Cel mai usor mod de a intelege felul in care functioneaza anumite rutine este prin exemplu si lucru practic. Consider ca aceasta aplicatie imbina nevoia de a experimenta cu o reprezentare grafica usor de urmarit. Astfel, putem explica mult mai usor niste probleme specifice algoritmicii grafurilor, care, de altfel ar fi fost mult mai greu de urmarit.

Un graf este format dintr-o multime finita de noduri si una de muchii, fiecare muchie avand rolul de a conecta o pereche de noduri. Exista doua tipuri de grafuri: grafuri orientate si neorientate. Deci, graful, ca structura de date, poate fi folosit foarte usor pentru a descrie retele si comportamente din viata reala, de la multimea strazilor dintr-un oras la diferitele relatii ce atasate unui grup de prieteni.

Modelarea grafurilor la probleme reale a dus la punerea accentului pe astfel de structuri de date. S-au dezvoltat o multitudine de algoritmi capabili sa resolve probleme specifice, de la parcurgeri si gasirea celor mai scurte cai la probleme de conectivitate si de gasire a fluxului maxim. Consider ca aceasta aplicatie aduce o reala imbunatatire la aplicatiile deja existente, ajutand atat studentii cat si alte persoane pasionate de domeniu in intelegerea mult mai usoara a unor algoritmi specifici grafurilor.

Ideea mea pentru acest proiect de licenta este dezvoltarea unei aplicatii desktop ce poate servii drept unealta educativa in materia “Algoritmica Grafurilor”. Aplicatia permite utilizatorului sa scrie sau sa incarce un algoritm specific si sa il ruleze pe un graf desenat anterior intr-o alta fereastra. Aplicatia trebuie sa ofere controale cat mai intuitive in ceea ce priveste editarea codului si a grafului. La rularea algoritmului aplicatia trebuie sa ofere utilizatorului controlul asupra vitezei la care algoritmul opereaza asupra grafului, pentru a asigura coerenta animatiilor. Totodata mi-am propus sa dezvolt si un generator de grafuri pe baza de constrangeri, pentru a ajuta la testarea rapida a algoritmilor.

O aplicatie asemanatoare ar fi VisuAlgo. Aceasta lucreaza pe mai multe structuri de date: liste, grafuri, vectori si altele. Fiind o aplicatie web destul de cunoscuta, VisuAlgo se foloseste de niste animatii extrem de bine puse la punct pentru a demonstra felul in care functioneaza o serie de algoritmi, am incercat sa reproduc pe cat posibil acest look&feel. Totusi exista si un mare neajuns, VisuAlgo nu permite utilizatorului sa scrie un nou algoritm si nici sa modifice unul deja existent. Acest lucru impiedica procesul de experimentare, care, din punctul meu de vedere, este crucial in buna intelegere a unor rutine.

O alta aplicatie similara este Greenfoot, menita sa ajute in invatarea programarii in limbajul Java, aceasta ofera uneltele necesare creari unor jocuri 2D. Avand un scop mult mai extins, Greenfoot m-a ajutat in a intelege cat de important este un mod inteligent de derulare a evenimentelor reprezentate grafic.

Proiectul este realizat in limbajul de programare Java. In cadrul dezvotarii aplicatiei m-am folosit de JavaFX, o platforma software dedicata creeari de aplicatii client moderne, care pot fi rulate pe o gama larga de sisteme de operare. Am mai utilizat RichTextFX, o librarie specifica JavaFX, pentru a avea acces la evidentierea sintaxei, numerotarea liniilor si alte caracterisitici necesare unui editor de cod modern. Majoritatea stilizarilor din aplicatie au fost realizate programatic, prin metodele JavaFX, insa, pentru o mai buna organizare am folosit si mici fragmente de stilizare css.

Pentru compilarea si rularea codului la runtime m-am folosit de clasa JavaCompiller oferita de catre system si de Reflection, un API Java folosit pentru a modifica, examina si rula metode ale unor obiecte java la runtime. Prin Reflection am avut posibilitatea de a instantia clasa fisierului rezultat in urma compilarii codului scris de catre utilizator. Formatarea codului respectiv a fost posibila folosind API-ul Google Java Format, care formateaza surse java dupa standardele cerute de catre Google.

