University of Belgrade
Faculty of Mathematics

RAZVOJ SOFTVERA

ZADACI ZA VEŽBU

Nikola Ajzenhamer

31. maj 2021.

Sadržaj

Napomena U svim zadacima se podrazumeva da je potrebno primenjivati tehnike o kojima je diskutovano na vežbama. Neke od tih tehnika su:

- Smisleno imenovanje jediničnih testova.
- Ispravna upotreba Arrange-Act-Assert paradigme za pisanje jediničnih testova.
- Ispravan rad sa dinamičkim objektima. Posebno voditi računa o curenju memorije.
- Ispravno razrešavanje problema koji mogu nastati u konkurentnom okruženju.
- Korektan dizajn grafičkog korisničkog interfejsa (koristiti odgovarajuće kontrole i raspoređivanje elemenata – vertikalno, horizontalno, formular ili mreža – da izgledaju kao na slici). Promenom veličine prozora, raspored treba da ostane isti (uz odgovarajuće promene veličine određenih kontrola).

Ovi zadaci mogu da posluže za vežbanje gradiva sa časova vežbi u cilju pripreme ispita. Ipak, ne treba pretpostaviti da će obim ili forma zadataka odgovarati zadacima na ispitu.

FUNKCIONALNA PARADIGMA

Zadatak 1.1. *U programskom jeziku C++ napisati klasu* `template<typename T> averager` *koja implementira:*

- *Metod* `void save_value(T new_val)` *koja pamti vrednost koja mu se prosleđuje kao argument.*
- Operator poziva funkcijskog objekta *(engl.* call operator*) koji izračunava prosečnu vrednost tekućih zapamćenih vrednosti.*

Zadatak 1.2. *U programskom jeziku C++ napisati klasu* `template<typename T> splitter` *koja implementira:*

- *Konstruktor* `splitter(std::map<int, T> values)` *koji pamti mapu koja mu se prosleđuje kao argument.*
- Operator poziva funkcijskog objekta *(engl.* call operator*) koji ima argument* `key` *tipa* `int` *i koji vraća vektor svih vrednosti (tipa* `T`*) iz zapamćene mape čiji ključ je manji od vrednosti* `key`*.*

REFAKTORISANJE KODA

Zadatak 2.1. *U programskom jeziku C++ refaktorisati operator poziva funkcijskog objekta iz zadatka 1.1 tako da ne koristi* `for`*,* `while`*,* `do while` *petlje, niti* `foreach` *konstrukt, niti* `goto`*, niti* `std::for_each`*, niti rekurziju.*

Zadatak 2.2. *U programskom jeziku C++ refaktorisati operator poziva funkcijskog objekta iz zadatka 1.2 tako da ne koristi* `for`*,* `while`*,* `do while` *petlje, niti* `foreach` *konstrukt, niti* `goto`*, niti* `std::for_each`*, niti rekurziju.*

Zadatak 2.3. *U programskom jeziku C++:*

- *Napisati rekurzivnu funkciju* `template<typename Iter> void quicksort(Iter first, Iter last)` *koja implementira algoritam brzog sortiranja i njime sortira kolekciju, čiji se iterator početka i kraja prosleđuju kao argumenti.*
- *Refaktorisati napisanu funkciju* `quicksort` *tako da ne koristi* `for`*,* `while`*,* `do while` *petlje, niti* `foreach` *konstrukt, niti* `goto`*, niti* `std::for_each` *(rekurzija je dozvoljena).*

TESTIRANJE SOFTVERA

Naredni zadaci služe za vežbanje pisanja jediničnih testova.

Zadatak 3.1. *U programskom jeziku C++ koristeći biblioteku Catch2:*

- *Napisati 10 jediničnih testova koji testiraju ponašanje operatora poziva funkcijskog objekta iz zadatka 1.1.*
- *Koristeći napisane testove, proveriti ispravnost refaktorisanog operatora iz zadatka 2.1.*

Zadatak 3.2. *U programskom jeziku C++ koristeći biblioteku Catch2:*

- *Napisati 10 jediničnih testova koji testiraju ponašanje operatora poziva funkcijskog objekta iz zadatka 1.2.*
- *Koristeći napisane testove, proveriti ispravnost refaktorisanog operatora iz zadatka 2.2.*

Zadatak 3.3. *U programskom jeziku C++ koristeći biblioteku Catch2:*

- *Napisati funkciju* `std::vector<std::string> load_songs()` *koja iz datoteke* `pesme.txt` *učitava nazive pesama. Svaka pesma se nalazi u jednom redu i može se sastojati od više reči.*
- *Napisati funkciju* `std::vector<std::string> switch_songs(std::vector<std::string> songs, size_t a, size_t b)` *koja premešta dve pesme koje se nalaze na pozicijama* `a` *i* `b`.
- *Napisati 10 jediničnih testova koji testiraju ispravnost funkcije* `switch_songs`.

RAZVOJ VOĐEN TESTOVIMA (TDD)

U narednim zadacima se očekuje da rešenja sadrže makar 7 jediničnih testova. Testove pisati u redosledu tako da bude jasan tok razvoja pomoću *razvoja vođenog testovima*.

