

Tomaž Dobravec

Univerza v Ljubljani

Fakultata za računalništvo in informatiko

KAZALO

1 Opis sistema 3

1.1 Namen 3

2 Delovanje sistema 3

2.1 ALGator kot samostojna aplikacija 3

2.2 ALGator kot spletna aplikacija 4

2.2.1 Vloge pri uporabi spletne aplikacije ALGator 4

2.2.2 Primer uporabe spletne aplikacije ALGator 5

3 Izvajanje algoritmov 6

3.1 Definicija osnovnih pojmov 6

3.2 Naloge in množice nalog 7

3.3 Algoritmi 10

3.4 Rešitev naloge in zapis rezultatov 12

3.4.1 Privzeti izhodni parametri in indikatorji 13

3.5 Meritve 13

3.5.1 Meritve časovnih in specifičnih indikatorjev izvajanja 13

3.5.2 Meritve s števci 14

3.5.3 Meritve s prirejenim JVM 15

3.6 Izdelava projekta in algoritma 15

4 Konfiguracija sistema 17

4.1 The structure of the <algator\_root> folder 17

4.2 System configuration 18

4.2.1 Building blocks 18

4.2.2 Projects 20

4.2.3 Algorithms 22

4.2.4 Tests 24

4.2.5 Result 24

4.2.6 Queries 26

4.2.7 Presenters 32

4.2.8 Configuration of the ALGator system 34

# Opis sistema

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

Sistem omogoča dodajanje in upravljanje s poljubnim številom 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 lahko uporabljamo na dva načina: kot samostojno ali kot spletno aplikacijo. Samostojna aplikacija je namenjena razvijalcem, ki želijo ALGator uporabljati na svojem računalniku, ločeno od ostalih uporabnikov. Spletna aplikacija omogoča uporabo sistema več uporabnikom hkrati. Rezultati izvajanja spletne aplikacije so vidni vsem uporabnikom.

## ALGator kot samostojna aplikacija

ALGator je zbirka javanskih razredov, ki so združeni v knjižnici ALGator.jar. Za pravilno delovanje sistema, je potrebno:

1. pridobiti program ALGator (ALGator.jar + folder lib)
2. ustvariti mapo data\_root in ustvariti vsaj en projekt
3. Pravilno nastaviti sistemske spremenljivke
   * + CLASSPATH (dodati je treba pot do ALGator.jar)
     + ALGATOR\_ROOT (kazati mora na folder, kjer je ALGator.jar)
     + ALGATOR\_DATA\_ROOT (kazati mora na data\_root folder)
     + ALGATOR\_DATA\_LOCAL (kazati mora na nek lokalni folder, ki ne sme biti podfolder ALGATOR\_DATA\_ROOT folderja)

Zadnja verzija programa ALGator se nahaja na https://github.com/ALGatorDevel/Algator. Tam je v datoteki README opisan postopek namestitve.

ALGator izvajamo z enim od naslednjih ukazov:

**java algator.Version**

izpis sporočila o verziji ALGatorja in privzeti nastavitvi

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

**java algator.New**

ustvarjanje konfiguracijskih datotek novega projekta/algoritma

**java algator.Execute**

izvajanje definiranih algoritmov izbranega problema

**java algator.Analyse**

analiza rezultatov izvajanja

Uporabnik ALGatorja kot samostojne aplikacije ima popoln nadzor nad celotno konfiguracijo sistema, saj hkrati nastopa v vlogi skrbnika sistema, skrbnika projekta in raziskovalca (za pomen posamezne vloge glej opis spodaj).

## ALGator kot spletna aplikacija

Za pravilno delovanje ALGator spletne aplikacije potrebujemo tri podsisteme: **AEE** (ALGator execution engine), **ADE** (ALGator data engine) in **AWE** (ALGator Web engine). Na posameznem fizičnem računalniku lahko teče več podsistemov (na primer: en fizični računalnik je lahko AEE ADE in AWE hkrati). Spletna aplikacija vsebuje natanko en ADE in en AWE ter enega ali več AEE.

### Vloge pri uporabi spletne aplikacije ALGator

Spletni sistem ALGator lahko uporabljajo uporabniki z različnimi vlogami in pravicami. Neprijavljen uporabnik (gost) lahko izvaja le funkcije, ki ne spreminjajo podatkov sistema (podatke lahko le pregleduje). Za izvajanje ostalih funkcij mora biti uporabnik v sistem prijavljen. Natančni pomen posamezne vloge je opisan v nadaljevanju.

#### Skrbnik sistema

Skrbnik sistema postavi celotni sistem in skrbi za strojno in programsko opremo in ima vse pravice in dostop do vseh virov sistema.

#### Skrbnik projekta

Skrbnik projekta definira en projekt. K definiciji projekta sodi:

* definicija problema,
* definicija testnih množic vhodnih podatkov,
* opis formata vhodnih in izhodnih podatkov,
* definicija vseh javanskih razredov, ki so potrebni za izvajanje algoritmov tega projekta.

Skrbnik projekta ima dostop do vseh virov projekta. Če je projekt javen (ta atribut nastavi skrbnik projekta), lahko podatke o projektu vidijo vsi uporabniki. Privatne projeket vidi samo skrbnik sistema in skrbnik projekta.

#### Raziskovalec

Raziskovalec definira en algoritem znotraj izbranega projekta. Javne algoritme (ta atribut nastavi raziskovalec) lahko vidijo vsi uporabniki, privatnega pa poleg skrbnikov le raziskovalec.

#### Gost

Gost lahko vidi podatke o javnih algoritmih javnih projektov.

Za podrobnejše informacije o delovanju spletne aplikacije glej ALGator-dev.docx

### Primer uporabe spletne aplikacije ALGator

V tem razdelku je opisan najpogostejši način uporabe spletne aplikacije ALGator.