Specificatii Functionale

Scopul aplicatiei este dezvoltarea unei aplicatii desktop capabila sa ofere utilizatorilor o platforma prietenoasa in ceea ce priveste scrierea si reprezentarea pasilor unor algoritmi specifici grafurilor. Odata cu rularea unui algoritm aplicatia trebuie sa ofere utilizatorului posibilitatea de a controla dupa bunul plac modul si viteza cu care respectivul algoritm manipuleaza graful. Utilizatorul poate alege sa suspende flow-ul algoritmului sau sa avanseze manual sau chiar automat prin pasii acestuia.

O alta functionalitate majora este generatorul de grafuri pe baza de constrangeri. Odata pornit, acesta interogheaza utilizatorul cu privire la constrangerile dorite, ca mai apoi sa incerce sa genereze un graf ce se incadreaza in limitele impuse de catre utilizator. Aceste constrangeri ar putea viza numarul de noduri sau muchii, conectivitatea grafului, valorile etichetelor componentelor si altele.

Am dedus ca pentru a duce la sfarsit tot ceea ce mi-am propus voi avea nevoie sa dezvolt 4 mari module: un editor de cod, un editor de grafuri, un sistem solid de management a resurselor utilizatorului si o infrastructura inteligenta care face legatura intre cod si graf, rezultand o serie de animatii coerente.

Editorul de Cod:

Editorul de cod este o fereastra ce permite utilizatorului sa isi scrie propriul program ce va manipula un graf. Am ales limbajul de programare Java, ca limbaj folosit de utilizator pentru contructia algoritmilor, fiind modern, cunoscut si cu o sintaxa prietenoasa. Permite toate operatiunile necesare unui editor de text: copiere, taiere, lipire, selectare etcetera. Fiind dezvoltat cu ajutorul RichTextFX, acesta profita de anumite caracteristici importante ca evidentierea sintaxei si numerotarea liniilor. Acest editor prezinta urmatoarele butoane si functii:

**Save**: permite salvarea progresului intr-un fisier ce poate fi ulterior deschis.­­­

**Compile**: compileaza codul sursa scris de catre utilizator, este necesara apasarea lui inainte de rulare. In cazul in care exista erori de compilare editorul de cod va evidentia acest lucru printr-un semn specific la linia cu pricina.

**Format**: preia codul sursa scris de catre utilizator si il formateaza comform standardelor impuse de Google.

**Run**: se foloseste de fisierul produs prin compilarea codului si, avand acces la graful desenat in cealalta fereastra, incepe procesul de reprezentare grafica a algoritmului.

Editorul de grafuri:

Este o functionalitate extrem de importanta in aplicatie. Pe langa necesitatea evidenta de a permite utilizatorului sa creeze grafuri dupa bunul plac, editorul de grafuri reprezinta totodata si locul in care se desfasoara toate animatiile si manipularile survenite in urma rularii unui algoritm.

Exista mai multe librarii care se ocupa cu reprezentarea grafica a grafurilor intr-o aplicatie java. Una dintre acestea este JGraphX care, avand la baza structura de date definite de libraria JGraphT, poate afisa cu usurinta grafurile necesare. Totusi, acestea nu au suport JavaFX, ci doar Swing. Desi exista posibilitatea importului de componente Swing intr-o aplicatie JavaFX, am decis ca din pricina acestui inconvenient dar si a altor motive precum aspectul nesatisfacator si lipsa controlului total asupra functionalitatii grafurilor, imi voi dezvolta propria modalitate de a reprezenta grafuri in JavaFX.

In ceea ce priveste infatisarea grafului am optat pentru o tema usor de inteles, punand pe primul loc coerenta si vizibilitatea datelor. Ca paleta initiala de culori am ales alb si negru, sperand sa scot in evidenta animatiile si culorile alese de catre utilizator.

