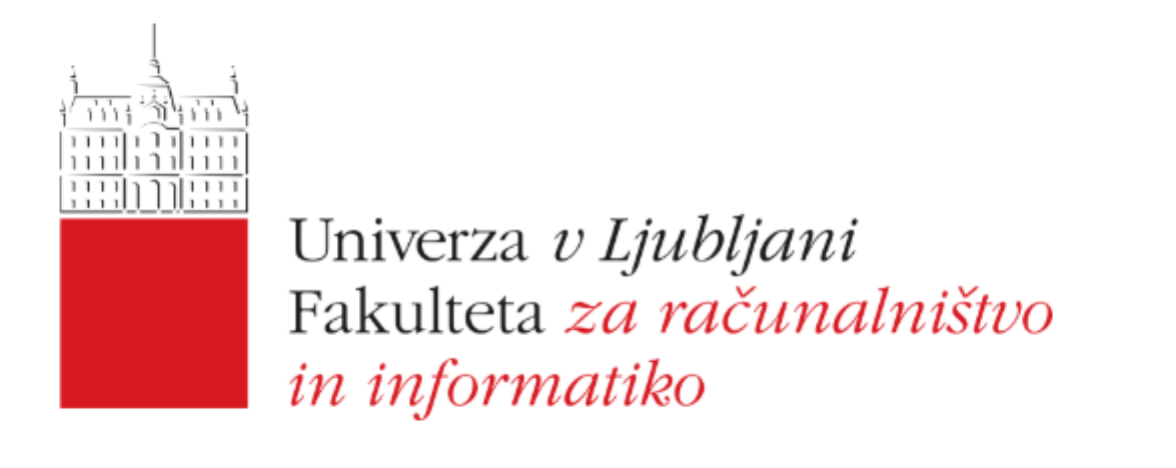


Opis sistema in osnovna navodila za uporabo

Tomaž Dobravec



KAZALO

1 Opis sistema 3

1.1 Namen 3

2 Delovanje sistema 3

2.1 ALGator kot samostojna aplikacija na lokalnem računalniku 4

2.2 ALGator v vsebniku sistema Docker 4

3 Izvajanje algoritmov 5

3.1 Definicija osnovnih pojmov 5

3.2 Naloge in množice nalog 6

3.3 Algoritmi 9

3.4 Rešitev naloge in zapis rezultatov 9

3.4.1 Zapis rezultatov 10

3.5 Meritve 11

3.5.1 Meritve časovnih in specifičnih indikatorjev izvajanja 11

3.5.2 Meritve s števci 12

3.5.3 Meritve s prirejenim JVM 13

4 Konfiguracija sistema 14

# Opis sistema

ALGator je sistem za izvajanje algoritmov na podanih testnih podatkih ter analizo rezultatov izvajanja.

Sistem omogoča dodajanje in upravljanje poljubnega števila projektov. V okviru enega projekta je definiran problem, testne množice vhodnih podatkov ter način reševanja nalog tega problema. Projekt lahko vsebuje poljubno število algoritmov, ki naloge rešujejo na predpisan način. Sistem omogoča analizo izvajanja posameznega algoritma ter primerjavo med algoritmi istega projekta.

## Namen

* Reševanje definiranih problemov z različnimi algoritmi.
* Analiza delovanja posameznega algoritma pri reševanju nalog danega problema.
* Primerjava različnih algoritmov za reševanje istega problema.

# Delovanje sistema

ALGator je zbirka orodij (javanskih razredov), ki so shranjeni v datoteki ALGator.jar. Najpomembnejša orodja zbirke so: Version (izpis verzije in osnovnih podatkov sistema), Admin (administracija sistema), Execute (izvajanje algoritmov), Analyse (analiza rezultatov izvajanja), … Za pravilno delovanje sistema se mora zbirka orodij ALGator.jar nahajati v poti (CLASSPATH). Posamezno orodje poženemo z ukazom, kot na primer: java algator.Execute.