* Skrbnik sistema postavi celotni sistem in objavi povezavo do spletne strani sistema.
* Skrbnik projekta doda nov projekt; s tem definira problem in nabore testnih primerov ter predpiše način reševanja tega problema. Za vsak definiran projekt sistem avtomatsko ustvari spletno stran z razlago problema in navodili za pripravo algoritma.
* Skrbnik projekta v projekt doda nekaj znanih algoritmov za reševanje tega problema, ki bodo služili za osnovno primerjavo z novimi algoritmi, ki jih bodo dodajali raziskovalci.
* Raziskovalec skladno z navodili, objavljenimi na spletni strani projekta, pripravi nov algoritem in ga doda v sistem. ALGator dodani algoritem avtomatsko požene na testnih podatkih. Raziskovalec nato primerja učinkovitost tega algoritma z učinkovitostjo algoritmov, ki so že v projektu. Če želi, lahko algoritem odpre za javnost (algoritem postane javen) ali pa ga skrije (privaten algoritem). Privatne algoritme lahko raziskovalec kadarkoli spremeni v javne.
* Gost sistema lahko pregleduje podatke o vseh javnih projektih in javnih algoritmih. Lahko izvaja primerjave in ostale operacije, ki ne spreminjajo konfiguracije sistema. Če želi, se lahko kadarkoli prijavi kot raziskovalec ali kot skrbnik projekta.

# 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.

* **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 (test case)**

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

* **REŠITEV NALOGE**

Skupek parametrov in indikatorjev, ki opisujejo obnašanje algoritma pri reševanju dane naloge (velikost problema, poraba časa, pravilnost rezulata, 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 ustrezno atrd datoteko.

## Naloge in množice nalog

* Izvrševalec z danim algoritmom reši vse naloge dane množice nalog.
* Za opis ene naloge v javi uporabljamo razred TestCase, ki vsebuje množico parametrov (generično ime problema, velikost problema, ...). Ker se opisi nalog posameznih problemov med seboj močno razlikujejo, mora skrbnik projekta napisati razred

[Project]TestCase extends TestCase

v katerem predvidi podatkovne strukture za shranjevanje podatkov o nalogi.

Primer: pri projektu Sort (urejanje podatkov) za shranjevanje podatkov v razredu SortTestCase definiram tabelo

public int [] arrayToSort;

* Ker je način zapisa naloge močno odvisen od problema (primer: naloga faktorizacije je podana z eno celoštevilsko vrednostjo, naloga urejanja s tabelo števil, naloga ZIP pa je podana z datoteko), je opis nalog prepuščen skrbniku projekta.

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 tipa TestSet (glej poglavje 3.2.4.1), v nje pa 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 te datoteke povsem odvisen od projekta. Format določi skrbnik projekta.

Skrbnik projekta mora predvideti, kako bo na podlagi podatkov, zapisanih v datotekah [TestSet].atts in [TestSet].txt pridobil podatke o vseh nalogah te testne množice.

* Za »sprehod« po dani množici nalog se uporablja razred AbstractTestSetIterator. Naloga tega razreda je, da prebere vse vrstice datoteke [TestSet].txt, jih primerno interpretira in na podlagi podatkov, ki so zapisani v posamezni vrstici ustvari primerek razreda [Project]TestCase. V ta namen razred AbstractTestSetIterator predvideva naslednje metode:
  + void initIterator()

Ko se pokliče ta metoda, je v atribut testSet že nastavljen. Naloga metode initIterator() je, da pripravi vse potrebno, da bo klic ostalih metod potekal nemoteno. Običajno ta metoda odpre konfiguracijsko datoteko (opisano v prejšnji alineji) in prebere (ali pa se pripravi za branje) podatkov o nalogah.

* + void hasNext();

Vrne true, če je v tej testni množici še kakšna naloga in false sicer.

* + void readNext();

Prebere in v internih strukturah shrani naslednjo nalogo.

* + TestCase getCurrent();

Vrne nalogo, ki je bila prebrana z zadnjim readNext().

Razred AbstractTestSetIterator je abstrakten (vse zgoraj naštete naloge so abstraktne). Naloga skrbnika projekta je, da napiše naslednika tega razred: [Project]TestSetIterator. V večini primerov so naloge, za katere poskrbijo prve tri omenjene metode, enake, zato je ALGatorju dodan razred DefaultTestSetIterator, v katerem so definirane vse metode AbstractTestSetIteratorja razen metode getCurrent(). Razen v izjemnih primerih (pri zelo nespecifičnih projektih) je torej dovolj, da skrbnik impementira naslednika razreda DefaultTestSetIterator in v njem napiše zgolj metodo getCurrent(), ki iz vsebine ene vrstice datoteke [TestSet].txt, ustvari [Project]TestCase. Vsebino vrstice datoteke metoda getCurrent() dobi s spremenljivki currentInputLine.

Primer izvedbe metode getCurrent() v razredu SortTestSetIterator je prikazan na sliki Slika 1.



Slika 1: Primer zapisa nalog in implementacije TestSetIterator-ja za problem Sort.

## Algoritmi

* Za izvajanje algoritmov ALGator uporablja razred AbsAlgorithm, ki vsebuje naslednje metode:
  + ErrorStatus init(TestCase test)

Metoda poskrbi, da se iz podatkov, ki so zapisani v objektu test, ustvarijo podatki, na katerih se bo izvedel test. Pri tem se morajo opraviti vsi časovno zahtevni postopki v zvezi z obdelavo vhodnih podatkov. Po koncu te metode mora biti vse pripravljeno za dejansko izvajanje algoritma.

