Извештај

**I Информације о аутору**

Име: Стефан Кандић

Број индекса: SW 73/2017

Предмет: Објектно орјентисано програмирање 2

Тема: визуализација различитих алгоритама сортирања у језику Це++ уз подршку ФЛТК библиотеке

**II Рад У/И подсистема**

Компонента програма задужена за руковање улазно-излазним системом је класа ИО, која се налази у тесној сарадњи са графичком компоненто, јер су једна од друге зависне.

Први задатак јој је да анализира аргументе командне линије, и да визуелној компоненти проследи задате параметре којима програм касније рукује.

При самом покретању програма, или након корисничке команде из графичког дела апликације, учитава летове из изабраног текстуалног фајла.

Ако фајл не задовољава унапред задату структуру, она главни прозор обавештава о грешци којом он даље рукује без икаквих ометњи у самом раду.

Структура улазне датотеке:

Destination Departure Flight No. Gate No.  
dеst1; departure1; fnum1; gate1;   
...

Када од главног прозора добије сигнал, у излазни фајл који садржи као обејежје уписује прослеђени тест који садржи излазне резултате и сортиране летове.

Несортирани летови

IO WINDOW

Сортирани летови

**III Списак класа,структура и изузетака**

1. [Структуре података](#Структуреподатака)
   1. [vector<?>](#vector)
   2. [list<?>](#list)
   3. [< iterator >](#iterator)
2. [Енумерације](#Енумерације)
   1. [Tip](#Tip)
3. [Изузеци](#Изузеци)
   1. [InvalidData](#InvalidData)
4. [Struct](#Struct)
   1. [CheckBox](#CheckBox)
   2. [Slider](#Slider)
5. [Класе](#Класе)
   1. [Flight](#Flight)
   2. [VisualizedFlight](#VisualizedFlight)
   3. [Instruction](#Instruction)
      1. [ColorChange](#ColorChange)
      2. [ColorSwap](#ColorSwap)
      3. [WidgetSwap](#WidgetSwap)
      4. [RevertColor](#RevertColor)
   4. [Sort](#Sort)
      1. [Selection Sort](#SelectionSort)
      2. [Quick Sort](#QuickSort)
   5. [IO](#IO)
   6. [MyWindow](#MyWindow)

**IV Објашњење коришћених структура**

**Структуре података:**

* vector<?> - вектори су секвенца контејнера који представљају низове променљиве величине

Као и низови, вектори користе суседне меморијске локације за складиштење елемената, али за разлику од низова њихова величина се може мењати динамички. Ово може изазвати да низ мора бити премештен на другу меморијску локацију када се додају нови елементи.

Због тога, у поређењу са низовима, вектори користе више меморије али пружају ефикасно решење за низове којима нећемо унапред знати број елемената.

* list<?> - листе су секвенце контејнера који дозвољавају операције брисања и додавања нових елемената на било којој позицији у константном времену

Двоструко су спрегнуте, и могу чувати своје елементе на логички неповезаним меморијским локацијама помоћу показивача.

У поређењу са другим контејнерима из стандардне библиотеке, листе имају боље перформансе при додавању,брисању и померању елемената са/на било коју позицију

Њихова мана је немогућност директног приступа својим елементима на основу њихове позиције у листи.

* <iterator> - итератор је објекат који показујући на одређени елемент у неком контејнеру елемената, има способност да итерира кроз остале елементе тог контејнера

**Енумерације:**

* enum Tip {flightNum, departure, destination, gate} - служи за одабир критеријума сортирања летова

**Изузеци :**

* InvalidData – позива се када улазни фајл нема предвиђену структуру и одбацује изабрани фајл, програм наставља нормално са радом и кориснику се нуди опција да види како улазни фајл треба да изгледа

**St****ruct -** ови**:**

* struct CheckBox – структура направљена по угледу на Button из graphlib .h и служи за избор активних поља у програму
* struct Slider - структура направљена по угледу на Button из graphlib.h и служи за избор вредности из одређеног опсега

**Класе** :

* Класа Flight са следећим члановима

[1] приватни чланови(атрибути)

* + string f\_number – број лета (јединствени идентификатор)
  + string destination - дестинација
  + string departure – време поласка
  + string gate– ознака капије

[2] методе

* Гетери и сетери за приватне чланове
* bool operator == (Flight &f2) – пореди два лета на основу њиховог идентификатора
* int compare(Flight f2,Tip crit) – пореди два лета на освнову енума
* int compareDeparture (Flight &f2) – пореди два лета на основу времена њиховог поласка

