska $O(|V|^2)$.

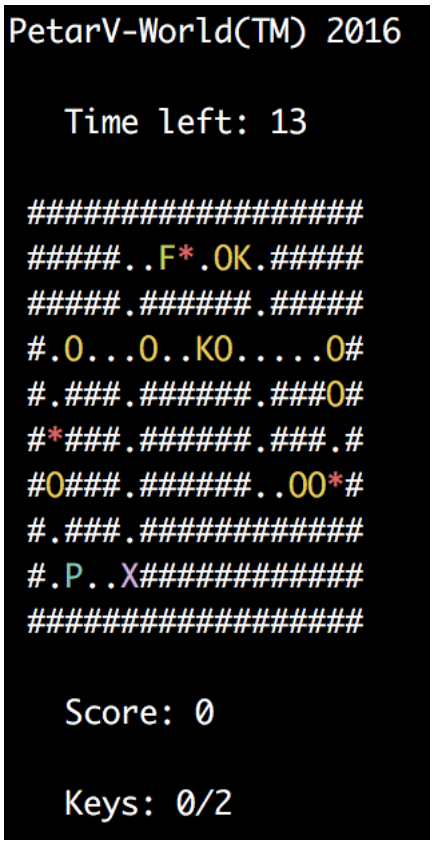
Zadatak se može rešiti i korišćenjem minimalnog razapinjajućeg stabla (u daljem tekstu MST - *minimum spanning tree*). Jedno zanimljivo svojstvo MST-a je da svaki (jedinstveni) put između dva čvora u i v u MST-u zapravo predstavlja put minimalne težine između ta dva čvora u grafu G nad kojim je izgrađen MST. MST možemo izgraditi korišćenjem *Kruskalov*-og algoritma koji radi u $O(|E| \log |V|)$, tj. kako je $E \sim cV$ u $O(|V| \log |V|)$. Zatim ako sačuvamo drvo preko lista susednosti , možemo pokrenuti BFS ili DFS u $O(V)$ iz svakog čvora (BFS radi u $O(V + E)$, ali kako je drvo u pitanju, $E = V - 1$), i beležiti u matrici vrednost do tada najveće grane u pretrazi do određenog čvora za konačno rešenje koje radi u vremenskoj i memorijskoj složenosti $O(V^2)$.

Složenost se može dodatno poboljšati korišćenjem LCA (*lowest common ancestor*) algoritma. LCA dva čvora u i v u root-ovanom drvetu je predak oba čvora koji je najviše udaljen od korena. Neka je $T[i]$ otac čvora i u drvetu. Sada možemo konstruisati niz $P[1..N][1..\log N]$ gde je $P[i][j]$ 2^j-i predak čvora i. Primetimo da ovo možemo uraditi rekurzivno kao $P[i][j] = T[i]$ za $j = 0$ i $P[i][j] = P[P[i][j-1]][j-1]$ za $j > 0$. Takođe u isto vreme možemo izračunati $G[i][j]$

koje predstavlja maksimalnu ivicu na putu između čvora i i $F[i][j]$. Ovo zauzima $O(|V| \log |V|)$ vremenske i memorijske složenosti. Sada korišćenjem LCA algoritma u $O(\log |V|)$ nalazimo najveću ivicu na putu između čvorova A i B za svaki upit Q. Ukupna vremenska složenost sada iznosi $O((|V|+|Q|) \log |V|)$ dok je memorijska $O(|V| \log |V|)$. Ovo rešenje nosi maksimalan broj poena. Detaljniji opis ovog LCA algoritma može se naći npr. na *TopCoder* LCA tutorijalu.

6 RAM Probing (Petar Veličković)

Pred vama je *PetarV-World*^{TM}, vrlo jednostavna igrica u kojoj je cilj da pomognete PetruV da izađe iz začaranog lavirinta. Kompletni materijali se mogu preuzeti sa sledećeg linka: `http://csnedelja.mg.edu.rs/static/resources/ram-probe.zip`.



Igricu možete pokrenuti preko priloženih Code::Blocks projekata—dovoljno je samo otvoriti i pokrenuti relevantan projekat za vašu platformu (Windows/Linux). Za uspešnu kompilaciju na Windows platformi trebaće vam što skorija verzija MinGW kompajlera koja u sebi ima podršku za `pthread` biblioteku, dok će vam za Linux trebati instalirana `ncurses` biblioteka.

Osim što vam dozvoljava kontrolu nad Petrom, ova igrica je ostavila dostupnom infrastrukturu pomoću koje možete "presresti" bilo koje zapisivanje podataka u internu memoriju igrice; naime, metoda

```
on_write(long long address, int value)
```

će se pozvati svaki put kada igrica zapiše vrednost `value` na memorijsku adresu `address`. Ovo je ujedno i jedini momenat kada vam igrica predaje kontrolu izvršavanja programa. Takođe vam je dostupna metoda

```
write(long long address, int value)
```

kojom vi možete direktno zapisati vrednost `value` na adresu `address`.