Zadatak 3.4. *Koristeći tehniku* razvoja vođenog testovima *razviti klasu* `TermSearch` *koja služi za pretraživanje pojmova u rečniku. Klasi je moguće dati unapred definisan rečnik na osnovu neke datoteke, ali je moguće i ručno dodavati nove pojmove i njihove definicije. Ukoliko se pojmovi čitaju iz datoteke, očekuje se da je datoteka u formatu*[1]*:*

```
pojam 1 - definicija 1
pojam 2 - definicija 2
...
```

Obezbediti naredni javni interfejs klase `TermSearch`*:*

- *Metod* `addFromDictionary` *koji prihvata objekat klase* `std::string` *koji sadrži putanju do jedne validne datoteke sa pojmovima i njihovim definicijama. Učitati iz datoteke pojmove i njihove definicije.*
- *Metod* `addTerm` *koji prihvata dve niske koje označavaju term i njegovu definiciju, redom, i dodaje ih u rečnik. Ne pretpostavljati ništa o sadržaju niski*[2]*.*
- Operator poziva funkcijskog objekta *(eng.* call operator*) koji prihvata nisku i pretražuje je u datom rečniku. Ukoliko je pronađe, metod vraća par (pojam, definicija). Inače, vraća par sa dve prazne niske. Za implementiranje ovog metoda nije dozvoljeno koristiti ni* `for`*, ni* `while`*, ni* `do-while`*, ni* `std::for_each`*, ni* `GOTO`*.*

Napisati C++ program koji demonstrira rad razvijene klase `TermSearch`*. Učitati rečnik iz datoteke ispod, a zatim dodati 3 pojma (ispod) i njihove definicije u rečnik. Proveriti postojanje pojmova* `"fantasy"`*,* `"make believe"` *i* `"make"`*.*

Primer validne datoteke:

```
backdrop - a painted cloth hung across the rear of a stage
cackle - to laugh especially in a harsh or sharp manner
waft - to move or go lightly on or as if on a buoyant medium
fantasy - imaginative fiction
pageant - an elaborate colorful exhibition or spectacle often with music
make believe - to act as if something known to be imaginary is real or true
immerge - to plunge into or immerse oneself in something
```

Reči koje se naknadno dodaju:

```
cabriolet - a light two wheeled one horse carriage with a folding leather hood
fabricate - to construct from diverse and usually standardized parts
garnish - to add decorative or savory touches to (food or drink)
```

[1] Pretpostaviti da se pojmovi i definicije mogu sastojati od više reči, ali da neće sadržati karakter – (crticu).
[2] Drugim rečima, razmisliti o mogućim specijalnim slučajevima i obraditi ih.

Zadatak 3.5. *Koristeći tehniku* razvoja vođenog testovima *razviti klasu* `LyricShuffler` *koja premešta stihove pesama. Objekat ove klase učitava stihove pesme iz datoteke čija je putanja data. Očekuje se da se u datoteci svaki stih nalazi u zasebnom redu (sa eventualnim praznim linijama). Omogućiti da klasa može da zamenjuje dva stiha pesme.*

Obezbediti naredni javni interfejs klase `LyricShuffler`*:*

- *Konstruktor koji prihvata objekat klase* `std::string` *koji sadrži putanju jedne validne datoteke sa pesmom. Ignorisati prazne linije.*
- *Metod* `toString()` *koji vraća* `std::string` *koji predstavlja sadržaj pesme sa rednim brojem ispred svakog stiha. Za implementiranje ovog metoda nije dozvoljeno koristiti ni* `for`*, ni* `while`*, ni* `do-while`*, ni* `std::for_each`*, ni* `GOTO`*.*
- Operator poziva funkcijskog objekta *(eng.* call operator*) koji prihvata dva nenegativna cela broja koja predstavljaju redne brojeve stihova i zamenjuje stihove.*

Napisati C++ program koji demonstrira rad razvijene klase `LyricShuffler`*. Po učitavanju stihova, na standardni izlaz se ispisuje pesma uz redne brojeve stihova. Korisnik tada unosi redne brojeve dva stiha čija mesta želi da zameni. Nakon toga, program ispisuje izmenjeni sadržaj pesme. Postupak se ponavlja dok korisnik ne unese nulu kao redni broj stiha.*

Primer validne datoteke:

```
Honno sabishisa ga
Konna watashi o
Tsuyoku shitekureru

Myakuutsu kodou
Shizukana hodo
Kikoeru hazu deshou

Ima wa mada hitomi tojite
Mimi o sumasu no
Kokoro no SIGNAL
```

SERIJALIZACIJA I DESERIJALIZACIJA

Zadatak 4.1. *U programskom jeziku C++ koristeći biblioteku Qt napisati:*

- *Klasu* `Song` *koja implementira:*
  - *Metod* `void load(QVariant data)` *koja iz prosleđenog objekta deserijalizuje podatke o jednoj pesmi. Podaci se smeštaju kao članice klase* `Song`*.*
  - *Metod* `QVariant save() const` *koja serijalizuje podatke serijalizuje podatke o pesmi. Podaci se čitaju iz članica klase* `Song`*.*
- *Aplikaciju koja redom:*
  - *Čita sadržaj datoteke* `song.xml` *koja sadrži podatke o jednoj pesmi u odgovarajućem formatu i deserijalizuje učitani podatak iz datoteke.*
  - *Zahteva od korisnika da unese rejting i vrši izmenu rejtinga učitane pesme.*
  - *Zahteva od korisnika da unese naziv izlazne XML datoteke i serijalizuje izmenjenu pesmu u tu datoteku.*