Za delovanje sistema je pomembna tudi mapa (folder) <algator\_root>. V tej mapi (in njenih podmapah) so shranjeni podatki o konfiguraciji sistema in podatki o projektih. Pot do mape <algator\_root> sistemu podamo s pomočjo okoljske spremenljivke ALGATOR\_ROOT (ali z uporabo stikala –r ob klicu posameznega orodja).

ALGator lahko uporabljamo na dva načina: kot samostojno aplikacijo na lokalnem računalniku ali kot aplikacijo, ki teče v vsebniku sistema Docker. Prvi način je primeren predvsem za razvijalce sistema ALGator, medtem ko je za končne uporabnike (razvijalce algoritmov) primernejši drugi.

## ALGator kot samostojna aplikacija na lokalnem računalniku

Za pravilno delovanje sistema, je potrebno:

1. ustvariti lokalno mapo <algator\_root> ter podmapi app in data\_root;
2. pridobiti program ALGator (ALGator.jar + folder lib) in ga posneti v mapo app; povezava: https://github.com/ALGatorDevel/Algator/blob/master/dist/ALGator.zip
3. pravilno nastaviti sistemske spremenljivke
   * + CLASSPATH (dodati je treba pot do ALGator.jar) in
     + ALGATOR\_ROOT (kazati mora na mapo <algator\_root>);
4. ustvariti vsaj en projekt (glej poglavje 3.6).

Po namestitvi lahko orodja sistema ALGator izvajamo iz lupine z ukazi

**java algator.Version**

izpis sporočila o verziji ALGatorja in privzeti nastavitvi folderjev

ALGATOR\_ROOT, ALGATOR\_DATA\_ROOT in ALGATOR\_DATA\_LOCAL

**java algator.Admin**

ustvarjanje konfiguracijskih datotek novega projekta/algoritma

**java algator.Execute**

izvajanje definiranih algoritmov izbranega problema

**java algator.Analyse**

analiza rezultatov izvajanja

## ALGator v vsebniku sistema Docker

ALGator lahko uporabljamo tudi tako, da na lokalnem računalniku namestimo sistem Docker in v njem poženemo sliko algatorweb. Ob pravilni namestitvi (navodila so navedena na githubu - <https://github.com/ALGatorDevel/Algator>) se na lokalni računalnik prenesejo skripte, ki omogočajo dostop do ALGatorjevih orodij. Ta se bodo v tem primeru izvajala znotraj vsebniku sistema Docker, vsi podatki (o konfiguraciji sistema in o projektih) pa se bodo zapisovali v mapo <algator\_root>, ki se nahaja na lokalnem računalniku in je preko okoljske spremenljivke ALGATOR\_ROOT povezana z vsebnikom.

# Izvajanje algoritmov

## Definicija osnovnih pojmov

* **KONFIGURACIJSKA DATOTEKA**

Vse entitete sistema ALGator so opisane v konfiguracijskih datotekah (ena datoteka opisuje eno entiteto). Format konfiguracijskih datotek je JSON.

Primer: izsek datoteke, ki opisuje projekt:

{

"Project" : {

"Name" : "Sorting",

"Description" : "Internal sorting of integer arrays",

"Algorithms" : ["Hoare","Wirth"],

"TestSets" : ["TestSet1","TestSet2"],

...

}

}

* **SPREMENLJIVKA (variable)**

Spremenjlivka, ki jo uporabljamo za opis neke lastnosti. Vsaka spremenljivka ima svoje ime, tip in vrednost. Spremenljivka je lahko zapisana v konfiguracijski datoteki ali pa »živi« znotraj javanskega razreda.

* **PARAMETER**

Spremenljivka, ki opisuje lastnost naloge. Primer: parameter N opisuje velikost tabele, ki jo želimo urediti.

* **INDIKATOR**

Spremenljivka, ki opisuje lastnost izvajanja algoritma pri dani nalogi. Primer: indikator Tmin opisuje čas izvajanja algoritma pri urejanju tabele.

* **PROBLEM**

Problem, ki ga rešujemo. Primer: urejanje podatkov, trgovski potnik, stiskanje datotek, linearno programiranje, …

* **PROJEKT**

Projekt je skupek konfiguracijskih datotek, s katerimi v sistemu ALGator opišemo problem in pripadajoče algoritme.