[3] преклопљени оператори

* istream& operator>> (istream &in, Flight &f); – оператор за учитавање лета из произвољног тока
* ostream& operator<< (ostream &out, Flight &f) - оператор за уписивање у произвољни излазни ток
* Класа VisualizedFlight : public Fl\_Widget – представља графичку репрезентацију једног лета у програму

1. атрибути
   * Flight flight - лет
   * int unsortedIndex – позиција лета у фајлу
   * int sortedIndex – позиција лета након сортирања
   * int numOfFlightsTotal – укупан број летова
   * Fl\_Color boja – боја компоненте

[2] методе

* + void draw () – исцртава компоненту на прозору
  + void write (const char\* s) – исписује текстуални садржај преко компоненте
  + void move (int jean, int dva) – помера компоненту
  + bool operator = (VisualizedFlight &vf) – пореди две компоненте на основу њиховог лета
* Класа Instruction – представља једну недељиву надердбу која се мора извршити како си се сортирање визуализовало

[1] атрибути

* VisualizedFlight \*vf – показивач на графичку компоненту над којом инструкција треба да се изврши

[2] методе

* virtual void do\_it () = 0 – виртуелна метода која представља имплементацију саме инструкције
* Класа ColorChange :public Instruction – промена боје једне компоненте приказене на главном прозору

[1] атрибути

* Атрибути родитељске класе
* Fl\_Color color – боја у коју компонента треба да се обоји

[2] методе

* void do\_it () – имплементације своје функционалности
* Класа ColorSwap : public Instruction - замена боје две компоненте приказене на главном прозору

[1] атрибути

* Атрибути родитељске класе
* VisualizedFlight \*vf2 – показивач на другу графичку компоненту над којом инструкција треба да се изврши

[2] методе

* void do\_it () – имплементације своје функционалности
* Класа WidgetSwap - замена места двема компонентама на главном прозору

[1] атрибути

* Атрибути родитељске класе
* VisualizedFlight \*vf2 – показивач на графичку компоненту над којом инструкција треба да се изврши
* Fl\_Color color1 – боја којом треба да се обоји друга компонента
* Fl\_Color color2 – боја којом треба да се обоји друга компонента

[2] методе

* void do\_it () – имплементације своје функционалности
* class RevertColor: public Instruction – бојење колекције компоненти у њихову почетну боју

[1] атрибути

* vector<VisualizedFlight\*> vector – колекција показивача на компоненте које треба да се обоје
* Fl\_Color color – боја у коју треба да се обоје

[2] методе

* void do\_it () – имплементације своје функционалности
* Класа Sort

[1] приватни атрибути

* unsigned long num\_cmps – број поређења приликом сортирања
* unsigned long num\_swaps - број замена елемената

[2] методе

* Гетери и сетери
* virtual void sort(vector<VisualizedFlight\*> &data, Tip, list<Instruction\*> &instructions) = 0 - виртуелна метода коју треба класе наследнице да имплеметирају
* class SelectionSort : public Sort

[1] методе

* void sort(vector<VisualizedFlight\*> &data, Tip, list<Instruction\*> &instructions) – преклопљена метода свог родитеља која омогућава сортирање вектора елемената Selection sort алгоритмом
* Класа QuickSort: public Sort

[1] методе

* void sort(vector<VisualizedFlight\*> &data, Tip, list<Instruction\*> &instructions) – преклопљена метода свог родитеља која омогућава сортирање вектора елемената Quick sort алгоритмом
* void quickSort(vector<VisualizedFlight\*> &data, list<Instruction\*> &instructions, int low, int high, Tip en)
* int partition(std::vector<VisualizedFlight\*> &a, list<Instruction\*> &instructions,int i ,int j,Tip en) – враћа позицију на коју треба да се уметне пивот
* Класа IO – класа задужена за улазно излазни систем програма

[1] атрибути

* string inPath – улазни фајл
* string outPath – излазни фајл
* string zaglavlje – излазни резултат сортирања који се уписује у излазну датотеку
* string crit – критеријум сортирања
* string type – тип алгоритма за сортирање
* vector<Flight> letovi
* vector<VisualizedFlight> viz

[2] методе

* void check\_arguments (int argc, char\* argv[]) – провера аргумената командне линије
* void loadFlights () – учитавање летова из улазног фајла
* void writeFlights(vector<VisualizedFlight\*> &v, string header) – уписивање сортираних летова у излазни фајл
* Класа MyWindow : Window

[1] приватни атрибути

* bool sorting - да ли је сортирање у току
* bool sorted – да ли је сортирање завршено
* vector<VisualizedFlight> mainVector
* IO inOut – изалазно улазна компонента

