Aufgabe 1: Superstar

Symbroson
Team-ID: 00165

24. November 2018

Inhaltsverzeichnis

1 Lösungsidee

Jeder User steht mit allen Anderen in Beziehung und jede Beziehung schließt jemanden davon aus, der Superstar zu sein: Folgt User A User B, kann User A kein Superstar sein, da der Superstar niemandem folgen darf, und die Wahrscheinlichkeit, dass User B der Superstar ist, steigt. Folgt User A User B nicht, kann User B kein Superstar sein, da alle anderen dem Superstar folgen müssen, dafür kan User A weiterhin ein Superstar sein.

Somit kann im ersten Schritt mit n-1 Abfragen ein einziger User bestimmt werden, der der Superstar sein kann. Danach muss dieser nur noch durch die Bedingungen, dass alle dem Superstar und der Superstar niemandem folgt, validiert werden, da es sein kann, das er von jemandem nicht gefolgt wird der noch nicht geprüft wurde, oder dass er jemandem folgt, der schon vorher ausgeschlossen wurde.

Um doppelte Anfragen zu vermeiden, können außerdem bereits abgefragte Beziehungen gespeichert werden.

2 Umsetzung

2.1 User

Jeder User wird durch eine Klasse dargestellt, die den Namen, eine ID (die dem Index in der User-Liste entspricht), einen Wahrheitswert der anzeigt, ob der User ein Superstar sein kann und eine Liste von Usern, denen er folgt, enthält. Letztere wird während des Einlesens der Eingabedatei initialisiert.

Beim Einlesen der Eingabedatei werden zusätzlich alle User in der users-Liste gespeichert, auf die dann in der Gefolgten-Liste nur noch per ID referenziert werden muss.

2.2 Anfragenspeicherung

Die Anfragen werden in einem (eigentlich zweidimensionalen, in diesem Fall wegen C++ in die erste Dimension projiziertes) Array gespeichert. In diesem Array sind für jede mögliche Beziehung 3 verschiedene Status Möglich: noch nicht abgefragt, folgt oder folgt nicht.

2.3 Superstarfindung

Der erste Schritt besteht nun darin, den ersten User als Superstar anzunehmen und die Beziehung zum folgenden User zu prüfen. Folgt User1 User2, wird User2 der neue Superstar und User1 ausgeschlossen, andernfalls umgekehrt. Danach wird die Beziehung des verbleibenden Superstars zum dritten User abgefragt, ein User ausgeschlossen usw.. Dies wiederholt man n-1 mal, sodass am Ende nur ein User als Superstar infrage kommt.

2.4 Superstarvalidierung

Die Validierung erfolgt dann nurnoch durch Abfragen zwischen allen Usern und dem Superstar und umgekehrt. Somit entstehen im Worst-Case noch 2*(n-1) Anfragen, also insgesamt 3*(n-1) Anfragen. Im Best-Case ist der erste gelistete User bereits der Superstar und es muss nur noch geprüft werden, ob alle anderen ihm folgen, was noch einmal n-1, also insgesamt 2*(n-1) Anfragen erfordert.

Gibt es keinen Superstar in der Gruppe, können im Best-Case n Anfragen ausreichen, wenn der zweite User dem ersten folgt.

2.5 Ausgabe

Es werden zunächst alle User mit deren entsprechenden ID im Format (ID:Name) ausgegeben, auf die sich dann in Folgenden Ausgaben bezogen wird. In den Abfragetabellen werden in dieser Reihenfolge erst der aktuell angenommene Superstar, der User zu dem die Beziehung getestet wird, das Ergebnis der Anfrage als Wahrheitswert, die Kosten, die die Anfrage verursacht hat (0, wenn bereits zuvor dieselbe Anfrage getätig wurde, sonst 1), und am Ende der Zeile die aktuelle Gesamtsumme der bereits entstandenen Kosten. Zwischen Superstarerkennung und Superstarvalidierung wird noch einmal der aktuell vermutete Superstar namentlich ausgegeben. Nachdem die Existenz eines Superstar ermittelt wurde, erfolgt eine entsprechende Ausgabe, die Anzahl der Personen samt entstandener Kosten und die Preisschätzung.