* **NALOGA (input)**

Naloga danega problema. Primer za problem urejanja podatkov: ena tabela, ki jo je potrebno urediti.

* **MNOŽICA NALOG (test set)**

Zbirka naloge, ki predstavljajo najmanjšo enoto reševanja (ob enem zagonu sistema ALGator rešimo eno množico nalog).

* **ALGORITEM**

Javanski razred, ki reši eno nalogo danega problema.

* **IZVRŠEVALEC (executor)**

Orodje, s katerim poženemo dani algoritem nad dano množico nalog. Gre za javanski razred, ki je zapakiran v JAR datoteko. Izvrševalec poženemo iz ukazne vrstice; pri tem podamo parametre, ki opisujejo projekt, algoritem in množico nalog. Primer klica:

java algator.Execute Sorting -a QuickSort -t TestSet1

(ob izvršitvi tega klica se bo ALGator ”sprehodil” čez vse naloge testne množice TestSet1 in z vsako nalogo pognal algoritem QuickSort)

* **REŠITEV NALOGE**

Skupek parametrov in indikatorjev, ki opisujejo obnašanje algoritma pri reševanju dane naloge (velikost problema, poraba časa, pravilnost rezultata, poraba virov, ...)

* **REŠITEV MNOŽICE NALOG**

Datoteka, ki vsebuje rešitve za vse naloge dane množice nalog v CSV obliki: rešitev vsake naloge je zapisana v svoji vrstici, parametri in indikatorji znotraj vrstice so ločeni s podpičjem. Vrstni red parametrov in indikatorjev je določen z ustreznima attc in atrd datotekama.

## Naloge in množice nalog

* ALGator z danim algoritmom reši vse naloge dane množice nalog.
* Za opis ene naloge v ALGatorju uporabljamo razred AbstractInput, ki vsebuje osnovne podatke o vhodu (ime naloge, velikost naloge, ...). Ker se opisi nalog posameznih problemov med seboj močno razlikujejo, mora skrbnik projekta napisati razred

class [Project]Input extends AbstractInput {…}

v katerem predvidi podatkovne strukture za shranjevanje podatkov o nalogi.

Primer: pri projektu Sorting (urejanje podatkov) za shranjevanje podatkov v razredu SortingInput definiram tabelo

public int [] arrayToSort;

* Podobno kot za vhod tudi za izhod algoritma velja, da je močno odvisen od problema. Zato mora skrbnik projekta napisati tudi razred

class [Project]Output extends AbstractOutput {…}

v katerem definira podatkovne strukture za shranjevanje podatkov o izhodu.

V primeru projekta Sorting je izhod podoben vhodu (ena tabela), zato sta tudi razreda SortingInput in SortingOutput podobna:

class SortingOutput extends AbstractOutput {

public int [] sortedArray;

...

