Social Network Challenge

# Challenge

Let's build a social network.

In this social network, each user has friends.

A chain of friends between two users, user A and user B, is a sequence of users starting with A and ending with B, such that for each user in the chain, ua, the subsequent user, ua + 1, are friends.

Given a social network and two users, user A and user B, please write a function that computes the length of the shortest chain of friends between A and B.

# Discussion

Am abordat solutia oferită într-o manieră centrată pe testarea eficientei funcței cerute si nu neaparat pe construirea unui prototip simplist al unui social network, cu alte cuvinte realizarea unui API.

Limbajul folosit este C#, iar mediul de programre Visual Studio. Decizia de a folosi C# cu toate ca exista alte limbaje percum C sau C++ mai rapide a fost determinată de intenția de a ma familiariza mai mult cu acesta.

1. How did you represent the social network? Why did you choose this representation?

Pentru reprezentarea social network-ului, am găsit potrivită reprezentarea sub forma unui graf. A urmat întrebarea cum va fi reprezentat graful în implementarea propriu-zisă. Cele mai uzuale opțiuni fiind printr-o matrice de adiacență sau o listă de adiacență. Următoarele aspecte m-au determinat să optez pentru "lista de adiacență":

• Un social network are dimensiuni foarte mari (Facebook: 27 \* 10 ^ 8 conturi)

• Complexitățile de spațiu ale celor două abordări sunt: O(V^2) pentru matrice și O(V + E) pentru listă, unde V = numărul de noduri și E = numărul de legături.

• Este destul de clar acum că din punctul de vedere al gestionării memoriei avem de comparat V^2 cu V + E.

• Facem doar un calcul intuitiv: V = 27 \* 10 ^ 8; E = V \* 200 (numărul aproximativ de prieteni pe care o persoană îi are pe Facebook, documentat de pe internet); (V^2) / (201) \* V = V / 201, deci abordarea cu matrice n-ar fi decât de 13432836 mai costisitoare ca memorie comparând cu cea cu liste. A urmat implementarea unei clase, NetworkGraph, care conține metode CRUD pentru modificarea structurii grafului la adăugarea/ștergerea unui cont sau modificarea unei relații de prietenie. Modul de reprezentare a grafului propriu-zis a fost următorul: Dictionary<int, HashSet<int>>. Ambele structuri au complexitatea O(1) atât la inserare cât și la căutare și ștergere în cazul normal. Cheia dicționarului o reprezintă contul, iar valoarea este un set (fiecare element este un cont) de prieteni (lista de prieteni)..

1. What algorithm did you use to compute the shortest chain of friends? What alternatives did you consider? Why did you choose this algorithm over the alternatives?

Algoritmul ales în final a fost BFS (Breadth-First Search). Altă alternativă considerată a fost Dijkstra. Deși încă de la început eram oarecum convinsă că BFS va fi algoritmul mai eficient, decizia a fost luată prin teste. Deși aveam de gând să implementez varianta optimizată prin calcul paralel pentru ambii algoritmi, am pornit de la implementările simple. Diferența timpilor de executie a fost atât de evidentă încât Dijkstra implementat cu calcul paralel reușise abia să aibă rezultate comparabile cu BFS-ul neoptimizat.

1. Please enumerate the test cases you considered and explain their relevance.

* Prima problemă întâmpinată în considerarea cazurilor de test a fost generarea unui graf relevant. Testarea pe un graf implementat manual cu 20 de noduri interconectate aleatoriu ar fi fost în mod evident irelevantă. Am început documentarea în ceea ce privește structura unui social network. Am ales Facebook ca exemplu. Informațiile obținute au fost: 2.700.000.000 de utilizatori
* O distributie power law cu coeficientul 2.3-2.5 a numărului de prieteni/utilizatori

f(x) = (1 / C) \* x^(-α)

f(x) = -x^(-α) / α, C = - α

F(x) = integrală(f(x)) = x^(1- α)

Inversa: F ^ (-1) (y) = Y ^ (1/(1- α)), de unde și codul utilizat:

var friendCount = Math.Round(Math.Pow(random.NextDouble(), -1.0 / (alpha - 1.0)));

Deși am paralelizat generarea grafului, pentru viabilitatea intervalului de timp (maxim 20 de secunde) a fost imposibilă generarea a mai mult de 27 \* 10 ^ 6 noduri, motiv pentru care rețeaua noastră va fi de 100 de ori mai restrânsă ca Facebook-ul.

