# Aufgabe 1: Superstar

Symbroson
Team-ID: 00165

25. November 2018

## Inhaltsverzeichnis

**1 Lösungsidee** **1**

**2 Umsetzung** **1**
2.1 User . . . . . 1
2.2 Anfragenspeicherung . . . . . 1
2.3 Superstarfindung . . . . . 2
2.4 Superstarvalidierung . . . . . 2

**3 Quellcode** **2**

## 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 boolean, der anzeigt, ob der User ein Superstar sein kann und eine Liste von Usern, denen er folgt, enthält.

Beim Einlesen der Eingabedatei werden dann zunächst alle User in der User-Liste gespeichert, auf die dann bei der Gefolgten-Liste nur noch 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 also 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 dem Superstar mit allen Anderen. Somit entstehen im Worst-Case noch zwei mal 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 insgesamt n-2 Anfragen erfordert.

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

## 3 Quellcode

```
/* ... */

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

struct User {
    char* name;            // Name
    uint16_t id;           // ID = users-Index
    bool star;             // Status: kann Superstar sein
    vector<User*> follows; // Folgt-Liste
};
vector<User> users; // User-Liste

uint16_t count = 0, // gesamt User-Anzahl
    stars      = 0, // verbleibende Superstars
    cost       = 0; // entstandene Kosten

uint8_t* follow; // gespeicherte Abfragen

/* ... */

bool initUsers(FILE* fp) {
    char *line = NULL, *pch;
    ssize_t read;
    size_t len = 0;

    // lese Namenliste
    read = getline(&line, &len, fp);
    if (read == -1) {
        error("Error reading file");
        return true;
    }

    char buffer[read];
    strncpy(buffer, line, read);

    // parse Namenliste
    pch = strtok(buffer, " \n");
    while (pch != NULL) {
        if (isalpha(*pch)) {
            users.push_back({.name    = strdup(pch),
                             .id      = count,
                             .star    = true,
                             .follows = {}});
```

```
            count++;
        }
        pch = strtok(NULL, " \n");
    }

    // lese Folgebeziehungen
    User* cur = NULL;
    while ((read = getline(&line, &len, fp)) != -1) {
        if (!(pch = strtok(line, " \n"))) continue;
        if (cur == NULL || strcmp(pch, cur->name)) cur = findUser(pch);
        cur->follows.push_back(findUser(strtok(NULL, " \n")));
    }

    free(line);
    return false;
}

/* ... */

bool follows(User& a, User& b, bool nobrk) {
    printf("%4i %4i%4i ", stars, a.id, b.id);
    uint8_t* folw = follow + (a.id * count + b.id);

    switch (follow[a.id * count + b.id]) {
        case FOLLOW_NOP: printf(" nein    * |%c", "\n "[nobrk]); return false;
        case FOLLOW_YES: printf("   ja    * |%c", "\n "[nobrk]); return true;

        default:
            cost++;
            // suche Übereinstimmung
            for (User* u: a.follows) {
                if (u == &b) {
                    printf("   ja%5i |%c", cost, "\n "[nobrk]);
                    *folw = FOLLOW_YES;
                    if (a.star) {
                        a.star = 0;
                        stars--;
                    }
                    return true;
                }
            }

            printf(" nein%5i |%c", cost, "\n "[nobrk]);
            *folw = FOLLOW_NOP;
            if (b.star) {
                b.star = 0;
                stars--;
            }
            return false;
    }
}

int main(int argc, const char* argv[]) {
    FILE* fp = NULL;

    // Datei öffnen
    if (argc > 1) tryOpen(argv[1], fp);
    if (fp == NULL && tryOpen("res/superstar1.txt", fp)) return true;

    // Datei einlesen
    if (initUsers(fp)) {
        fclose(fp);
        error("initialization failed");
        return 1;
    }

    printf("\nNamen:\n");
    for (User& a: users) printf("(%i:%s)", a.id, a.name);
    printf("\n\nstar   U1  U2 folgt Cost |\n");

    uint8_t _follow[count * count];
    uint16_t i, j;
```

```
    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(
        "\nvermuteter Star: %s\n\nstar   U1  U2 folgt Cost | star   U1  U2 "
        "folgt Cost |\n",
        first->name);

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

    // Ausgabe
    if (stars == 1)
        printf("\n\033[0;32m%s ist der Superstar!\033[0;37m\n", first->name);
    else
        printf(
            "\n\033[0;33mEs gibt keinen Superstar in dieser "
            "Gruppe.\033[0;37m\n");

    printf("\nPersonen:%4i\nPreis:   %4i\n", count, cost);
    printf(
        "\nSchätzung (bei Erfolg):\n Worst:  %4i\n Best:   %4i\n",
        3 * (count - 1), 2 * (count - 1));
    freeUsers();
    fclose(fp);
}
```