Primer XML datoteke `song.xml` *je dat u nastavku. Pretpostaviti da broj podataka* `<Feature>` *i* `<Genre>` *može biti proizvoljan nenegativan ceo broj.*

```
<?xml version="1.0"?>
<Song type="QVariantMap">
    <Artist type="QString">Lady Gaga</Artist>
    <Title type="QString">Rain On Me</Title>
    <Album type="QString">Chromatica</Album>
    <Features type="QVariantList">
        <Feature type="QString">Ariana Grande</Feature>
    </Features>
    <Rating type="double">10.00</Rating>
    <Genres type="QVariantList">
        <Genre type="QString">Pop</Genre>
        <Genre type="QString">Dance</Genre>
    </Genres>
    <Year type="uint">2020</Year>
</Song>
```

Zadatak 4.2. *U programskom jeziku C++ koristeći biblioteku Qt napisati:*

- *Klasu* `Horse` *koja implementira:*
  - *Metod* `void load(QVariant data)` *koja iz prosleđenog objekta deserijalizuje podatke o trkama za jednog trkačkog konja. Podaci se smeštaju kao članice klase* `Horse`.
  - *Metod* `QVariant save() const` *koja serijalizuje podatke o trkama za jednog trkačkog konja. Podaci se čitaju iz članica klase* `Horse`.
- *Aplikaciju koja redom:*
  - *Čita sadržaj datoteke* `horses.json` *koja sadrži proizvoljan pozitivan celi broj linija, pri čemu svaka linija sadrži podatke o jednom trkačkom konju u odgovarajućem formatu (videti ispod).*
  - *Zahteva od korisnika da unese prosečno vreme.*
  - *Deserijalizuje učitane podatke i ispisuje imena samo onih trkačkih konja čije je prosečno vreme manje od unetog prosečnog vremena.*

Primer datoteke `horses.json` *je dat u nastavku:*

```
[
    {
        "name": "SuperHorsey",
        "numberOfRaces": 10,
        "numberOfVictories": 5,
        "averageTime": 1000
    },
    {
        "name": "LucyInTheSky",
        "numberOfRaces": 20,
        "numberOfVictories": 2,
        "averageTime": 4500
    },
    {
        "name": "SuperMoose",
        "numberOfRaces": 10,
        "numberOfVictories": 9,
        "averageTime": 1500
    },
    {
        "name": "TimTheEnchanter",
        "numberOfRaces": 50,
```

```
        "numberOfVictories": 40,
        "averageTime": 2000
    },
]
```

Zadatak 4.3. *Napisati datoteku* `song.json` *koja sadrži podatke koji odgovaraju podacima u datoteci* `song.xml`*, samo u JSON formatu. Uraditi zadatak 4.1 s tim da se podaci učitavaju u JSON formatu iz napisane datoteke.*

Zadatak 4.4. *Napisati datoteku* `horses.xml` *koja sadrži podatke koji odgovaraju podacima u datoteci* `horses.json`*, samo u XML formatu. Uraditi zadatak 4.2 s tim da se podaci učitavaju u XML formatu iz napisane datoteke.*

KONKURENTNO PROGRAMIRANJE

Zadatak 5.1. *U programskom jeziku C++ koristeći biblioteku Qt napisati aplikaciju koja:*

- *Iz datoteke* `lozinka.txt` *učitava celi broj* K *i jednu nisku* S *(koja. se sastoji samo od malih i velikih slova).*
- *Konstruiše* $|S|$ *niti i svakoj niti prosleđuje odgovarajući karakter niske* S *i broj* K*.*
- *Jedna nit izračunava šifrovani karakter tako što izvršava „rotiranje udesno za* K*". Na primer, ako je* $K = 3$*, onda nit šifruje* `A` *u* `D`*,* `B` *u* `E`*, ...,* `W` *u* `Z`*,* `X` *u* `A`*, itd. (slično za mala slova).*
- *Nakon što sve niti šifruju svoje karaktere, potrebno je rekonstruisati šifrovanu nisku spajanjem šifrovanih karaktera u ispravnom poretku.*

Zadatak 5.2. *U programskom jeziku C++ koristeći biblioteku Qt napisati aplikaciju koja:*

- *Iz datoteke* `horses.json` *učitava podatke o trkačkim konjima, pri čemu je datoteka zapisana u formatu kao u zadatku 4.2. Potrebno je iz datoteke deserijalizovati podatke u objekte klase* `Horse` *iz istog zadatka.*
- *Konstruiše za svakog konja po jednu nit koja izračunava vreme koje je konju potrebno da stigne do cilja. Vreme se računa po narednoj formuli:*

$$vreme = prosecno\_vreme * (1 + random(-broj\_pobeda/broj\_trka, broj\_pobeda/broj\_trka))$$