[2] јавни атрибути

* CheckBox компоненте помоћу којих се контролише критеријум и начин сортирања
* Slider slider – избор брзине визуелизације сортирања
* float plotSpeed – вредност горенадеведене компоненте
* Button компоненте којима се контролише сама визуелизација сортирања
* list<Instruction\*> instrukcije – листа инструкција извршених приликом сортирања елемената
* vector<VisualizedFlight\*> вектори који садрже графичке компоненте у зависности од изабраног критеријума сортирања
* Tip criteria – тренутно изабрани критеријум сортирања

[3] методе

* Callback методе за графичке компоненте прозора
* bool wait\_for\_button() – позива петљу графичке компоненте
* void fill() – добављање информација о летовима из УИ компоненте
* bool criteriaCheck() – провера да ли су сви критеријуми задовољени за почетак сортирања
* bool initializeSort() – инцијализација сортирања
* void displayStat() – повратна информација о завршеном сортирању
* void draw() – исртавање прозора

**V Структуре аргумената командне линије и пример коришћења програма**

Аргументи командне линије треба да имају следећу структуру

naziv.exe putanja1.txt putanja2.txt nacin\_sortiranja algoritam

где је:

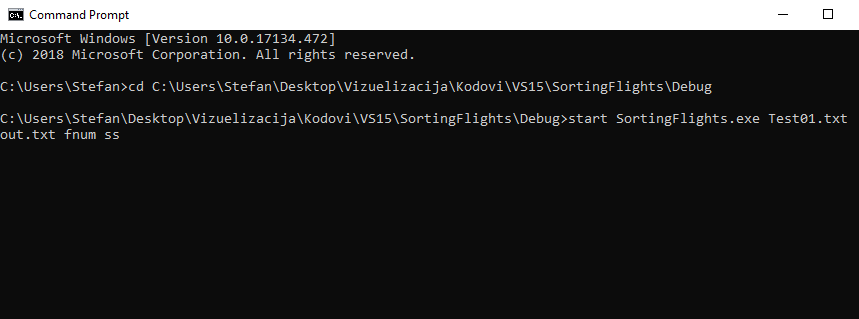
1. Први аргумент име извршног фајла
2. Други аргумент је путања до фајла из ког учитавамо податке о летовима
3. Трећи аргумент је путања до фајла у који уписујемо податке о сортираним летовима
4. Четврти критеријум сортирања летова ( “fnum” за сортирање по броју лета, “dep” за сортирање по времену одлска, “dest” по дестинацији, “gate” по капији)
5. И пети алгоритам сортирања ( “ss” за Selection Sort , и “qs” за Quick Sort )

Програм може да ради нормално са било којим бројем унесених аргумената (једино је извршни фајл обавезан аргумент)

Ако је неки аргумент неправилно унесен он ће бити игнорисан

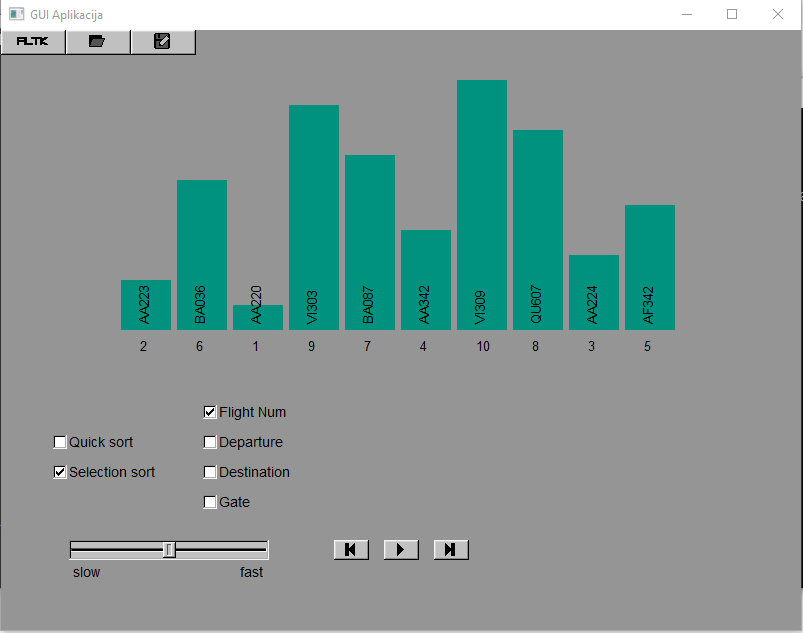
Позиционирањем у директоријум где се налази извршни фајл, и покретањем команде :

start SortingFlights.exe Test01.txt out.txt fnum ss



Доћи ће до самог покретања програма где ће :

* Као улазни фајл бити изабран *Test01.txt*
* А као излазни *out.txt*
* Критеријум сортирања ће бити број лета
* А алгоритам сортирања Selection sort



**Структура излазног фајла:**

-Коришћен алгоритам : Х

-Критеријум сортирања : Х

-Очекивани број поређења за алгоритам: Х

-Број поређења: Х

-Броја замена елемената: Х

-сортирани лет1

-сортирани лет 2

...