}

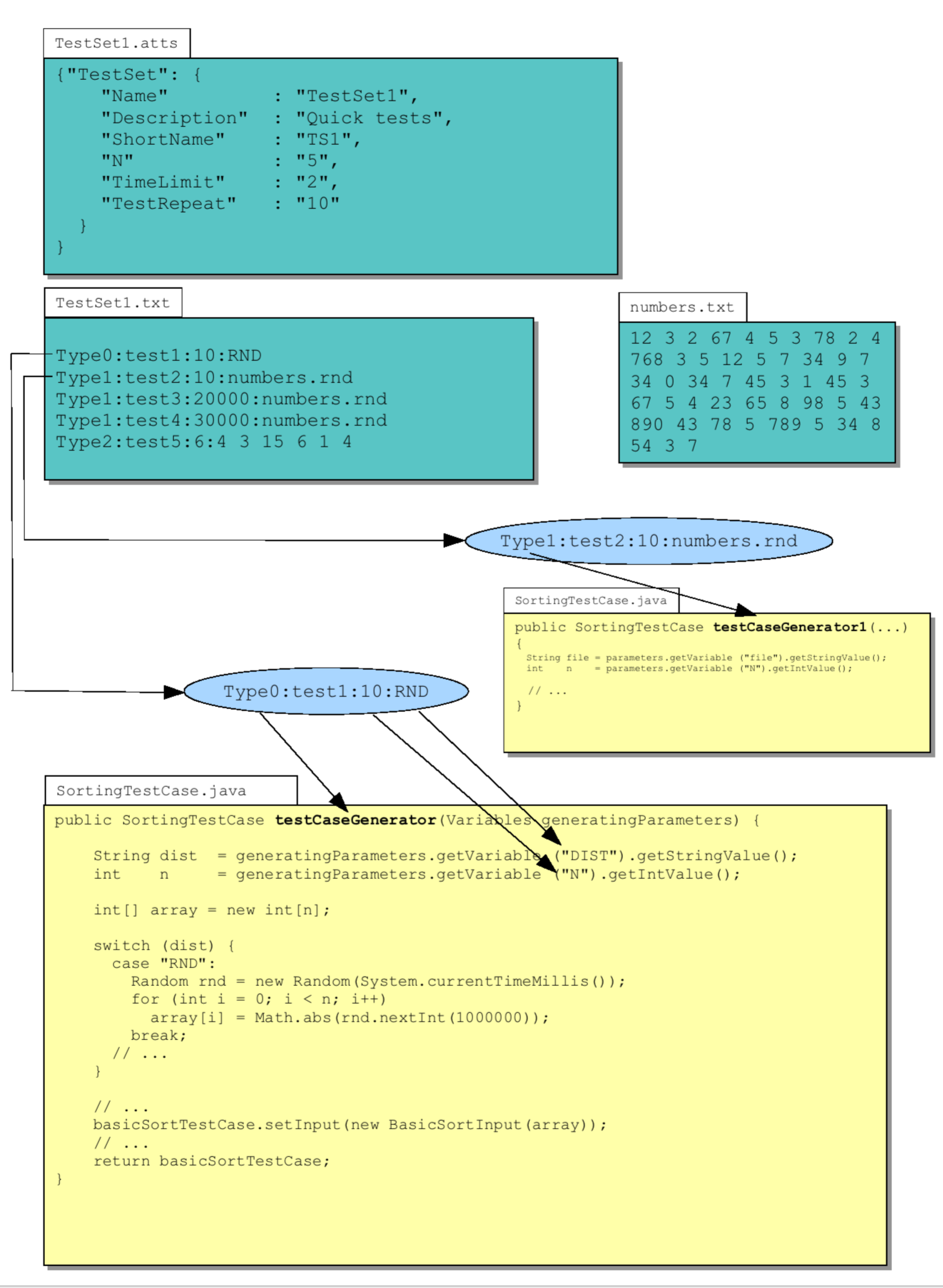
* ALGator z algoritmi, podanimi v projektu, rešuje naloge, ki so zapisane v t.i. testnih množicah. Vsaka testna množica vsebuje eno ali več nalog (število nalog v testni množici določi skrbnik projekta, običajne testne množice vsebujejo od 10 do 100 nalog). Kakšen vhod bo dobil algoritem, je odvisno od problema (primer: naloga faktorizacije je podana z eno celoštevilsko vrednostjo, naloga urejanja s tabelo števil, naloga ZIP pa je podana z datoteko), zato je tudi zapis vhoda v testnih množicah odvisen od problema. Skrbnik projekta (ki edini ve, kakšen vhod bodo dobili algoritmi tega problema) določi, kako se bo vhod posamezne naloge zapisal v testne množice.

ALGator predpostavlja naslednje:

* + Za vsako testno množico [Testset] sta v direktoriju tests danega projekta dve datoteki: [Testset].atts in [Testset].txt.
  + V direktoriju tests (ali v poddirektorijih, ki jih po lastni presoji ustvari skrbnik projekta) se nahajajo vse datoteke, ki jih potrebujemo za izvajanje testov (vhodni podatki). Ko se v konfiguracijskih datotekah sklicujemo na te datoteke, uporabimo relativne poti (glede na direktorij tests).
  + Datoteka [Testset].atts je konfiguracijska datoteka, v kateri je podano ime in opis ter število nalog te testne množice.
  + Datoteka [Testset].txt opisuje posamezne naloge - **vsaka naloga je opisana v eni vrstici te datoteke**. Ker je opis naloge močno odvisen od problema, je format vrstice te datoteke povsem odvisen od projekta. Format določi skrbnik projekta.