3 Beispiele

3.1 superstar1.txt

Ausgabe:

```
Names: 0:Selena 1:Justin 2:Hailey

Star User Follows Cost | Sum
   0    1    true    1 |   1
   1    2   false    1 |   2

Suspected star: Justin

Star User Follows Cost | Star User Follows Cost | Sum
   1    0   false    1 |    0    1    true    0 |   3
   1    2   false    0 |    2    1    true    1 |   4

Justin is the superstar!

Persons:    3
Price:      4€

Estimation (on success):
 Worst:     6€
 Best:      4€
```

3.2 superstar3.txt

Ausgabe:

```
Names: 0:Edsger 1:Jitse 2:Jorrit 3:Peter 4:Pia 5:Rineke 6:Rinus 7:Sjoukje

Star User Follows Cost | Sum
   0    1    true    1 |   1
   1    2   false    1 |   2
   1    3   false    1 |   3
   1    4   false    1 |   4
   1    5   false    1 |   5
   1    6   false    1 |   6
   1    7   false    1 |   7

Suspected star: Jitse

Star User Follows Cost | Star User Follows Cost | Sum
   1    0    true    1 |

Termination due to (1:Jitse) (0:Edsger)
There is no superstar in this group.

Persons:    8
Price:      8€

Estimation (on success):
 Worst:    21€
 Best:     14€
```

4 Quellcode

```
/* ... */

#define FOLLOW_NOP 0 // folgt nicht
#define FOLLOW_UKN 1 // nicht abgefragt
#define FOLLOW_YES 2 // folgt

struct User {
    char* name;            // Name
    uint id;               // ID = users-Index
    bool star;             // Status: kann Superstar sein
    vector<User*> follows; // Folgt-Liste
};

vector<User> users; // User-Liste

uint count = 0, // gesamt User-Anzahl
    stars  = 0, // verbleibende Superstars
    cost   = 0, // entstandene Kosten
    *follow;    // gespeicherte Abfragen

/* ... */

bool follows(User& a, User& b, bool nobrk) {
    printf("%4i %4i ", a.id, b.id);
    uint* folw = follow + (a.id * count + b.id);

    switch (follow[a.id * count + b.id]) {
        case FOLLOW_NOP: printReq(false, 0, nobrk); return false;
        case FOLLOW_YES: printReq(true, 0, nobrk); return true;

        default:
            cost++;
            // suche Übereinstimmung
            for (User* u: a.follows) {
                if (u == &b) {
                    printReq(true, 1, nobrk);
                    *folw = FOLLOW_YES;
                    if (a.star) {
                        a.star = 0;
                        stars--;
                    }
                    return true;
                }
            }

            printReq(false, 1, nobrk);
            *folw = FOLLOW_NOP;
            if (b.star) {
                b.star = 0;
                stars--;
            }
            return false;
    }
}

int main(int argc, const char* argv[]) {
    uint i;

    /* ... */

    // Namen mit ID ausgeben
    printf("\nNames:\n");
    for (User& a: users) printf(" %i:%s", a.id, a.name);
    printf("\n\nStar User Follows Cost | Sum\n");

    uint j, _follow[count * count];

    stars  = count;
    follow = _follow;
```

```
    // resette Folgebeziehungen
    for (i = 0; i < count; i++)
        for (j = 0; j < count; j++) follow[i * count + j] = FOLLOW_UKN;

    // schließe User als Superstar aus
    User *first = &users[0], *second;
    while (stars > 1) {
        second = NULL;
        // wähle zwei verbleibende Stars aus
        for (User& a: users) {
            if (a.star && &a != first) {
                second = &a;
                break;
            }
        }

        // speichere verbleibenden Star in first
        if (follows(*first, *second, false)) first = second;
    }

    printf(
        "\n\033[0;33mSuspected star: %s\033[0;37m\n\nStar User Follows Cost | "
        "Star User Follows Cost | Sum\n",
        first->name);

    // validiere Star
    for (User& b: users)
        if (&b != first &&
            (follows(*first, b, true) || !follows(b, *first, false))) {
            printf(
                "\n\nTermination due to (%i:%s) (%i:%s)", first->id,
                first->name, b.id, b.name);
            break;
        }


    // Ausgabe
    if (stars == 1)
        printf("\n\033[0;32m%s is the superstar!\033[0;37m\n", first->name);
    else
        printf("\n\033[0;33mThere is no superstar in this group.\033[0;37m\n");

    printf(
        "\nPersons:%5i\nPrice:  %5i€\n"
        "\nEstimation (on success):\n Worst:  %4i€\n Best:   %4i€\n",
        count, cost, 3 * (count - 1), 2 * (count - 1));


    freeSuperstar();
    fclose(fp);
}
```