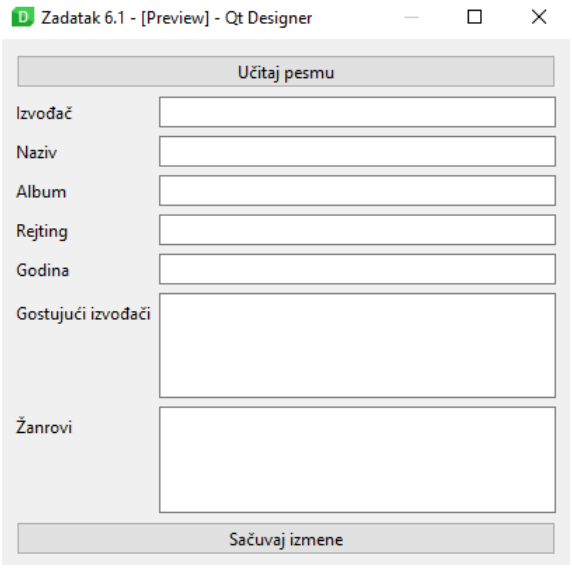
- *Nakon što sve niti izračunaju vremena, potrebno je ispisati u konzolu imena konja, sortirana opadajuće po izračunatim vremenima.*

APLIKACIJE SA GRAFIČKIM KORISNIČKIM INTERFEJSOM

Zadatak 6.1. *U programskom jeziku C++ koristeći biblioteku Qt napisati aplikaciju sa grafičkim korisničkim interfejsom kao na slici 1.*

- *Klikom na dugme „Učitaj pesmu" otvoriti dijalog za pretraživanje datoteka na sistemu datoteka. Aplikacija zahteva od korisnika da odabere jednu XML datoteku u formatu kao u zadatku 4.1. Kreirati objekat klase* `Song` *u koji će biti deserijalizovani podaci iz učitane datoteke. Ovaj objekat će biti korišćen nadalje.*
- *U formularu prikazati podatke o učitanoj pesmi iz odabrane XML datoteke. Za članice koje se odnose na: (1) izvođače koji učestvuju u pesmi i (2) žanrove, u odgovarajuća višelinijska tekstualna polja, upisati svaku vrednost u novom redu. Korisnik može da izvrši izmene u ovom formularu. Omogućiti da svakom izmenom bilo kog polja, odgovarajuće članice kreiranog objekta klase* `Song` *budu takođe izmene novom vrednošću.*

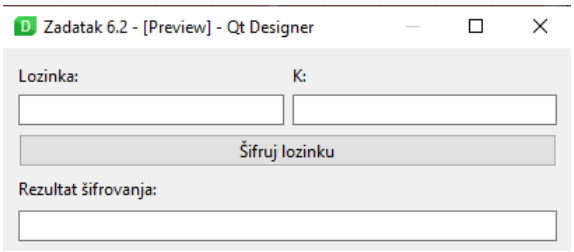
- *Klikom na dugme „Sačuvaj izmene”, koje je onemogućeno sve dok korisnik ne učita neku pesmu pomoću dugmeta „Učitaj pesmu”, otvoriti se dijalog za čuvanje datoteke na sistemu datoteka. Aplikacija zahteva od korisnika da odabere datoteku u koju će serijalizovati objekat klase* `Song` *i zatim serijalizovati taj objekat.*



Slika 1: Grafički korisnički interfejs za zadatak 6.1.

Zadatak 6.2. *U programskom jeziku C++ koristeći biblioteku Qt napisati aplikaciju sa grafičkim korisničkim interfejsom kao na slici 2.*

- *Korisnik prvo unosi unosi lozinku u polje „Lozinka” koja predstavlja nisku S i broj K u polju „K” (oznake se koriste kao u zadatku 5.1).*
- *Klikom na dugme „Šifruj lozinku”, šifrovati unesenu lozinku pomoću niti procesom opisanom u zadatku 5.1.*
- *Šifrovanu lozinku upisati u jednolinijsko tekstualno polje „Rezultat šifrovanja”.*

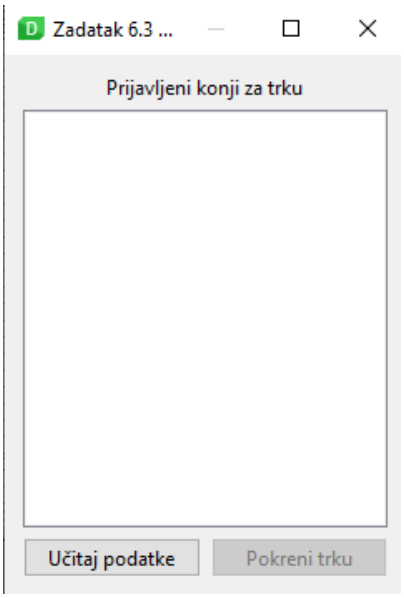


Slika 2: Grafički korisnički interfejs za zadatak 6.2.