....

**VI Опис алгоритама сортирања**

* Selection Sort

Као први алгоритам сортирања коришћен је Selection sort. За сваку колекцију независно од распореда елемената он сортирање врши квадратном сложеношћу.

Прецизније, број поређења ће сваки пут износити , где је n број елемената колекције.

У свакој итерацији овог алгоритма крећемо од *i*-тог елемента (0 ≤ *i* ≤ n), и идемо до краја колекције тражећи најмањи елемент.

Када дођемо до краја колекције, елемент који смо нашли заменимо са

*i*- тим елементом.

Пошто имамо *n* елемената, сваки пут ћемо имати *n* итерација.

* Quick Sort

Као други алгоритам сортирања коришћен је Quick sort, пре свега због своје in-place особине, односно не креирања никаквих помоћних структура ради сортирања колекције. Његов најгори случај сортирања ,али у просеку он ради боље од алгоритама чије је време

Алгоритам чине три једноставна корака који се одвијају рекурзивно:

1. Одређивање пивота, тј елемента са којим ћемо поредити све остале елементе из колекције
2. Следеће, користећи пивот, колекцију делимо на два дела, лево од пивота су сви елементи мањи од њега, а десно сви елементи већи од њега
3. Понављамо алгоритам над овим партицијама док колекција није потпуно сортирана

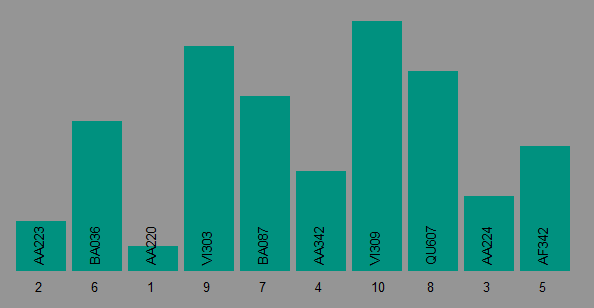
**VII Напредни ООП Концепти**

* Наслеђивање
  + Класа MyWindow наслеђује класу Window која наслеђује класу Fl\_Window из FLTK библиотеке
  + Структуре CheckBox и Slider које наслеђују класу Widget дефинисану у gui.h
  + Класа VisualizedFlight наслеђује класу Widget
  + Класе ColorChange, ColorSwap, WidgetSwap, RevertColor наслеђују апстрактну класу Instruction
* Преклапање оператора
  + Класе Flight и VisualizedFlight имају преклопљен оператор = који служи за поређење две инстанце исте класе
  + Класа има преклопљене оператора за читање и писање у произвољни ток података
* Полиморфизам
  + При визуализацији сортирања, графичка компонента има листу list<Instruction\*> класе Instruction и позивањем њене методе do\_it() се реализује визуелизација, али пошто је ова класа апстрактна, овај вектор чине њене класе наследнице и позивањем методе do\_it() се позива метода коју је свака од класа наследница преклопила тако да обавља различите функционалности
* Енкапсулација
  + При сортирању летова, класа Sort не зна како се елементи пореде, већ о томе брине enum Tip и сама класа Flight

**VIII Графичка спрега**

Графичку корисничку спрегу чини прозор са следећим компонентама :

1. На централном делу прозора се налазе компоненте које представљају летове, то су правоугаони елементи са одређеним атрибутом лета и позицијом коју ће тај елемент заузимати након сортирања.



Ако летови нису учитани овај део прозора ће бити празан

1. Мени бар

На самом врху прозора се налази мени бар који се састоји од 3 дугмета:

1. Дугме за приказ основних информација о аутору
2. Дугме за избор улазног фајла из кога се летови читају
3. Дугме за избор излазног фајла, када је улазни фајл изабран ово дугме служи за упис резултата сортирање у тај фајл

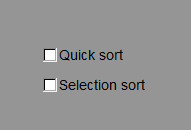


1. Поља за избор алгоритама сортирања

Испод самих летова налази се прва група поља, тзв поља за избор алгоритма сортирања.