* + void run()

V tej metodi se kliče metoda execute(<parameters>). Parametri metode execute() so odvisni od problema in jih v času ustvarjanja projekta predvidi skrbnik. Čas izvajanja metode run() se šteje kot čas izvajanja algoritma, zato morajo biti parametri, ki se podajo v metodi execute(), že prej skrbno pripravljeni (v metodi init()). Priprava oziroma obdelava parametrov znotraj metode run() po krivici veča časovno zahtevnost algoritma!

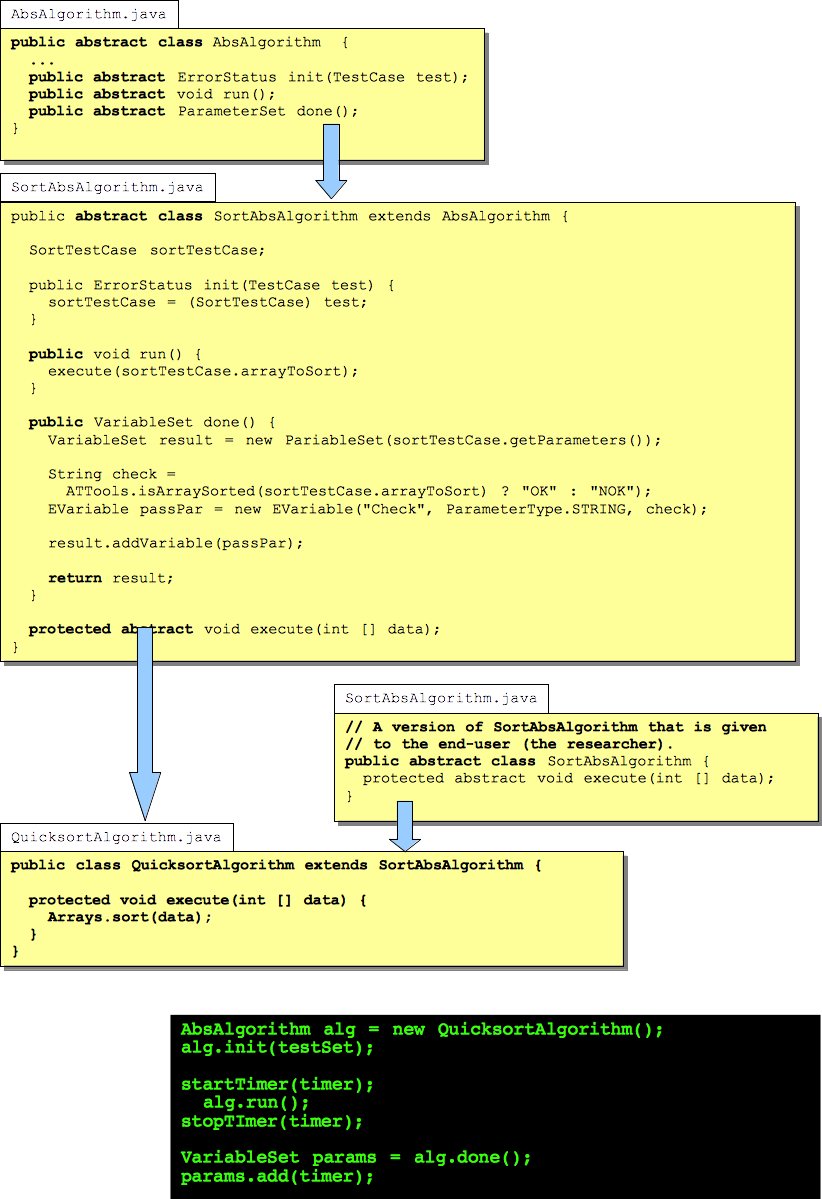
* + VariableSet done();

Metoda zbere vse parametre (vhodni parametri, parametri izvajanja in časovni parametri) in jih vrne v VariableSet obliki.

Razred AbsAlgorithm je abstrakten. Naloga skrbnika projekta je, da napiše naslednika tega razred -[Project]AbsAlgorithm - in v njem napiše vse tri zgoraj omenjene metode. Poleg tega mora deklarirati atribute za shranjevanje podatkov o nalogi (v te atribute bo metoda init() prepisala podatke o nalogi iz podanega TestCase-a) ter abstraktno metodo execute(...) s problemu prirejenim naborom parametrov.

* Naloga raziskovalca je, da implementira naslednika razreda [Project]AbsAlgorithm. Da bi preprečili morebitno zmedo, se raziskovalcu posreduje poenostavljeno verzijo tega razreda – razred, ki je direkten naslednik razreda Object in vsebuje le abstraktno metodo execute(...). Razred, ki je naslednik tega poenostavljenega razreda, bo hkrati (brez sprememb kode) tudi naslednik pravega razreda [Project]AbsAlgorithm. Na ta način raziskovalcu ni treba vedeti za zapleteno strukturo pravega očeta in se lahko osredotoči samo na implementacijo algoritma, hkrati pa bo njegov razred brez sprememb uporaben v širšem kontekstu celotnega sistema.

Primer implementacija razreda AbsAlgorithm in njegovih naslednikov za konkreten problem je prikazan na sliki Slika 2.



Slika 2. Razred AbsAlgorithm in njegovi nasledniki

## 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]AbsAlgorithm.done().

* Časovni indikatorji

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

* Števci

Števci povejo, koilkokrat 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.

Algoritmi se 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. Kateri parametri se nahajajo v tej vrstici in kakšen je njihov vrstni red, je določeno s pripadajočo atrd konfiguracijsko datoteko:

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

### Privzeti izhodni parametri in indikatorji

Vsaka vrstica z rešitvijo vsebuje tri privzete izhodne parametre ter en indikator:

* ime algoritma,
* ime testne množice,
* ime testa,
* 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).

## 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 parametre in 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 v metodi [Project]AbsAlgorithm.done() na podlagi podatkov o izvajanju algoritma.