Zadatak 6.3. *U programskom jeziku C++ koristeći biblioteku Qt napisati aplikaciju sa grafičkim korisničkim interfejsom kao na slici 3.*

- *Klikom na dugme „Učitaj podatke”, otvara se dijalog za izbor JSON datoteke sa sistema datoteka. Pretpostaviti da je izabrana datoteka u formatu kao iz zadatka 4.2. Pretpostaviti da ime jedinstveno identifikuje konja. Nakon odabira datoteke, potrebno je deserijalizovati podatke u objekte klase* `Horse`*, opisane u istom zadatku.*
- *Nakon učitavanja podataka, u listi „Prijavljeni konji za trku” tipa* `QListWidget` *prikazati imena trkačkih konja.*

- *Aplikacija prvo očekuje od korisnika da klikne na neko od ponuđenih imena konja iz liste (kako bi se kladio na njega). Kada korisnik klikne na nekog konja iz liste, omogućiti dugme „Pokreni trku" koje je inicijalno onemogućeno. Klikom na ovo dugme, pokrenuti za svakog konja po jednu nit koja izračunava njegovo vreme do cilja, kao u zadatku 5.2.*
- *Kada sve niti izračunaju vremena za sve konje, prikazati podatke o pobedniku i osvojeno mesto konja na koga se korisnik kladio u prozoru sa porukom (`QMessageBox`).*



Slika 3: Grafički korisnički interfejs za zadatak 6.3.

Naredni zadatak kombinuje sve prethodno gradivo, te stoga može predstavljati veći izazov od prethodnih zadataka. Zbog toga će jedino za ovaj zadatak biti dato rešenje ispod. Dodatno, s obzirom da je cilj ovog zadatka da čitaoci provere koliko dobro (i brzo) umeju da povezuju gradivo koje su naučili za ispit, *ne preporučujemo* da se rešenje pogleda pre nego što čitaoci pokušaju samostalno da urade zadatak. *Posebno ne preporučujemo* da se rešenje uči napamet, već da se porazmisli o svakoj odluci koja je doneta u rešenju. Dodatno, porazmisliti o mogućim alternativama u rešenju, najpre, da li je zadatak mogao da se uradi „kraće".

Zadatak 6.4. *U programskom jeziku C++ koristeći biblioteku Qt napisati aplikaciju sa grafičkim korisničkim interfejsom kao na slici 4.*

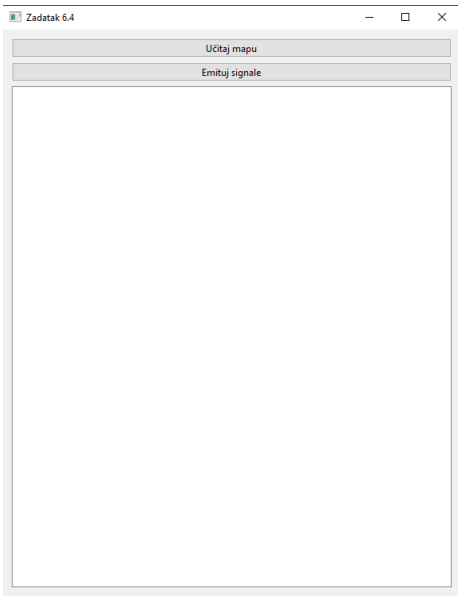
- *Napisati klasu `Map` koja predstavlja dvodimenzionu mapu.*
  - *Mapa je predstavljena matricom celih brojeva (`m_map`), vektorom koordinata izvora signala (`m_sources`), brojem redova (`m_rows`) i brojem kolona (`m_columns`).*
  - *Implementirati metod `void fromVariant(const QVariant &variant)` koji deserijalizuje podatke o mapi iz JSON datoteke (opis datoteke i primer su dati u narednoj tački).*
- *Klikom na dugme „Učitaj mapu", otvara se dijalog za izbor JSON datoteke sa sistema datoteka. JSON datoteka koja se učitava predstavlja objekat koji sadrži naredne informacije o mapi:*
  - *Broj redova (n).*
  - *Broj kolona (m).*
  - *Lokacije izvora signala na mapi ($sources$). Svaki izvor ima svoje koordinate na mapi (i, j) i jačinu signala ($sourcePower$).*

  Primer datoteke je dat u nastavku:

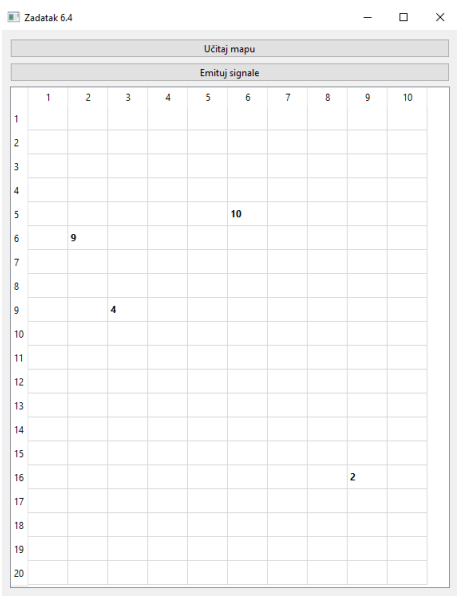
```
{
    "n": 20,
    "m": 10,
    "sources": [
        { "i": 4, "j": 5, "sourcePower": 10 },
        { "i": 8, "j": 2, "sourcePower": 4 },
        { "i": 15, "j": 8, "sourcePower": 2 },
        { "i": 5, "j": 1, "sourcePower": 9 }
    ]
}
```

Na osnovu učitanih podataka deserijalizovati podatke u objekat klase `Map`*. Sva polja u mapi koja ne predstavljaju izvore signala postaviti na* 0*.*

- *Zatim, prikazati deserijalizovanu mapu u tabelu tipa* `QTableWidget` *kao na slici 5. Vrednosti* 0 *se prikazuju kao prazna niska, a izvore signala prikazati podebljanim slovima. Sva ostala polja treba prikazati normalnim fontom.*



Slika 4: Grafički korisnički interfejs za zadatak 6.4 na početku programa.