Састоји се од два поља :

* + Поље за избор Quick Sort алгоритма
  + Поље за избор Selection Sort алгоритма

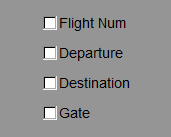


1. Поља за избор критеријума сортирања

Друга група поља испод летова су поља којима бирамо у односу на које облележје ћемо сортирати летове

Састоји се од 3 поља :

* + Поље за сортирање у односу на број лета
  + Поље за сортирање у односу на време полетања
  + Поље за сортирање у односу на дестинацију лета
  + Поље за сортирање у односу на број капије



1. Слајдер за подешавање брзине

Испод наведених поља се налази слајдер чијом манипулацијом можемо мењати саму брзину визуализације сортирања



1. Дугмад за контролисање саме визуализације

На самом дну прозора се налазе 3 дугмета којима се цео процес визуализације контролише.

Састоји се од три дугмета, које са лева на десно обављају следеће фунцкионалности:

1. Дугме за враћање сортираних летова у њихов несортиран облик учитан из фајла
2. Дугме за приказ следеће инструкције сортирања
3. Дугме за приказ свих инструкција до краја сортирања



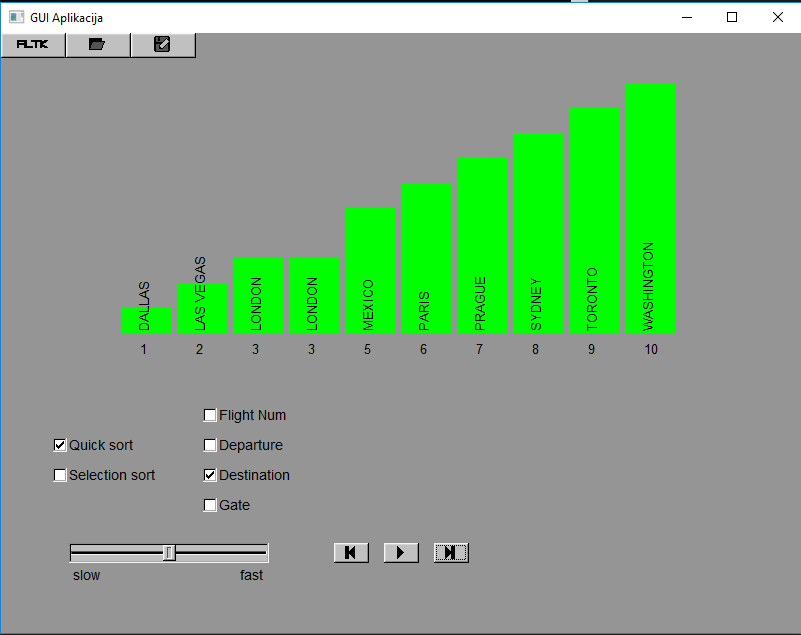
**IX Тестови**

* Тест 01 - унапред задати тест случај

Структура улазне датотеке :

|  |  |  |  |
| --- | --- | --- | --- |
| Flight number | Departure | Destination | Gate |
| AA223  BA036  AA220  VI303  BA087  AA342  VI309  QU607  AA224  AF342 | *21:00*  *20:30*  *19:00*  *17:45*  *16:00*  *13:20*  *08:30*  *08:20*  *07:45* | Las Vegas  Dallas  London  Mexico  London  Paris  Prague  Toronto  Sydney  Washington | A3  A3  B4  B4  B4  A7  F2  F2  A7  A3 |

Изглед програма након сортирања летова у односу на дестинацију:

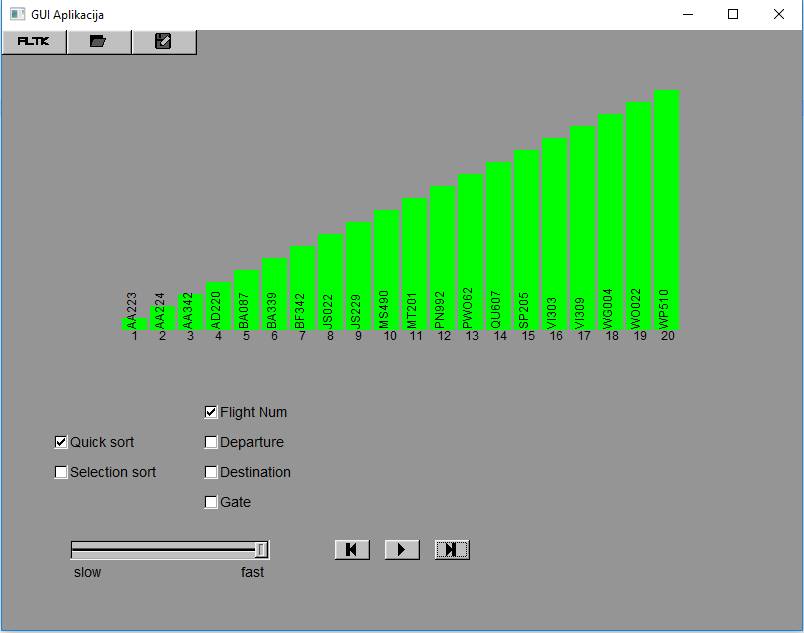


* Тест 02 – датотека са двадесет летова летова

Структура улазне датотеке :