### 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).

## Izdelava projekta in algoritma

Skrbnik projekta ustvari nov projekt in testni algoritem z ukazom

java algator.Admin –ca <ime\_projekta> <ime\_algoritma>

Pri tem vsi podatki za nov projekt nastanejo v direktoriju $ALGATOR\_DATA\_ROOT/projects/PROJ-<ime\_projekta> in njegovih poddirektorijih. Privzet direktorij lahko skrbnik spremeni s stikalom –dr.

Ko skrbnik z zgoraj navedenim ukazom ustvari projekt in testni algoritem, mora popraviti še naslednje:

**1) Testne množice**

* V datoteki TestSet1.atts mora vpisati pravilne vrednosti za parametre N, TimeLimit, TestRepeat.
* V datoteki TestSet1.txt mora napisati N vrstic (N testnih primerov)

**2) Izvorna koda projekta**

* V datoteki [Project]TestCase.java definira podatkovno strukturo
* V datoteki [Project]TestSetIterator.java v metodi getCurrent()
  + na podlagi podatkov iz currentInputLine ustvari vhodne podatke za izvajanje algoritma na tem testnem primeru in jih zapiše v tCase
  + med parametre testnega primera doda vse vhodne parametre, ki so zapisani v currentInputLine
* V datoteki [Project]AbsAlgorithm.java
  + v metodi init() vhodne podatke, ki so zapisani v tCase, prepiše v interne atribute in opravi vse potrebne predpriprave teh podatkov,
  + definira podpis metode execute(),
  + v metodi run() kliče metodo execute() s prej pripravljenimi podatki
  + v metodi done() na podlagi rezultatov, ki jih generira metoda execute(), izračuna vrednost indikatorjev in jih zapiše med spremenljivke (variables).

**3) Izvorna koda algorima**

* V datoteki [Algorithm]Algorithm.java napiše definira metodo execute() in jo tudi napiše (izvajanje algoritma).

**4) Ostali podatki**

Vse zgoraj opisane naloge (1, 2, in 3) mora skrbnik opraviti za pravilno »delovanje« projekta (prevajanje kode in izvajanje algoritmov). Za administrativne potrebe (vzdrževanje sistema, prezentacija rezultatov, analiza, ...) pa mora skrbnik vpisati tudi ostale podatke v konfiguracijskih datotekah (Name, Description, ShortName, ...) in vsebino HTML datotek (te datoteke se uporabljajo za prikaz rezultatov na spletu).

# 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 na $ALGATOR\_DATA\_ROOT. Če vrednost te sistemske spremenljivke ni nastavljena, se uporabi $ALGATOR\_ROOT/data\_root. Obe vrednosti lahko uporabnik nastavi tudi s pomočjo stikal –r in –dr.

## The structure of the <algator\_root> folder

<algator\_root>

data\_local // local data for this computer

local.acfg // configuration of this computer

projects // local copy of projects

data\_root // folder with information about projects

global.acfg // configuration of the system

projects // root for the projects

log // logging folder

tasks // communication folder

schema // folder for schema

app // folder with programs and services

ALGator // folder with ALGator.jar, ...

* thelocal.acfg file contains the configuration of the current computer (i.e. computer ID, path for VMEP virtual machine, ....). See section 4.2.8.5 for datails.
* the <data\_root> folder contains informatin about the ALGator projects. If a computer is a part of a group of computers composing ALGator system, this folder should be set as a link to this system's data\_root folder (that is mounted, for example, using the samba protocol).
* in the <global\_config> folder containes inforamtion about the ALGator system. See section 2.2.9 for datails.
* In the <tasks> folder serves as a comunication chanel between AED and AEEs of the system.

The <projects> folder contains the configuration information for all the projects that are defined in the system. For every project [Project] it contains a folder named PROJ-[Project] with the following structure:

PROJ-[Project] // project root folder

proj // folder for all project files

[Project].atp // project configuration file

[Project]-em.atrd // result description file for em

src // folder for project source files

[Project]AbsAlgorithm.java

[Project]TestSetIterator.java

[Project]TestCase.java

doc // html documentation for project

algs // folder for algorithms

ALG-[Algorithm] // algorithm root folder (\*1)

[Algorithm].atal // algorithm configuration file

src // algorithm source files

[Algorithm]Algorithm.java

doc // html documentation for algorithm

reports // folder for algorithms reports

tests // project-specific test files folder (\*2)

TestSetName1.atts // one or more testset files

TestSetName1.txt // one or more testset files

results // folder with results

queries // folder for predefined queries

reports // folder for projects reports

\*1 - for each algotihm AlgName there is one folder with this structure

\*2 - files in this folder can be organized in sub-folders. The structure of this

folder is specific to the project and it is not defined on the ATSystem

level. Only the curent TestSetIterator knows for the structure of

this folder.

## System configuration

All the entities of the ALGator are defined by configuration files that are placed within the <algator\_root> folder. The type, location and the structure of these files is described in the following.

### Building blocks

#### NameAndAbrev

A String that identifies the name of an entity and its abreviation. A reserved word AS is used as delimiter: Example:

"BubbleSort AS BS".

If the second part of NameAndAbrev is missing (i.e. "BubbleSort"), the value is interpreted as "Name AS Name".