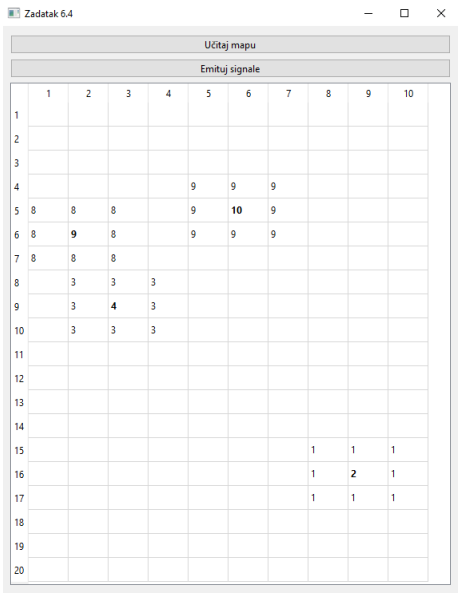
Slika 5: GKI za zadatak 6.4 nakon učitavanja mape iz datoteke.

- *Klikom na dugme „Emituj signale”, za svaki izvor pokrenuti po jednu nit koja emituje signal iz tog izvora.*
- *Jedna nit dobija informacije o koordinatama izvora i jačini izvora signala. Zatim, ponavlja sledeće korake sve dok signal ne „oslabi” do nule:*
  - *Uspavljuje se na 2 sekunde.*
  - *„Proširuje” dejstvo signala za 1 oko izvora signala. Ovo je prikazano na slikama 6 i 7. Objasnimo ovo ponašanje malo detaljnije:*
    - *Na slici 6 je prikazana situacija nakon prvog uspavljivanja niti, kada se signal proširio za korak 1 oko izvora signala. Primetite da proširivanje signala umanjuje jačinu tog signala za 1. Na primer, oko izvora 10 vidimo da polja imaju vrednost 9.*
    - *Zatim, nakon još jednog uspavljivanja niti, signali sada utiču na polja sa korakom 2 od njihovih izvora, kao na slici 7. Signal sa izvorom 10 sada ima jačinu 8 i on utiče na polja oko vrednosti 9 iz prethodnog koraka , na primer, polje sa koordinatama* $(4, 4)$*. Međutim, na to polje utiče i signal sa izvorom 9 i to jačinom 7, te zbog toga u toj ćeliji vidimo vrednost* $8 + 7 = 15$*. U istom trenutku, signal sa izvorom 2 je oslabio do nule i time se njegova nit završava.*

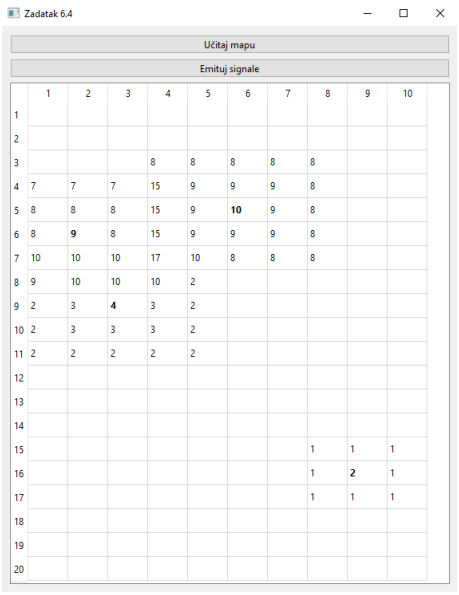
* *Ostale niti nastavljaju da utiču na polja sa koracima 3, 4, itd. sve do njihovog „oslabljenja" do nule. Pod „oslabljenjem" signala do nule smatra se trenutak kada ne postoji nijedno novo polje na mapi na koje signal može da utiče.*

Zabranjeno je kopirati/pomerati mapu između niti! Sve niti moraju da operišu nad istim podacima!

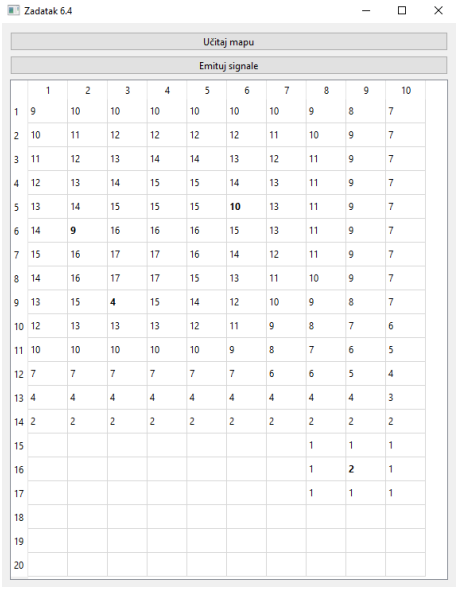
- *Na slici 8 je prikazana situacija kada su sve niti završile, s obzirom da su svi signali „oslabili". Primetite da neka polja na mapi nisu pod uticajem nijednog signala. Očigledno, ovo ponašanje zavisi od dimenzija mape, raspoređivanja izvora signala i njihove snage.*



Slika 6: GKI za zadatak 6.4 nakon započinjanja emitovanja signala (korak 1).