Skrbnik projekta mora določiti, kako bo na podlagi podatkov, zapisanih v vrstici datoteke [TestSet].txt, ustvaril testni primer. V ta namen mora ustvariti vsaj en generator – napisati mora metodo testCaseGenerator(), ki na podlagi vrednosti, zapisanih v tej vrstici, ustvari testni primer.

Na spodnji sliki je prikazano, kako se na podlagi podatkov zapisanih v [TestSet].txt ustvarijo testni primeri.



Slika 1: Primer zapisa nalog in implementacije metode testCaseGenerator() za problem Sorting.

Sistem ALGator ob izvajanju testne množice bere datoteko [TestSet].txt, vrstico po vrstici. Na podlagi prvega podatka v vrstici (Type0, Type1, …) izbere generator, ki bo ustvaril testni primer in pokliče ustrezno metodo (testCaseGenerator(), testCaseGenerator1(), …). Vrstni red in pomen parametrov (ostali podatki v vrstici) je za vsak generator posebej zapisan v datoteki attc.

## Algoritmi

Algoritem je implementiran kot naslednik razreda [Project]AbsAlgorithm. Srce algoritma je metoda execute(), ki kot vhod prejme objekt razreda [Project]Input, vrniti pa mora objekt razreda [Project]Output. Čas izvajanja metode execute() se šteje kot čas izvajanja algoritma.

Primer: v projektu Sorting je preprost algoritem (ki uporablja kar obstoječo javansko metodo sort()) definiran takole:

|  |
| --- |
| public class **JavaSortAlgorithm** extends **SortingAbsAlgorithm** {  @Override  protected **SortingOutput** execute(**SortingInput** testCase) {  doSort(testCase.arrayToSort);  return new SortingOutput(testCase.arrayToSort);  }    private void doSort(int[] data) {  java.util.Arrays.sort(data);  }  } |

## Rešitev naloge in zapis rezultatov

Rešitev ene naloge je sestavljena iz parametrov in indikatorjev.

Parametri opisujejo lastnosti naloge (na primer: velikost problema N, skupina problema, ...). Zapisani so v datoteki, ki opisuje test ([Testset].txt). Za lažjo analizo rezultatov lahko vrednost parametrov prenesemo tudi v datoteko z rezultati testiranje (primer: parameter o velikost vhoda uporabimo za urejanje rezultatov glede na velikost vhoda ali za združevanje rezultatov z isto velikostjo vhoda).

Indikatorji opisujejo lastnosti izvajanja algoritma na danem testnem primeru. Indikatorje delimo na

* Specifični indikatorji

Specifični indikatorji opisujejo specifične lastnosti algoritma. Primer: numerična rešitev naloge, kakovost rešitve, pravilnost rešitve, ... (compresionRate, myResult/optResult, OK/NOK, ..). Vrednost specifičnih indikatorjev se določi v metodi [Project]Output.getIndicatorValue().

* Časovni indikatorji

Časovni indikatorji opisujejo čas(e) izvajanja. Osnovni časovni indikator meri čas izvajanj celotnega algoritma (čas izvajanja metode [Project]AbsAlgorithm.execute()).

* Števci

Števci povejo, kolikokrat se je med izvajanjem algoritma na danem testu izvedla vrstica, ki v kodi sledi ukazu //@COUNT{counter\_name, value}. Primer: če v algoritem za urejanje podatkov pred vsako vrstico, v kateri primerjamo dva podatka, zapišemo //@COUNT{compare, 1}, bo na koncu izvajanja algoritma števec compare vseboval število vseh primerjav, ki so se izvedle.

* JVM indikatorji

Število posameznih java bytecode ukazov, ki so se izvedli med izvajanjem algoritma.