|  |  |  |  |
| --- | --- | --- | --- |
| Flight number | Departure | Destination | Gate |
| MS490  VI303  PWO62  AA223  AD220  JS229  AA342  VI309  QU607  BA339  BF342  AA224  JS022  BA087  WO022  WP510  SP205  PN992  WG004  MT201 | *21:00*  *19:00*  *22:20*  *21:15*  *20:30*  *13:30*  *16:00*  *13:20*  *8:30*  *8:20*  *7:45*  *12:45*  *13:25*  *17:45*  *19:30*  *21:00*  *17:15*  *7:15*  *12:30*  *21:45* | Montana  Mexico  Charlotte  Las Vegas  London  New York  Paris  Prague  Toronto  Sydney  Washington  Los Angeles  Ontario  London  Memphis  Boston  New Hampshire  Seattle  Denver  Los Angeles | A3  B4  C1  A3  B4  C1  A7  F2  F2  A7  A3  A1  A7  B4  C2  D7  B3  C2  A3  A1 |

Изглед програма након сортирања ових летова у односу на број лета:

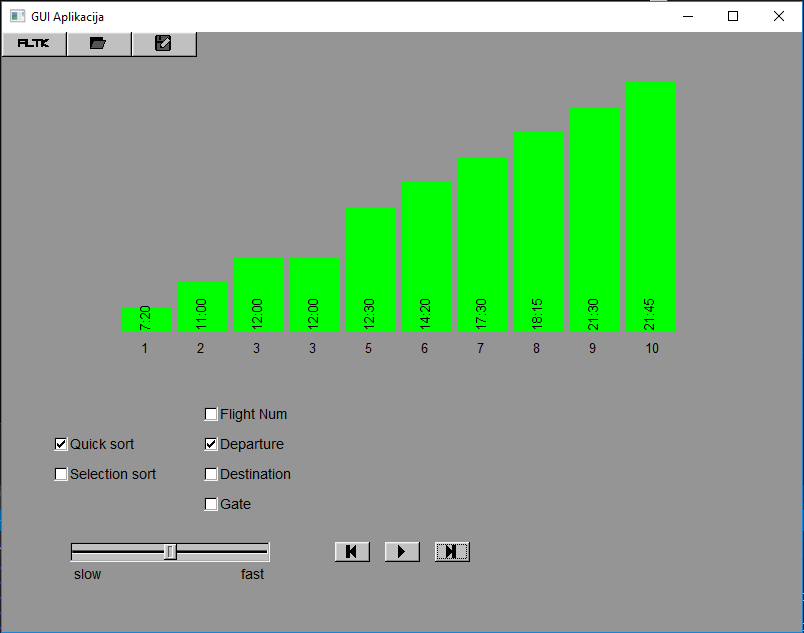


* Тест 03 – пивот елемент у првом рекурзивном позиву је средишњи елемент

Структура улазне датотеке :

|  |  |  |  |
| --- | --- | --- | --- |
| Flight number | Departure | Destination | Gate |
| BD001  AA220  AA130  BC325  AA150  CD425  CC322  BD215  DM321  DM400 | *12:30*  *7:20*  *11:00*  *12:00*  *12:00*  *14:20*  *18:15*  *21:45*  *17:30*  *21:30* | Frankfurt  Barbados  Abu Dhabi  Dubai  Copenhagen  Jerusalem  Rome  Genoa  Helsinki  Stara Pazova | B1  B2  A5  C2  A1  C1  A7  B3  C2  A7 |

Изглед програма након сортирања ових летова у односу на време поласка:

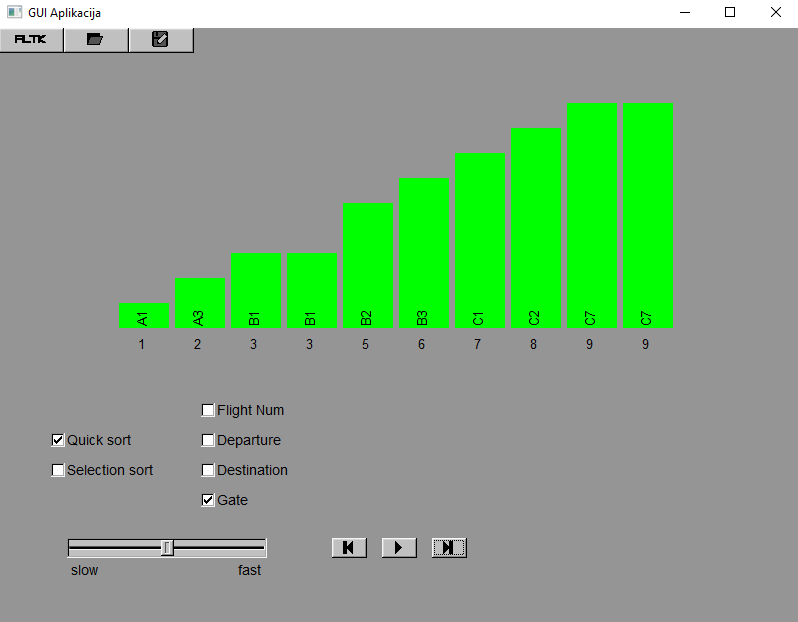


* Тест 04 – скоро сортирана колекција летова

Структура улазне датотеке :

|  |  |  |  |
| --- | --- | --- | --- |
| Flight number | Departure | Destination | Gate |
| AA130  AA220  AA150  BC325  BD001  BD215  CD425  DM321  CC322  DM400 | *11:00*  *7:20*  *12:00*  *12:00*  *12:30*  *21:45*  *14:20*  *17:30*  *18:15*  *21:30* | Abu Dhabi  Barbados  Copenhagen  Dubai  Frankfurt  Genoa  Jerusalem  Helsinki  Rome  Stara Pazova | A1  B1  A3  B2  B1  B3  C1  C2  C7  C7 |

Изглед програма након сортирања ових летова у односу на број капије:

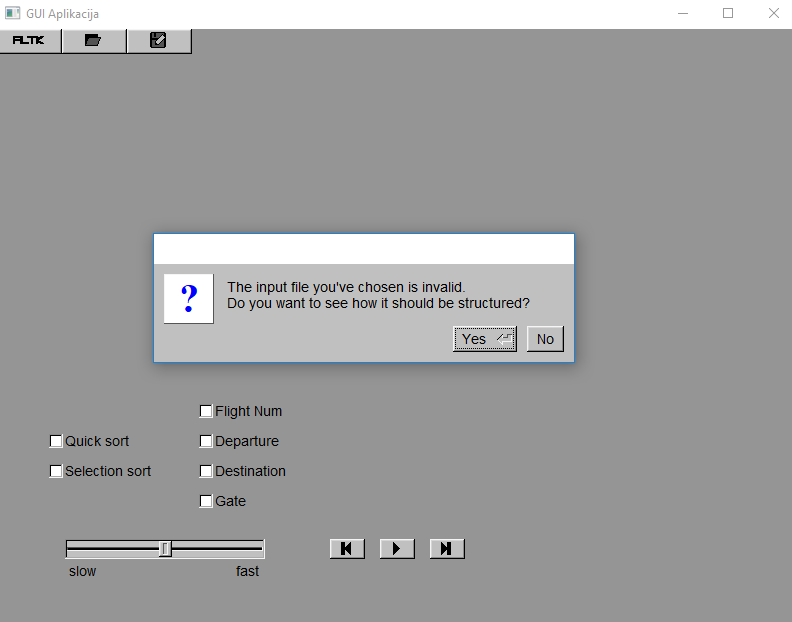


* Тест 05 – празна улазна датотека

Структура улазне датотеке :

|  |  |  |  |
| --- | --- | --- | --- |
| Flight number | Departure | Destination | Gate |
|  |  |  |  |

Изглед програма након избора ове датотеке:



Избором “Yes” опције се отвара овај фајл и корисник може прочитати како да структуира улазну датотеку.

**X Излазни резултати**

* Тест01 :

|  |  |  |
| --- | --- | --- |
| Selection sort | Број поређења | Број замена |
| Број лета | 45 | 7 |
| Време поласка | 45 | 5 |
| Дестинација | 45 | 3 |
| Капија | 45 | 5 |
| Просек | 45 | 5 |

|  |  |  |
| --- | --- | --- |
| Quick sort | Број поређења | Број замена |
| Број лета | 25 | 9 |
| Време поласка | 45 | 5 |
| Дестинација | 31 | 3 |
| Капија | 31 | 7 |
| Просек | 33 | 6 |

Као што можемо приметити, Quick sort има знатно мање поређења у односу на Selection sort уз занемарљиво већи број замена елемената.