#### StatFunc

A statistical function used to calculate one scalar from a list of values. The functions are one of the following:

MIN the minimal element in the list

MAX the maximal element in the list

SUM the sum of all elements in the list

AVG the average of the elements in the list

VAR the variance of the elements in the list

FIRST the first element in the list

LAST the last element in the list

#### Sorting functions

To sort data, one of the following methods can be used:

* + numerical sort, ascending
* - numerical sort, descending
* > lexicogvvdraphic sort, ascending
* < lexicographic sort, descending

#### VariableType

The variables are used in the program for several purposes (i.e. to hold a value of a test parameter or result indicator, ... ) are of the following types:

* timer a timer for one stage of algorithm execution (int value)
* counter counter of operations (integer)
* int integer value
* double double value; for decimal separator “.” is used;
* string a string inside quotes “”

Each type can have a subtype. A meaning of subtype is type-dependant. For timers a subtype defines a StatFunc to be used

|  |  |  |
| --- | --- | --- |
| Type | Subtype | Description |
| timer | # StatFunc | # ... the id of a timer (0 is the default timer id used to measure overall time; other timers may be activated by project administrator by calling timer.next() method).    StatFunc ... a statistical function used to calculate the value of this timer.  Default: 0 MIN |
| double | # | # ... number of decimal places to be used |

#### Variable

A description for one variable.

**File type**: JSON

**EntityID:** Variable

**Fields:**

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| Name | String | Name of the variable |
| Description | String | Description of the variable |
| Type | VariableType | A type of a variable |
| Subtype | String | Optional. A space-delimited list of VariableType specific arguments. This arguments are used by Executor to distinguish, for example, different timers. |
| Value | String | Optional. A value of a variable (if known). |

**Example:**

{

"Name" : "T1",

"Description" : "Preprocessing timer",

"Type" : "timer",

"Subtype" : "1 AVG"

}

### Projects

Each project ProjName is defined in the folder named PROJ-ProjName within the projects folder.

Shortcut: <project\_root> = <data\_root>/projects/PROJ-ProjName

#### Project

The configuration of an individual project ProjName is defined in atp file as follows.

**File type**: JSON

**EntityID:** Project

**File extension**: atp

**File location**: <project\_root>/proj/ProjName.atp

**Fields:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | | Type | | Description |
| Name | | String | | [Project] |
| Description | | String | | HTML text to be shown on project website |
| Author | | String | | Author of the project |
| Date | | String | | The date of the last change |
| TestSets | | String[] | | a list of TestSets |
| Algorithms | | String[] | | a list of Algorithms |
| ProjectJARs | Filename[] | | An array of jars that are available when compiling the project; names are relative to the folder <project\_root>/proj/lib | |
| AlgorithmJARs | Filename[] | | An array of jars that are available when compiling an algorithm; names are relative to the folder <project\_root>/proj/lib. | |
| EMExecFamily | String | | Identification of the family of computers that can execute em measurements for algorithms of this problem. If not defined, any computer with EM capability can execute algorithms. When first algorithm is executed by a given computer, this value has to be set so that other algorithms of this project will be run on the same family. | |
| CNTExecFamily | String | | Identification of the family of computers that can execute cnt measurements for algorithms of this problem. If not defined, any computer with CNT capability can execute algorithms. | |
| JVMExecFamily | String | | Identification of the family of computers that can execute jvm measurements for algorithms of this problem. If not defined, any computer with JVM capability can execute algorithms. | |
| ProjPresenters | String [] | | An array of presenters to be displayed on the project results description page. | |
| MainProjPresenters | String [] | | An array of main presenters to be displayed on the project's first page. | |
| AlgPresenters | String [] | | An array of presenters to be displayed on the algorithms results description page. | |
| MainAlgPresenters | String [] | | An array of mainpresenters to be displayed on the algorithm's first page. | |

Note1: project description implicitly includes three source files for each language (3 for java, 3 for C, ...) named <project\_name>AbsAlgorithm.<ext>, <project\_name>TestCase.<ext> and <project\_name>TestSetIterator.<ext> (where <ext> = java or cpp). These files should be placed in the src subfolder of a project.

Note2: AlgorithJARs are provided by project administrator. An author of the algorithm CAN NOT attach a JAR file to his algorithm (all sources must be evident to the public) .

Example:

{

"Project" : {

"Description" : "Testing several sorting methods",

"Author" : "Tomaz",

"Date" : "30/07/2013",

"Algorithms" : ["QuickSort", "BubbleSort"],

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

}

}

### Algorithms

All the data of the algorithm [Algorithm] is placed in the folder algs/ALG-[Algorithm] inside the <project\_root> folder.

Shortcut: <algorithm\_root> = <project\_root>/algs/ALG-[Algorithm]

The <algorithm\_root> folder contains:

* [Algorithm].atal file with the description of the algorithm
* src folder with all source files. The name of main algorithm's class should be [Algorithm]Algorithm.<ext> (where <ext> = java or cpp). Note: if algorithm requires more than one source file, names of these files have to be listed in Classes property (names are relative to algorithms's src folder). The main algorithm class MUST be in <algorithm\_root>/src folder (not in a subfolder!).

#### Algorithm

The configuration of an individual algorithm [Algorithm] is defined in atal file as follows.

**File type**: JSON

**EntityID:** Algorithm

**File extension**: atal

**File location**: <algorithm\_root>/AlgName.atal

**Fields:**

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| Name | String | [Algorithm] |
| ShortName | String | a project unique short name (at most 8 chars) |
| Description | String | HTML text to be shown on algorithm website |
| Author | String | Author of the algorithm |
| Date | String | Creating date of the algorithm |
| Classes | Filename[] | an array of source files (other than <algorithm\_name>Algorithm.java |
| Language | String | Valid values: JAVA (default) or C |
| ExecuteSignature | String | Signiture for execute function of the algorithm. This property is used only for C++ algorithms. Example:    int \*tab, int size |
| MainPresenters | String[] | Main presenters for algorithms (to be displayed on first page) |

Example: QuickSort1.atal

{

"Algorithm" : {

"ShortName" : "BS",

"Description" : "The classical bubble sort algorithm",

"HTMLDescFile" : "bubblesort.hmtl",

"Author" : "BS and Tomaz",

"Date" : "07/30/2013"

}

}

### Tests

#### TestSet

A description of a collection of tests.