### Zapis rezultatov

Algoritmi se običajno izvajajo nad celotno množico nalog (ob enem izvajanju se izvedejo vsi testi podane množice). Rezultati izvajanja se zapišejo v izhodno datoteko, v kateri vsakemu testu pripada ena vrstica s parametri in indikatorji.

Podatki v izhodni vrstici vsebujejo naslednje podatke (ločene s podpičjem):

1. privzeti izhodni parametri in indikatorji,
2. parametri testa,
3. lastnosti testa in
4. indikatorji izvajanja.

#### Privzeti izhodni parametri in indikatorji

Vsaka vrstica z rešitvijo vsebuje tri privzete izhodne parametre ter dva indikatorja:

* ime algoritma,
* ime testne množice,
* ime ali ID testa,
* indikator časovnega žiga (timestamp) in
* indikator uspešnosti testa.

Ime testa mora biti enolično za dano testno množico (dva testa znotraj iste testne množice ne smeta imeti istega imena). Če združimo več datotek z rezultati, prvi trije parametri enolično določajo test.

Indikator uspešnosti testa pove, ali se je izvajanje algoritma končalo v predvidenem času (parameter TimeLimit v atts datoteki). Indikator uspešnosti ima lahko tri vrednosti: PASS (izvajanje se je končalo brez napak), FAILED (med izvajanjem je prišlo do izjeme) ali KILLED (izvajanje je bilo ustavljeno po TimeLimit sekund).

#### Parametri testa

Vrstni red parametrov je določen v attc datoteki.

#### Lastnosti testa

Lastnosti testa (TC\_PROPS) so opcijski podatek, ki ga lahko ustvari generator testa. Lastnosti se zapišejo v izhodno vrstico na mesto za zadnjim parametrom in sicer kot seznam z vejico ločenih podatkov tipa lastnost=vrednost. Latnosti se lahko uporabijo v poizvedbah za bolj natančno analizo.

#### Indikatorji izvajanja

Indikatorji, ki se izpišejo v izhodno datoteko, in njihov vrstni red so določeni s pripadajočo atrd konfiguracijsko datoteko in sicer:

* specifične in časovne indikatorje – [Project]-em.atrd,
* za števce – [Project]-cnt.atrd,
* za JVM indikatorje – [Project]-jvm.atrd,

## Meritve

Izračun indikatorjev ALGator opravi z več ločenimi meritvami (primer: z eno meritvijo izmerimo časovne, z drugo pa jvm parametre). Ena meritev se izvede nad celotno množico nalog. Rezultati vsake meritve so zapisani v samostojni datoteki. Za par (algoritem, testna množica) imamo zato več izhodnih datotek, za vsako meritev po eno.

Vsebina izhodnih datotek (katere indikatorje vsebuje) je opisana v pripadajoči atrd datoteki (proj/[Project]-[ext].atrd). Ime izhodne datoteke je [Algoritem]-[TestSetName].[ext]. Pri tem je [ext] končnica, specifična za to meritev (em, cnt in jvm).

### Meritve časovnih in specifičnih indikatorjev izvajanja

Ime atrd datoteke: proj/Project-em.atrd

Končnica: izhodne datoteke: .em

Glavni namen te meritve je izračun specifičnih ter časovnih indikatorjev.

Časovni indikatorji se izmerijo avtomatsko (čas izvajanja metode run()). Vsak test se požene večkrat (parameter TestRepeat v atts datoteki). V izhodno datoteko se zapiše ena številska vrednost, ki jo sistem izračuna na podlagi izmerjenih časov po različnih formulah. Katera formula se bo uporabila, določi avtor atrd datoteke. Primer: če bosta v atrd datoteki definirana indikatorja

{

"Name" : "Tmin",

"Description" : "Sorting MIN time",

"Type" : "timer",

"Subtype" : "0 MIN"

}

{

"Name" : "Tavg",

"Description" : "Sorting AVG time",

"Type" : "timer",

"Subtype" : "0 AVG"

}

bosta v izhodni datoteki zapisana minimalni in povprečni čas izvajanja algoritma pri TestRepeat ponovitvah tega testa. Osnovni namen ponavljanja je izničenje naželjenih vplivov iz okolja. Pri natančnem ocenjevanju izvajanja algoritma običajno uporabimo minimalno vrednost (pri tem izvajanju je prišlo do najmanj motenj iz okolja).