Slika 7: GKI za zadatak 6.4 u toku emitovanja signala (korak 2).



Slika 8: GKI za zadatak 6.4 nakon što su svi izvori signala emitovali signal najdalje što su mogli.

Datoteka `Map.hpp`:

```
#ifndef MAP_HPP
```

```
#define MAP_HPP

#include <QVector>
#include <QVariant>

class Map
{
public:
    Map();

    void increaseSignalPower(int i, int j, int power);
    void fromVariant(const QVariant &variant);
    inline int rowCount() {
        return m_rows;
    }
    inline int columnCount() {
        return m_columns;
    }
    inline int field(int i, int j) {
        return m_map[i][j];
    }
    bool isSource(int i, int j);
    inline const QVector<QPair<int, int>> &sources() const {
        return m_sources;
    }

private:
    void initMap(int n, int m);
    void addSignalSource(int i, int j, int sourcePower);

private:
    QVector<QVector<int>> m_map;
    QVector<QPair<int, int>> m_sources;
    int m_rows;
    int m_columns;
};

#endif // MAP_HPP
```

Datoteka `Map.cpp`:

```
#include "Map.hpp"

Map::Map()
{

}

void Map::initMap(int n, int m)
{
    m_rows = n;
    m_columns = m;
    m_sources = {};
    m_map = {};

    for (int i = 0; i < m_rows; ++i) {
        m_map.push_back({});
        for (int j = 0; j < m_columns; ++j) {
```

```
            m_map[i].push_back(0);
        }
    }
}

void Map::addSignalSource(int i, int j, int sourcePower)
{
    m_map[i][j] = sourcePower;
    m_sources.push_back({i, j});
}

bool Map::isSource(int i, int j)
{
    for (const auto &source : m_sources) {
        if (source.first == i && source.second == j) {
            return true;
        }
    }
    return false;
}

void Map::increaseSignalPower(int i, int j, int power)
{
    if (isSource(i, j)) {
        return;
    }
    m_map[i][j] += power;
}

void Map::fromVariant(const QVariant &variant)
{
    const auto varMap = variant.toMap();

    const auto n = varMap.value("n").toInt();
    const auto m = varMap.value("m").toInt();
    initMap(n, m);

    const auto sourcesList = varMap.value("sources").toList();
    for (const auto &source : sourcesList) {
        const auto sourceMap = source.toMap();
        const auto i = sourceMap.value("i").toInt();
        const auto j = sourceMap.value("j").toInt();
        const auto sourcePower = sourceMap.value("sourcePower").toInt();
        addSignalSource(i, j, sourcePower);
    }
}
```

Datoteka `MainWindow.hpp`:

```
#ifndef MAINWINDOW_HPP
#define MAINWINDOW_HPP

#include <QWidget>
#include "Map.hpp"
#include <QMutex>

QT_BEGIN_NAMESPACE
namespace Ui { class MainWindow; }
```

```
QT_END_NAMESPACE

class MainWindow : public QWidget
{
    Q_OBJECT

public:
    MainWindow(QWidget *parent = nullptr);
    ~MainWindow();

    inline Map &map() {
        return m_map;
    }

    inline QMutex &mutexMap() {
        return m_mutexMap;
    }

private slots:
    void on_pbLoadMap_clicked();
    void on_pbStartSignaling_clicked();
    void on_signalMadeProgress();

private:
    void initTableWidget();
    void drawMapInTableWidget();

private:
    Ui::MainWindow *ui;
    Map m_map;
    QMutex m_mutexMap;
};
#endif // MAINWINDOW_HPP
```