Nodurile sunt reprezentate vizual de cercuri albe cu un contur negru. La dorinta utilizatorului , nodurile pot contine o etucheta cu un nume sau o valoare numerica. Centrata in interiorul nodului, aceasta eticheta este de o culoare cenusie transparenta, pentru a nu obstruction prea mult reprezentarea nodului.

Muchiile, pe de alta parte, sunt linii negre ce conecteaza doua noduri. La fel ca nodurile, muchiile pot sau nu avea etichete cu valori, care, de aceasta data, vor fi asezate la mijlocul liniei. In cazul in care avem de-a face cu un graf orientat, muchiile vor arata “cursul” sursa-destinatie printr-o sageata prezenta la unul dintre capete. In cazul in care muchia respective este una dubla, reprezentarea grafica a acesteia va infatisa sageti la ambele capete.

Graful rezultat, reprezentat in editor, este in totalitate responsive. Utilizatorul poate modifica pozitia oricaror componente prin drag and drop, iar starea componentelor afectate se va schimba in mod corespunzator. Editorul de grafuri prezinta urmatoarele butoane si functii:

**Save**: similar editorului de cod, permite salvarea locala a progresului realizat in graful curent.

**Add Edges**: schimba modul de lucru asupra grafului pentru a permite utilizatorului sa adauge muchii. O muchie este adaugata atunci cand doua noduri sunt selectate consecutive. Primul nod reprezinta sursa, iar cel de-al doilea destinatia.

**Add Nodes**: Permite adaugarea de noduri, schimband modul de lucru asupra grafului. Un nod este adaugat la apasarea pe o zona libera din editor.

**Delete Components**: Permite stergerea de muchii si noduri odata cu selectarea acestora. O apasare dubla a acestui buton va conduce la stergerea intregului graf.

**Edit**: Fiecare componenta a grafului are o eticheta cu valori care este ascunsa atunci cand component nu are valoare. Acest buton faciliteaza aparitia etichetelor pentru a putea fi editate.

**Switch**: Schimba arhitectura grafului din orientat in neorientat si invers. Este atat o schimbare de reprezentare grafica cat si una pe partea de “back-end”.

**Generate**: Porneste generatorul de grafuri pe baza de constrangeri. Voi discuta modul in care functioneaza mai jos.

Management-ul datelor

Atat algoritmii scrisi, cat si grafurile desenate pot fi salvate local si incarcate la dorinta utilizatorului. Acestea vor fi stocate sub forma de fisiere intr-un folder vecin al proiectului, fiecare fisier avand un nume unic selectat de utilizator. Algoritmii sunt salvati ca fisiere .java , facand astfel mai usoara compilarea si utilizarea lor. Avand in vedere simplitatea resursei folosite de editorul de cod ( algoritumul fiind un simplu string ), aceasta a fost cea mai intuitiva varianta.

Pe de alta parte, in cazul grafurilor nu am avut o solutie asa de usoara. Am inceput prin a le serializa binar si stoca in fisiere. Aceasta varianta a fost foarte usor de implementat, serializarea obiectelor in java este usoara si bine documentata. Totusi, exista cateva neajunsuri. In cazul aplicatiei mele, clasa grafului este una in continua schimbare, in sensul in care comportamentul acesteia este imbogatit regulat. Acest lucru nu este suportat de catre serializarea binara, intrucat aceasta descrie un model anterior al clasei, asadar la fiecare schimbare a clasei Graph, toate obiectele serializate anterior vor fi pierdute.

Cautand o solutie adecvata am descoperit formatul de fisier GraphML. Acesta este un format intuitiv si unor de folosit pentru grafuri. Are suport pentru grafuri orientate si neorientate, si este usor de extins pentru a include diferite atribute nodurilor si muchiilor. Am cautat o biblioteca capabila sa parseze un graf si sa imi ofere serializarea acesteia GraphML, insa nu am gasit unul potrivit, fiecare fiind specializat pe grafuri descrise de librarii ca JGraphX sau, in cazul celor care puteau lucra cu un graf custom, adaugau prea multe atribute nefolosite de modelul meu de graf, rezultand o serializare greu de urmarit.