**File type**: JSON

**EntityID:** TestSet

**File extension**: atts

**File location**: <tests\_root> = <project\_root>/tests

**Fields:**

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Type | | Description |
| Name | String | | [TestSet] |
| ShortName | String | | A shortname of this test set. This description will be used in the result presentation forms. |
| Description | String | | A detailed description of this test set |
| N | Integer | | The number of tests in this test set. When performing this test set, Executor must produce N outputs. This property is used to check if a tesset has already been completed (i.e. if the output file contains N results). |
| TestRepeat | | Integer | Number of times to execute a single test from this test set. |
| TimeLimit | | Integer | Maximum time allowed (in seconds) for execution of one **test**; if the algorithm does not finish in this time, the execution is killed. Default value: 10. |

**Example:**

"TestSet": {

"Name" : "TestSet1",

"ShortName" : "TS1",

"Description" : "This testset contains set of simple tests",

"N" : "13",

"TestRepeat" : "10",

"TimeLimit" : "10"

}

### Result

A description of a format of a result, containing a description of all parameters and indicators that represent in a single result.

**File type**: JSON

**EntityID:** Result

**File extension**: atr

**File location**: <project\_root>/proj

**Fields:**

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| ParameterOrder | String [] | The order of test parameters. |
| Parameters | Variable[] | An array of variables representing parameters. |
| IndicatorOrder | String [] | The order of indicators. |
| Indicators | Variable[] | An array of variables representing indicators. |

The parameters ParameterOrder and IndicatorOrder define the order of parameters and indicators listed in a result file. The property Parameters describes the parameters of the test (i.e. the number of elements, group name, ...) while the property Indicators describe the indicators of the result (algorithm execution on a given test).

A default variable type is INT. If a parameter is listed in ParameterOrder or a indicator is listed in IndicatorOrder and is not defined in Parameters or Indicators, a default INT variable is used.

**Example:**

{

"Result":

{

"ParameterOrder" : ["Test", "Group", "N"],

"Parameters": [

{

"Name": "Test",

"Description": "The name of the test",

"Type": "string"

},

{

"Name": "Group",

"Description": "The name of the group",

"Type": "string"

},

{

"Name": "N",

"Description": "The size of the test",

"Type": "int"

}

],

"IndicatorOrder" : ["Tmax", "Check"],

"Indicators": [

{

"Name" : "Tmax",

"Description" : "Sorting MAX time",

"Type" : "timer",

"Subtype" : "0 MAX"

},

{

"Name" : "Check",

"Description" : "The correctness of the algorithm (OK/NOK)",

"Type" : "string"

}

]

}

}

Note: A result contains the following data

* algorithm … algorithm name
* testset … testset name
* test … unique id of test withi this testset
* status ... DONE if processes finished normally; KILLED if the test

exceeded the given amount of time; FAILED if other error

occured during the execution.

* the fifth (and the following columns) contain:

- if status == DONE,

- parameters and indicators are listed in the orders given in

ParamaterOrder and IndicatorOrder

- else

- ErrorMessage

If a value of a variable representing a given parameter or indicator is not known, it is presented by an empty string.

If an error occurs during the execution of a test, result for this test contains the first 4 data (algorithm, testset, test and status), followed by the error description string. All such lines in result file will be omitted during the analyses of the results, but they must be there (for error tracking and because the result file must contain exactly N lines)!

Example of one line in CSV result file:

quicksort;testset1;Test-7;DONE;sorted;10;35;5;7;23.5

### Queries

A query to combine results from several result files into single array of data.

**File type**: JSON

**EntityID:** Query

**File extension**: atqd

**File location**: <project\_root>/queries/QueryName.atqd

**Fields:**

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| Description | String | Description of a query (HTML text) |
| Algorithms | NameAndAbrev[] | An array of algorithms to be included in this query. The value "\*" means all algorithms. |
| TestSets | NameAndAbrev[] | An array of testsets to be included in this query. The value "\*" means all TestSets. |
| Parameters | NameAndAbrev[] | An array of test parameters . The value "\*" means all TestParameters. |
| Indicators | NameAndAbrev[] | An array of result indicators. The value "\*EM" means all EM indicators, "\*CNT" all CNT indicators and "\*JVM" all JVM indicators. |
| GroupBy | String[] | An array of groupby description |
| Filter | String[] | An array of filters |
| SortBy | String[] | An array of sort parameters or indicators |
| Count | String | This query is COUNT query?  »1« - true, other - false |
| ComputerID | String | ID of computer that will provide the results file; if null or "", the most suitable result file is chosen |

Query example:

{

"Description" : "Demo query",

"Algorithms" : ["QuickSort AS QS", "BubbleSort AS BS"],

"TestSets" : ["TestSet1 AS TS1", "TestSet2 AS TS2"],

"Parameters" : ["TestSet", "Test AS T", "N"],

"Indicators" : ["TMin", "TMax"],

"GroupBy" : ["N; Tmax:SUM; MIN"],

"Filter" : ["N > 100", "N < 1000"],

"SortBy" : ["N:-"],

"Count" : ["1"]

}

Dodatna pojasnila:

* Polja Algorithms, Testsets, Parameters, Indicators določajo obseg poizvedbe (kateri algoritmi in testne množice dobo vključeni v rezultat) in širino tabele (kateri stolpci bodo nastopali v rezultatu).

Podatki o izhodnih indikatorjih (Indicators) se črpajo iz vseh atrd datotek (glej poglavje 3.5). Podatki o parameterih testa se črpajo le iz Project-em.atrd.

Izhodna tabela vsebuje tri privzete stolpce (se jih ne da izključiti): ID, TestSet, TestID in Pass.

Če imamo v poizvedbi N algoritmov in M testnih množic ter T testnih parametrov in R indikatorjev rezultata, potem bo tabela vsebovala 4 + T + N \* R stolpcev.

