Социална Мрежа

1. Анализ на задачата- задачата за социална мрежа изискваше да могат да бъдат постигнати неща като добавяне на потребители, премахването им, търсенето в тях и създаването на различни видове приятелства между тях, както и бан. Важно за целта на задачата бе възможността тези данни да бъдат запаметени и след затварянето на програмата. За тази цел аз използвах два файла в които да съхранявам информацията, след затварянето и. Един със списък с потребителите и втори с техните приятелства. Реших потребителите да ги записвам в бинарен файл, а приятелствата в един текстов файл в който да пазя матрицата на съседство на графа с който имплементирам приятелствата. За целта аз използвам ориентиран граф като при:

   0-потребителите са забранени(banned)
   1-потребителите не са приятели нито забранени
   2-потребителите са приятели
   3-потребителите са семейство
   4-потребителите са най-добри приятели

   Важна и същинска част от проекта е възможността за препоръки приятелство. Тази функционалност трябваше да бъде реализирана в два различни случая.

   В първият случай на тази функционалност даденият потребител няма никакви приятели. В този случай подходът който използвах бе да намеря хората с най-много приятелства в нашата социална мрежа и да му предложа именно тях.

   Във вторият случай нашият потребител вече има няколко свой приятеля. В този случай подходът който използвам е търсенето на приятелите на всеки от неговите приятели и записвайки ги в списък заедно с една допълнителна променлива. Тази променлива пази колко близък е нашият приятел. В този случай ако например Петър ни е най-добър приятел(bestie) то неговите приятели ще ни излизат първи като предложения.

2. Описание на класовете- За реализирането на задачата аз се придържах към ООП парадигмата. Всяка една от допустимите

команди за нашата социална мрежа е представена като наследник на виртуалният клас Command. В нашата main функция се чете до прочитането на командата изход(exit). След прочитането на даден текст тази команда се дава за разбиване на думи и първата от тях се анализира дали е от допустимият списък с команда за нашето приложение. Ако е така, то се създава команда от този тип с нужните параметри и на тази вече детерминирана команда, с прикачени нужните параметри се извиква изпълни. След като е извикана функцията за изпълнение на нашата команда, то вече самата функция върши нужните неща и извиква за изпълнение вече нужната функционалност която вече върши нужната работа.
За достъп през цялото време до нашият граф и нашите потребители съм създал два класа. Единият за динамичен масив, а вторият за динамичен граф. В конструкторите на тези два класа се зарежда информацията от файловете в тях, а в де структурите тя се записва. Именно те предоставят нужната функционалност за работа с данните.

3. Идеи за бъдещи подобрения- Идеи които имам за бъдещи подобрения са намаляването на размера на матрицата на съседство, защото нашето приятелство е симетрична релация. Също така смятам за хубаво подобрение внедряването на по добър алгоритъм за предложения за приятелство. Друг проблем който виждам и смятам, че в бъдещи реализации той е добре да бъде поправен е, ако имаме наличие на огромно количество потребители, толкова много, че да не могат да бъдат заредени в нашата оперативна памет. Използването на друга структура така, че търсенето по име да не отнема O(n) време. Използването на std::set вместо std::vector, така че да използваме binary search и търсенето ни да е с логаритмично време, но тази промяна пък ще доведе до търсене с голяма сложност по id затова ни трябва комбиниране структура в която да използваме двоичното търсене както и по id така и по name, но пък сега достъпа ни по-време е константен, така че това трябва да се обмисли според нуждите дали би било по-добре или не.
4. GitHub Repository- https://github.com/5ko99/SDP-Practicum-Social-Network