Unutar datog Code::Blocks projekta nalazi se C++ fajl `probe.cpp`, koji će automatski da se linkuje sa kôdom igrice, sa praznom implementacijom metode `on_write`.

Kroz ovo pitanje, vaš zadatak će biti da osmislite pristupe koji će, koristeći ove dve metode, na razne načine eksploatisati igricu i naterati je da se ponaša na nepredviđene načine. Zadatak predajte u `.zip` arhivi gde je, za svaki urađeni deo zadatka, potrebno priložiti prikladnu implementaciju fajla `probe.cpp`, kao i `.txt/.pdf` fajl koji sadrži detaljno objašnjenje vaših pristupa.

Analiza i manipulacija memorijskim sadržajem

(*a*) U osnovnoj implementaciji igrice, izuzetno je teško uspešno preći zadati nivo sa pokupljenim *svim* poenima; glavni razlog je vrlo strogo vremensko ograničenje. Implementirajte pristup kojim će vremensko ograničenje moći da se podigne u toku igranja, tako da omogući nesmetan prelazak nivoa.

Odgovor: Najjednostavniji od tri podzadatka. Preostalo vreme se tokom jednog nivoa konstantno čuva na jednoj adresi, i ne prenosi se ni na koji način. Samim tim, očekivani pristup treba da identifikuje ovaj registar i u njega zapiše dovoljno visoku vrednost.

Očekivanje je da će ovo uspešno rešiti gotovo svi kandidati koji se odluče za ovo pitanje.

N.B. 1. U ovom—kao i svakom sledećem—podzadatku, podrazumeva se drugačiji *nasumični seed* pri svakom pokretanju programa, i samim tim ne davati pune bodove pristupima koji identifikuju neku konkretnu adresu i zatim svaki put zapisuju u istu.

N.B. 2. Tokom ocenjivanja ovog zadatka je identifikovano da, na određenim verzijama MinGW kompajlera, u implementaciji klase `std::random_device` (koji se koristi za postavljanje početnog *seed*-a) postoji bag[5] koji izaziva da se svaki put vraća ista vrednost. Na sreću, nije bilo mnogo predatih rešenja koja hardkoduju vrednosti adresa, ali su ta rešenja kao posledicu dobila više bodova nego što bi, po originalnom planu, trebalo da dobiju.

(*b*) Korisniku je u igrici dozvoljeno kretanje koristeći tastere `W/A/S/D` na tastaturi, i neophodan je relativno veliki broj pokreta da bi se nivo prešao. Ukoliko bismo mogli da omogućimo kretanje drugačijim sredstvima, nivo je moguće preći u *samo tri promene pozicije*! Implementirajte pristup koji nam ovo omogućava.

Odgovor: U ovom podzadatku je potrebno zaključiti da se za procesiranje kretanja smenjuju uloge dva para adresa (u stilu {`prev, next`}_{`x, y`}). Pristupi koji ažuriraju samo jedan registar mogu da osvoje najviše 50% bodova, s obzirom da je tako (nesvesno) napravljeno šest poteza. Takođe je moguće (i jednostavnije) promeniti broj skupljenih ključeva (fiksan registar) na 2 i onda odraditi tri normalna pokreta.

(*c*) Broj bodova koji se dobija za određene akcije u igrici je identifikovan kao jako mali. Implementirajte pristup kojim se omogućava prelazak nivoa uz osvajanje *preko* 10^6 *bodova* unutar njega.

[5] `http://stackoverflow.com/questions/18880654/why-do-i-get-the-same-sequence-for-every-run-with-stdrandom-device-with-mingw`

Odgovor: Najteži podzadatak koji ujedno i nosi najviše bodova; trenutni bodovi se zapisuju na *nasumičnu neiskorišćenu adresu* (SSA stil) prilikom svake manipulacije, i takođe se, radi efekta "animacije", skor zapisuje na konačnu adresu sekvencijalno *pet* puta (svaki put dodajući jednu petinu od ukupne, hardkodovane, vrednosti trenutnog skupljenog elementa). Dodatna nasumična zapisivanja u privremene adrese dodatno otežavaju frekvencijsku analizu, i nelegitimni inkrementi bodova se odbijaju! Za pune bodove neophodno je da rešenje radi za veliki dijapazon mogućih okolnosti!

(*d*) Navedite (bez implementacije) *bar još dva* aspekta igrice koji se mogu manipulisati na sličan način.

Odgovor: Mogući odgovori: manipulacija stanja mape, količine skupljenih ključeva, teleportacija van granica mape...