Vrednost specifičnih indikatorjev se določi (izračuna) v metodi getIndicatorValue() v razredi [Project]Output. Metoda vrednost indikatorja določi na podlagi podatkov o vhodu, pričakovanem izhodu in dejanskem izhodu.

### Meritve s števci

Ime atrd datoteke: proj/Project-cnt.atrd

Končnica izhodne datoteke: .cnt

Koda algoritma lahko vsebuje ukaz //@COUNT{counter\_name, value} ki poveča vrednost števca counter\_name za value (primer: //@COUNT{SWAP, 1} poveča vrednost števca SWAP za 1). Po končanem izvajanju izvajalni sistem vrne vrednost vseh števcev, ki so navedeni v polju Indicators v datoteki proj/Project-cnt.atrd. (to polje določa tudi vrstni red zapisa števcev v izhodno datoteko). Polje Parameters v atrd datoteki je pri tej meritvi opcijsko.

Meritev s števci vsak algoritem nad posameznim testom požene samo enkrat. Parameter TestRepeat v tej datoteki se ignorira.

Pred izvajanjem meritve s števci se izvorna koda [alg\_name]Algorithm.java prepiše v datoteko [alg\_name]ALgorithm\_COUNT.java, pri tem pa se vsi ukazi //@COUNT{counter\_name, value} zamenjajo z odgovarjajočimi javanskimi ukazi za povečevanje števcev. Ob izvajanju cnt meritve, se nato izvaja razred [alg\_name]ALgorithm\_COUNT.class.

Pri prepisu datoteke se

1) vse pojavitve

//@COUNT{cnt\_name, value}

nadomestijo z

Counters.addToCounter("cnt\_name", value);

2) vse vrstice, ki vsebujejo //@REMOVE\_LINE, odstranijo.

Primer: Izvorna koda Koda v \_COUNT datoteki

|  |  |
| --- | --- |
| /\*//@REMOVE\_LINE  //@COUNT{CMP, 1}  if (a[e1] != a[e2]) {  //@COUNT{CMP, 1}  }  \*///@REMOVE\_LINE    **if** (a[e1] != a[e2] && a[e2] != a[e3]) {  //@COUNT{CMP, 1}  **while** (a[++less] < pivot1) {  //@COUNT{CMP, 1}  }  } | Counters.addToCounter("CMP", 1);  **if** (a[e1] != a[e2]) {  Counters.addToCounter("CMP", 1);  }  **if** (a[e1] != a[e2] && a[e2] != a[e3]) {  Counters.addToCounter("CMP", 1);  **while** (a[++less] < pivot1) {  Counters.addToCounter("CMP", 1);  }  } |

### Meritve s prirejenim JVM

Ime atrd datoteke: proj/Project-jvm.atrd

Končnica izhodne datoteke: .jvm

Pri tej meritvi štejemo število posameznih bytecode ukazov ter velikost porabljenega pomnilnika (ukazi new, newarray, ...). Za to meritev uporabljamo drugačno izvajalno okolje (prirejen JVM, ki šteje ukaze).

# Konfiguracija sistema

Vsi podatki, s katerimi upravlja sistem ALGator, so zapisani v mapi <algator\_root> in njenih podmapah v tekstovnih konfiguracijskih datotekah tipa JSON ali CSV. Tip posamezne datoteke je razviden iz opisa datoteke v tem poglavju.

Sistem ALGator uporabi vrednosti sistemske spremenljivke $ALGATOR\_ROOT in predpostavi, da kaže na <algator\_root>. Če ta spremenljivka ni nastavljena, ALGator v ta namen uporabi trenutni direktorij. Podatki o projektih se nahajajo v mapi $ALGATOR\_ROOT/data\_root. Obe vrednosti lahko uporabnik nastavi tudi s pomočjo stikal –r in –dr ob klici programov.

Vsebina datotek, ki predstavljajo konfiguracijo sistema je opisana v datoteki ALGator\_conf.docx.