Asadar, mi-am facut propriul parser de grafuri, capabil sa transforme modelul meu de graf intr-un fisier GraphML si viceversa. GraphML este un format bazat pe XML, deci nu are o sintaxa stricta, urmand setul de reguli propus de documentatie am reusit sa fac un parser satisfacator. Cu ajutorul acestuia am rezolvat problema schimbarii modelului de graf, atata timp cat nu modific definitia grafului in sine (o multime de noduri si o multime de muchii ce conecteaza perechi de noduri), pot adauga si modifica functionalitati, serializarea GraphML v-a descrie corect modelul de graf. Odata cu acest avantaj mai apare unul. Standardul GraphML este destul de folosit si in alte aplicatii de constructie de grafuri, ceea ce inseamna ca este foarte usor sa exportam un graf din NumeAplicatie si sa il folosim in respectivele aplicatii si invers. Un user poate prelua un graf interesant si il poate incarca in NumeAplicatie, atata timp cat serializarea acestuia respecta standardul GraphML, lucru imposibil de realizat folosind serializarea binara.

Flow-ul de lucru:

Ce se intampla cu adevarat la apasarea butonului “Run”?

Pentru inceput se verifica existenta fisierului produs in urma compilarii codului scris de catre utilizator. Apoi, folosindu-ne de Reflection API cream o instanta a clasei descrise de acel fisier. Un lucru foarte important de mentionat este ca obiectul rezultat trebuie sa fie o instanta a clasei Program, astfel incat putem fi siguri de implementarea unor metode cruciale in rularea algoritmului, si anume, a metodei “run”. Daca aceste conditii sunt indeplinite vom avansa la urmatoarea etapa.

Aplicatia isi propune reprezentarea manipularii grafului intr-un mod controlat de catre utilizator. Asta presupune derularea manuala sau, setand un interval, automata a comenzilor asupra grafului. Deci nu putem permite algoritmului sa lucreze direct pe graful din editor, intrucat asta ar insemna ca modificarile asupra grafului vor avea loc odata cu rularea codului, mult prea rapid pentru a putea fi urmarite de catre utilizator. Aceasta problema poate fi rezolvata in doua moduri:

* Putem “coopera” cu utilizatorul. Instruind utilizatorul sa foloaseasca apeluri de sleep ale sistemului pentru a creea iluzia de parcurgere secventiala a comenzilor asupra grafului. Totusi, acest approach nupermite derularea manuala si nici pauza, insa cel mai mare neajuns este nevoia de a restrictiona utilizatorul din a scrie cod curat si usor de urmarit.
* Putem reduce graful reprezentat “grafic” la o structura de date “dummy” mai simpla. Astfel, atat graful cat si toate componentele sale sunt reduse la obiecte ce implementeaza comportamentul unui graf, si care ofera posibilitatea adaugarii de functionalitate. Folosindu-ne de asta am putea modifica obiectele “dummy” in mici istoricuri, capabile sa memorize toate evenimentele produse de catre utilizator. Astfel, algoritmul utilizatorului este rulat pe un graf dummy ce joaca rolul de istoric, iar la final putem culege toate evenimentele, ca mai apoi sa fie rulate pe graful autentic, dupa bunul plac.

Evident, am ales varianta a doua. Astfel, preluam graful din editor si ne folosim de el pentru a creea un graf dummy. In continuare algoritmul utilizatorului este rulat prin Reflection API, oferindu-I graful dummy ca parametru. Rezultatul rularii este o lista de comenzi, care, puse in ordine descriu perfect evenimentele ce trebuie parcurse pentru a respecta algoritmul utilizatorului. Aceasta lista de comenzi este pasata mai departe unei clase CommandsRunner care, folosindu-se de input-ul utilizatorului, ruleaza comenzile in modul dorit. Vom discuta amanuntit aceasta functionalitate mai jos.