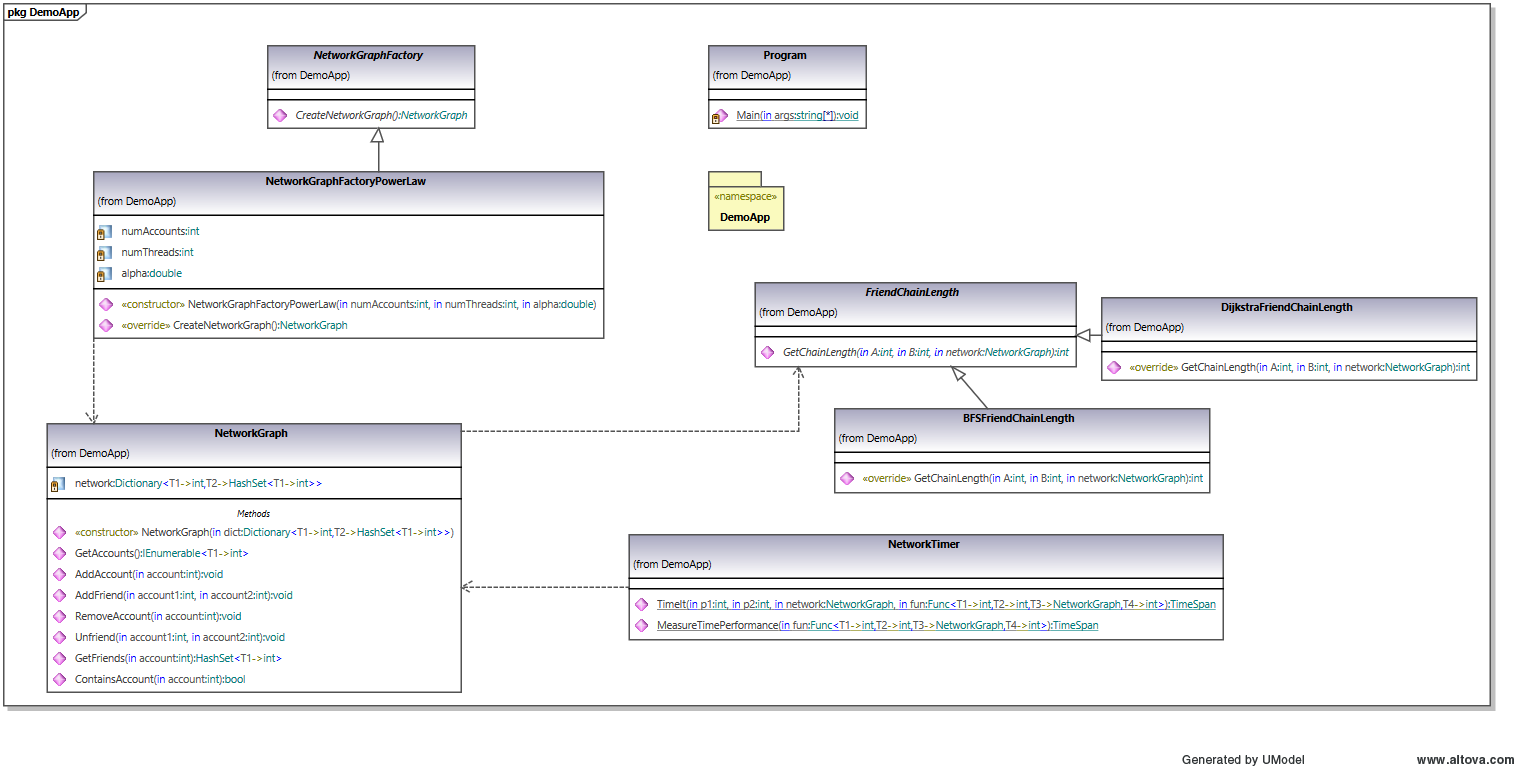
S-au efectuat 2 tipuri de măsurători:

* Scenariul automat: A si B sunt alese random, dimensiunea grafului este de 27.000.000 noduri, se face câte o măsuratoare pentru fiecare algoritm.
* Scenariul manual: prim intermediul meniului din consolă se introduce dimensiune social newtwork-ului si nodurile A si B intre care sa se calculeze distanța.

Din persepectiva implementării am considerat o clasă separată NetworkTimers care utilizează ca instrument de măsFacurarea performanțelor metodei aplelate clasa System.Diagnostics.StopWatch.

Pentru că s-a considerat că un social network ar putea urmări și alte distribuții inafară de cea power low, s-a utilizat și metoda toy pentru crearea de obiecte de tip NetworkGraph.

**Diagrama UML a proiectului de test:**



**Rezultate ale câtorva măsurători:**

****