* Polji GroupBy in Filter vplivata na število vrstic (lahko jih zmanjšata)

- Operacije se izvajajo v tem vrstnem redu: generate table, filter, group, sort.

* Polje Filter:
  + Filter iz tabele izloči vse vrstice, ki NE zadoščajo pogojem.
  + Filtri se lahko izvaja po vseh parametrih in indikatorjih. Primer: če je CompRate indikator, se ob filtru "CompRate < 100" v tabeli ostanejo samo vrstice, pri katerih je vrednost CompRate < 100 pri VSEH algoritmih, ki so vključeni v poizvedbo.
  + Filter je tabela preprostih pogojev, za združevanje se uporabi operator and; z drugimi besedami, po filtru v tabeli ostanejo samo tiste vrstice, ki ustrezajo vsem pogojem filtra
    - Primer: Filter : ["N > 100", "N < 1000"]: med rezultati ostanejo samo tiste vrstice, v katerih je vrednost parametra N večja od 100 in manjša od 1000.
  + Posamezen filter je oblike: <ime\_parametra> <operator> <vrednost>
  + Operatorji filtra:
    - Za številske parametre: <, <=, >, >=, ==, !=
    - Za znakovne paramtere: ==, !=
  + Filter lahko vsebuje en parameter ali indikator. Primer:

Tavg < $1 @(10,100,10)

Tak filter se bo pognal večkrat (v zgornjem primeru za vrednosti $1=10,20, ..., 100). Ker je rezultat večkratnega poganjanja filtra več tabel, je tak filter smiseln samo v kombinaciji s parametrom COUNT==1 (glej spodaj). Izhodna tabela za vsako vrednost parametra $1 vsebuje eno vrstico, ki pripadajo poizvedbam

COUNT Tavg < 10

COUNT Tavg < 20

...

COUNT Tavg < 100

Obseg (range) parametra $1 je podan s tremi celoštevilskimi parametri:

@(start,stop,step)

Obseg pišemo na koncu kateregakoli pogoja, ki sestavlja filter.

* Polje GroupBy:
  + Vse vrstice z enako vrednostjo podanega parametra se združijo v eno vrstico.

* + GroupBy se lahko izvaja samo po parametrih testa,
  + Če GroupBy vsebuje več opisov, se združevanja naredijo eno za drugim (najprej prvo združevanje; drugo združevanje se naredi na rezultatu prvega, ...)
  + Pred posameznim združevanjem se tabela uredi po polju združevanja; način urejanja je lahko podan ob imenu polja za dvopičjem (primer: "GroupBy" = ["N:>; Tmax:SUM; MIN"]) – glej 2.2.1.3. Če način urejanja ni podan, se uporabi numerično urejanje v naraščajočem vrstnem redu (+).

* + Format zapisa:
    - Primer: "N; Tmax:sum; min"
    - Eden ali več podatkov, ločenih s podpičjem
    - Prvi podatek je ime parametra (in opcijsko način urejanja), po katerem se združuje
    - Ostali podatki opisiujejo, katera statistična funkcija naj se uporabi za izračun vrednosti posameznega stolpca.
      * Podatek tipa polje:StatfFunc podaja statistično funkcijo za to polje
      * Podatek tipa StatfFunc podaja statistično funkcijo za vsa polja, za katera ni eksplicitno določeno drugače
      * Če za katero polje ni podana statistična funkcija, se uporabi funkcija FIRST.

Primeri: Recimo, da ima tabela stoplce N, Tmin in Tmax.

* + - * GroupBy = "N" ... za vse stolpce se uporabi AVG
      * GroupBy = "N; MIN" ... za vse stolpce se uporabi MIN
      * GroupBy = "N; Tmax:MAX; MIN" ... za Tmax se uporabi MAX, za osttale MIN
      * GroupBy = "N; Tmax:MAX" ... za Tmax se uporabi MAX, za osttale AVG
* Polje SortBy:
  + tabela se uredi po stolpch, ki so navedane v tem polju
  + način urejanja je naveden z uporabo dvopičja, ki mu sledi podatek o načinu urejanja (glej 2.2.1.3.). Primer: "N:-" uporaba numeričnega urejanja, podajoče.
  + privzeto urejanje: numerično, naraščajoče (+).
* Polje Count
  + Ob izbrani opciji Count=1, poizvedba vrne tabelo z eno vrstico in N stolpci, za vsak izbran algoritem po en stolpec. Vrednost v tem stolpcu število vrstic, ki jih za pripadajoči algoritem filter NE izloči.
  + Primer: če je filter Tavg < 30, dobim za vsak izbran algoritem podatek o tem, koliko meritev se je končalo prej kot v 30ms.
  + Če se filter uporablja v kombinaciji s parametrom v filtru, potem ima rezultat toliko vrstic, kolikor različnih vrednosti zavzame filter.

#### Primer uporabe filtra in polja count:

Imamo dva algoritma, A in B, ter izhodni indikator Tavg.

|  |  |  |
| --- | --- | --- |
| Test | A.Tavg | B.Tavg |
| test1 | 12 | 9 |
| test2 | 14 | 14 |
| test3 | 15 | 16 |
| test4 | 14 | 16 |
| test5 | 23 | 17 |

a) Filter: Tavg < 15 COUNT=0

|  |  |  |
| --- | --- | --- |
| Test | A.Tavg | B.Tavg |
| test1 | 12 | 9 |
| test2 | 14 | 14 |

(razlaga: samo v 1. in 2. Vrstici je Tavg < 15 pri obeh algoritmih)

b) Filter: Tavg < 15 COUNT=1

|  |  |  |
| --- | --- | --- |
| # | A.COUNT | B.COUNT |
| 1 | 3 | 2 |

(razlaga: pri A imajo trije testi (test1, test2 in test4) Tavg < 15,

pri B pa le dva (test1 in test2)).

b) Filter: Tavg < $1 @(10,20,5) COUNT=1

|  |  |  |
| --- | --- | --- |
| # | A.COUNT | B.COUNT |
| 10 | 0 | 1 |
| 15 | 3 | 2 |
| 20 | 4 | 5 |

(opomba: 2. vrstica je enaka edini vrstici v primeru b)

### Presenters

#### GraphType

A GraphType specifies a type of a graph to be ploted. ALGator supports the following graph types:

* line
* stair
* bar
* box
* pie
* area
* donut

#### Presenter

Presenters are presentations of results used to present data produced by a query. Results can be presented on two different ways: as a table containing rows and columns as generated by query and/or as a graph.