Datoteka `MainWindow.cpp`:

```
#include "MainWindow.hpp"
#include "ui_MainWindow.h"
#include "SignalThread.hpp"

#include <QFileDialog>
#include <QJsonDocument>
#include <QVariant>
#include <QDebug>
#include <QFile>
#include <QMutexLocker>

MainWindow::MainWindow(QWidget *parent)
    : QWidget(parent)
    , ui(new Ui::MainWindow)
{
    ui->setupUi(this);
}

MainWindow::~MainWindow()
{
    delete ui;
}
```

```
void MainWindow::on_pbLoadMap_clicked()
{
    const auto fileName =
        QFileDialog::getOpenFileName(this, "Molimo odaberite JSON datoteku",
                                     "", "JSON (*.json)");
    if (fileName.isNull()) {
        return;
    }

    QFile file(fileName);
    if (!file.open(QFile::ReadOnly)) {
        return;
    }
    const auto mapData = QJsonDocument::fromJson(file.readAll());
    const auto mapVariant = mapData.toVariant();
    m_map.fromVariant(mapVariant);

    initTableWidget();
    drawMapInTableWidget();
}

void MainWindow::on_pbStartSignaling_clicked()
{
    const auto &sources = m_map.sources();
    for (const auto &source : sources) {
        const auto i = source.first;
        const auto j = source.second;
        const auto sourcePower = m_map.field(i, j);

        auto thread = new SignalThread(i, j, sourcePower, this);
        connect(thread, &SignalThread::signalMadeProgress,
                this, &MainWindow::on_signalMadeProgress);
        connect(thread, &SignalThread::finished,
                thread, &SignalThread::deleteLater);
        thread->start();
    }
}

void MainWindow::on_signalMadeProgress()
{
    QMutexLocker lock(&m_mutexMap);
    drawMapInTableWidget();
}

void MainWindow::initTableWidget()
{
    ui->twMap->clear();

    const auto rows = m_map.rowCount();
    const auto columns = m_map.columnCount();
    ui->twMap->setRowCount(rows);
    ui->twMap->setColumnCount(columns);
    for (int i = 0; i < rows; ++i) {
        for (int j = 0; j < columns; ++j) {
            const auto tableItem = new QTableWidgetItem();
            if (m_map.isSource(i, j)) {
```

```
                QFont font;
                font.setBold(true);
                tableItem->setFont(font);
            }
            ui->twMap->setItem(i, j, tableItem);
        }
    }
}

void MainWindow::drawMapInTableWidget()
{
    const auto rows = m_map.rowCount();
    const auto columns = m_map.columnCount();
    for (int i = 0; i < rows; ++i) {
        for (int j = 0; j < columns; ++j) {
            const auto fieldText =
                m_map.field(i, j) != 0 ? QString::number(m_map.field(i, j)) : "";
            ui->twMap->item(i, j)->setText(fieldText);
        }
    }
}
```

Datoteka `SignalThread.hpp`:

```
#ifndef SIGNALTHREAD_HPP
#define SIGNALTHREAD_HPP

#include <QThread>

class SignalThread : public QThread
{
    Q_OBJECT
public:
    SignalThread(int i, int j, int sourcePower, QObject *parent);

    void run() override;

signals:
    void signalMadeProgress();

private:
    int m_i;
    int m_j;
    int m_sourcePower;
};

#endif // SIGNALTHREAD_HPP
```

Datoteka `SignalThread.cpp`:

```
#include "SignalThread.hpp"
#include "MainWindow.hpp"

#include <QMutexLocker>

SignalThread::SignalThread(int i, int j, int sourcePower, QObject *parent)
    : QThread(parent)
    , m_i(i)
```

```
    , m_j(j)
    , m_sourcePower(sourcePower)
{

}

void SignalThread::run()
{
    auto mainWindow = qobject_cast<MainWindow *>(parent());
    auto step = 0;

    for (;;) {
        sleep(2);
        ++step;

        if (m_sourcePower-step <= 0) {
            break;
        }

        QMutexLocker lock(&mainWindow->mutexMap());
        auto &map = mainWindow->map();

        const auto n = map.rowCount();
        const auto m = map.columnCount();
        const auto i_min = m_i - step;
        const auto i_max = m_i + step;
        const auto j_min = m_j - step;
        const auto j_max = m_j + step;
        auto numberOfChanges = 0;

        for (int i = i_min; i <= i_max; ++i) {
            if (i < 0 || i > n-1) {
                continue;
            }
            if (i == i_min || i == i_max) {
                for (int j = j_min; j <= j_max; ++j) {
                    if (j < 0 || j > m-1) {
                        continue;
                    }
                    map.increaseSignalPower(i, j, m_sourcePower-step);
                    ++numberOfChanges;
                }
            }
            else {
                int j = j_min;
                if (j >= 0 && j <= m-1) {
                    map.increaseSignalPower(i, j, m_sourcePower-step);
                    ++numberOfChanges;
                }
                j = j_max;
                if (j >= 0 && j <= m-1) {
                    map.increaseSignalPower(i, j, m_sourcePower-step);
                    ++numberOfChanges;
                }
            }
        }
```

```
        if (!numberOfChanges) {
            break;
        }
        else {
            emit signalMadeProgress();
        }
    }
}
```

Datoteka `main.cpp`:

```
#include "MainWindow.hpp"

#include <QApplication>

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    MainWindow w;
    w.show();
    return a.exec();
}
```

Datoteka `signals.pro`:

```
QT       += core gui

greaterThan(QT_MAJOR_VERSION, 4): QT += widgets

CONFIG += c++17

# You can make your code fail to compile if it uses deprecated APIs.
# In order to do so, uncomment the following line.
# disables all the APIs deprecated before Qt 6.0.0
#DEFINES += QT_DISABLE_DEPRECATED_BEFORE=0x060000

SOURCES += \
    Map.cpp \
    SignalThread.cpp \
    main.cpp \
    MainWindow.cpp

HEADERS += \
    MainWindow.hpp \
    Map.hpp \
    SignalThread.hpp

FORMS += \
    MainWindow.ui

# Default rules for deployment.
qnx: target.path = /tmp/$${TARGET}/bin
else: unix:!android: target.path = /opt/$${TARGET}/bin
!isEmpty(target.path): INSTALLS += target
```