Али при сортирању у односу на време поласка, оба алгоритма имају исте перформансе, из разлога зато што ће пивотски елемент у сваком рекурзивном позиву бити највећи или најмањи елемент и Quick sort неће моћи да искористи своју предност што бирањем добром пивот елемента дели колекцију на два дела која су скоро сортирана и тим убрза процес сортирања.

* Тест02 :

|  |  |  |
| --- | --- | --- |
| Selection sort | Број поређења | Број замена |
| Број лета | 190 | 18 |
| Време поласка | 190 | 18 |
| Дестинација | 190 | 15 |
| Капија | 190 | 15 |
| Просек | 190 | 16.5 |

|  |  |  |
| --- | --- | --- |
| Quick sort | Број поређења | Број замена |
| Број лета | 58 | 24 |
| Време поласка | 70 | 31 |
| Дестинација | 59 | 26 |
| Капија | 87 | 18 |
| Просек | 68.5 | 24.75 |

Тек сад са више од 10 елемената можемо видети колико је уствари Quick sort потребно мање поређења да би колекцију сортирао.

Опет можемо примети да Quick sort има већи број замена елемената, али у односу на побољшање при броју поређења можемо их заменарити.

* Тест03

|  |  |  |
| --- | --- | --- |
| Selection sort | Број поређења | Број замена |
| Број лета | 45 | 4 |
| Време поласка | 45 | 7 |
| Дестинација | 45 | 4 |
| Капија | 45 | 6 |
| Просек | 45 | 5.25 |

|  |  |  |
| --- | --- | --- |
| Quick sort | Број поређења | Број замена |
| Број лета | 19 | 4 |
| Време поласка | 21 | 6 |
| Дестинација | 19 | 6 |
| Капија | 19 | 9 |
| Просек | 19.5 | 6.25 |

Број поређења за Selection sort је стандардано , али перформансе Quick sort-а су сада боље чак и од .

То смо постигли бирањем средишњег елемента за пивот у првом рекурзивном позиву.

* Тест04

|  |  |  |
| --- | --- | --- |
| Selection sort | Број поређења | Број замена |
| Број лета | 45 | 3 |
| Време поласка | 45 | 5 |
| Дестинација | 45 | 1 |
| Капија | 45 | 2 |
| Просек | 45 | 2.75 |

|  |  |  |
| --- | --- | --- |
| Quick sort | Број поређења | Број замена |
| Број лета | 36 | 3 |
| Време поласка | 37 | 5 |
| Дестинација | 43 | 1 |
| Капија | 33 | 2 |
| Просек | 37.25 | 2.75 |

Још у првом тест случају смо видели да Quick sort има проблем са већ сортираним колекцијама.

Сада када имамо скоро сортирану колекцију ситуација је слична, те можемо закључити да што је колекција “сортиранија” то Quick sort све више почиње да личи на Selection sort, и када је колекција потпуно сортирана алгоритми раде идентично, с тим што се Quick sort позива рекурзивно што је за рачунар захтевније од итеративног приступа Selection sort-а.

**XI Уочени проблеми и ограничења**

Први проблем је било усклађивање звучних ефеката како би визуализација осим графичке имала и звучну компоненту.

Наиме, пошто је брзина визуализације кориснички задата променљива вредност, проблем је био ускладити звук тако да он подједнако прецизно ради на оба спректрума брзине.

Оптимално решење би била и променљива дужина звучног сигнала у зависности од брзине визуализације, али за сада то није имплементирано.

Звук је подешен тако да најбоље ради за неке средње вредности брзине визуализације, те да на крајевима спректрума буде подједнако лош.

Други проблем је представљало губљење фрејмова од стране главног прозора при брзој визуализацији.

Приликом саме визуализације се понекад може приметити, како се прозор исцртава услед неколико узастопних позива Fl::flush() методе.

Овај проблем би се могао решити употребом multithreading -а али услед недовољних знања о тој теми као и временског ограничења то такође још увек није имплементирано.

Ограничења:

1. Коришћење ФЛТК библиотеке за графички део
2. Улазна датотека мора задовољавати структуру иначе летови неће бити учитани (заглавље датотеке је опционо)
3. Пошто се ширина widget-а рачуна као број летова подељен са 500, ако се у датотеци налази више од 500 летова, видгети ће бити ширине мање од 1 (неће бити видљиви на прозору)
4. Иако ће програм подржати фајлове са до 500 летова, већ приликом бројке од око отприлике 100 долази до проблема да су widget-i прилично мали и није лако видети који лет они представљају
5. Алгоритми су само Selection sort и Quick sort
6. Имамо само 4 податка о појединачном лету те и 4 критеријума сортирања