**File type**: JSON

**EntityID:** Presenter

**File extension**: atpd

**File location**: <project\_root>/presenter/PresaenterName.atpd

**Fields:**

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| Name | String | Name of this presenter entity |
| Title | String | Title of this presenter (to be displayed on a web page) |
| ShortTitle | String | Short title to be displayed in limited space (i.e. html menu) |
| Description | String | Description of this presenter |
| Query | String | Filename of the query (or query itself if string starts with { ) that produces data for this presenter |
| HasGraph | Boolean | This presenter contains graph? |
| Xaxis | String | The name of the column that contains x axis data (available only if hasGraph==true) |
| Yaxes | String[] | The names of the columns that contains y axis data (available only if hasGraph==true) |
| GraphTypes | GraphType[] | Types of a graph to be shown. (available only if hasGraph==true) |
| XaxisLabel | String | X axis label. (available only if hasGraph==true) |
| YaxisLabel | String | Y axis label. (available only if hasGraph==true) |
| HasTable | Boolean | This presenter contains table? |
| Columns | String[] | Columns to be displayed in a table. (available only if HasTable==true) |

Example 1:

{

"Name" : "presenter1",

"Title" : "Minimum time of execution",

"ShortTitle" : "3Min",

"Description" : "This presenter presents minimum time of execution.",

"Query" : "query1.atqd",

"HasGraph" : "true",

"Xaxis" : "N",

"Yaxes" : ["Java7.Tavg", "Hoare.Tavg"],

"GraphTypes" : ["Line", "Stair"],

"XaxisLabel" : "N",

"YaxisLabel" : ["Average time"]

"HasTable" : "true",

"Columns" : ["N", "\*.Tmin"]

}

Query can alse be embeded, like

"Query" : {

"Description" : "Demo query",

"Algorithms" : ["QuickSort AS QS"],

"TestSets" : ["TestSet1 AS TS1"],

"Parameters" : ["N"],

"Indicators" : ["TMin", "TMax"]

},

### Configuration of the ALGator system

#### CompCap (computer capability)

Computers in the system play different roles depending on their capabilities.

Capability Meaning

AEE\_EM this computer can execute measurements of time

AEE\_CNT this computer can execute measurements of counters

AEE\_JVM this computer can execute measurements of jvm bytecodes

AEE\_QUICK this computer can execute quick tests

WEB this computer is a web server (ALGator homepage)

#### Computer

Describes the computer.

**File type**: JSON

**EntityID:** Computer

**Computer:**

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| ComputerID | String | ID of this computer; if ALGator can not identify the ID, CO is assigned as generic id |
| Name | String | Name of the computer |
| Description | String | Description of the computer (including comp. specificities) |
| IP | String | IP address of this computer |
| Capabilities | CompCap[] | The capabilities of this computer |

Example

{

ComputerID : "C1",

FamilyID : "F1",

Name : "kepica",

Description : "My notebook",

CompIP : "212.235.189.120",

Capabilities : ["AEE\_EM", "AEE\_CNT"]

}

#### Computer Family

Describes the hardware and software information of computers. All computers in one family have the same characteristics (hardware, operating system and instaled software) .

Intent: each computer in the system belongs to a computer family. The results produced by two computers belonging to the same family are equivalent.

**File type**: JSON

**EntityID:** Family

**Family:**

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| FamilyID | String | ID of the family |
| Name | String | Name of the family |
| Description | String | Description of this family |
| Platform | String | The platform (i.e. Linux, ...) |
| Hardware | String | Info. about hardware |
| SystemType | String | Values: 32 or 64 |
| Computers | Computer[] | A list of all computers belonging to this family |

Example:

{

FamilyID : "F1",

Name : "Mac",

Description : "My Mac computer (kepica)",

Platform : "MacOS, 10.9.5",

Hardware : "2,66GHz Intel Core 2Duo, 8GB DDR3",

SystemType : "64",

Computers : [

{

ComputerID : "C1",

FamilyID : "F1",

Name : "kepica",

Description : "My notebook",

IP : "212.235.189.120",

Capabilities : ["AEE\_EM", "AEE\_CNT"]

}

  ]

}

#### Global configuration

The configuration of the ALGator system is placed in (shared) data\_root folder. Every computer that shares data\_root folder can read/write configuration files. This configuration resides in <data\_root>/global.acfg file.

##### global.acfg

**File type**: JSON

**EntityID:** Config

**File extension**: atgc

**File location**: <data\_root>/global.acfg

**Fields in computer.atcf file:**

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| ComputerFamilies | Family[] | An array of computer families |

#### Local configuration

A configuration of a single ALGator computer.

**File type**: JSON

**EntityID:** Config

**File extension**: acfg

**File location**: <data\_local>/local.acfg

**Fields:**

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| FamilyID | String | The identification of computer's family |
| ComputerID | String | The identification of computer |
| VMEP | String | A path to the vmep executable |
| VMEPClasspath | String | Classes (or JARs) that have to be added to the class when executing VMEP |
| TaskServerName | String | IP or computer name of a task server; defult: localhost |