Aufgabe 5: Stadtführung

Team-ID: 00988

Team-Name: BitShifter

Bearbeiter dieser Aufgabe:
Filip Zelinskyi

21. November 2023

Inhaltsverzeichnis

1 Abstract

Das Programm macht genau das, was in der Aufgabenstellung beschrieben wurde. Es sucht geschlossene Teilrouten in der Liste. Im Falle eines Schnitts zwischen zwei Teilrouten wird jene mit der größten Distanz ausgewählt. Unser Ziel besteht darin, den kürzestmöglichen Weg zu finden.

In diesem Dokument wird der Lösungsansatz beschrieben, die Umsetzung auf C++20 erläutert und die Lösungen für die Beispiele vorgestellt.

2 Lösungsidee

Zu Beginn müssen die Punkte identifiziert werden, die aus der Tour entfernt werden dürfen. Hierbei handelt es sich um nicht essenzielle Punkte, die sich in einer geschlossenen Teilroute befinden sollen. Die Liste der Punkte in der Route sollte in chronologischer Reihenfolge vorliegen. Eine zusätzliche Sortierung ist nicht erforderlich, da der Algorithmus die Liste linear durchgeht.

Eine geschlossene Teilroute ist dadurch gekennzeichnet, dass ein Punkt innerhalb der Route zweimal auftritt, und zwischen diesen beiden Punkten dürfen nur nicht essenzielle Punkte liegen. Andernfalls sollte die Teilroute nicht für die Entfernung in Betracht gezogen werden.

Die akkumulierte Distanz zwischen den Punkten in der Route, wie in der Aufgabenstellung beschrieben, dient dazu, bei der Entscheidung zwischen schneidenden Teilrouten zu helfen. Hierbei gilt die Regel, dass größere Teilrouten bevorzugt werden. Bei der Ausgabe sollten die Distanzen entsprechend berücksichtigt werden.

Im Falle einer entfernten Teilroute sind in der Ausgabe zwei Punkte an derselben Stelle zu sehen, bei denen die akkumulierte Distanz gleich ist.

3 Umsetzung

Zunächst werden die Basisstrukturen für diese Aufgabe initialisiert, wie zuvor definiert.

```
type RoutePunkt ← (name : string, year : int, essential : bool, accDistance : int)
type TeilRoute ← (startIndex : int, endIndex : int, distance : int)
```

Die Funktion $IsValidTeilRoute$ gewährleistet, dass lediglich Teilrouten entfernt werden, die keine essenziellen Punkte enthalten.

```
function IsValidTeilRoute(route: RoutePunkt, tr: TeilRoute)
    for i ← tr.startIndex + 1 to tr.endIndex do
        if route[i] ist essenziell then
            return false
        end if
    end for
    return true
end function
```

In der Suchfunktion für Teilrouten werden Wiederholungen erkannt, und die Punkte zwischen ihnen gelten als potenzielle Teilrouten. Solche Routen, die keine essenziellen Punkte beinhalten, werden in der Liste der Teilrouten beibehalten. Anschließend erfolgt die Überprüfung der Schnittpunkte, wobei Routen mit größeren Distanzen bevorzugt werden.

```
function FindTeilRouten(route: arr(RoutePunkt))        ▷Alle Indizes von einzigartigen Punkten
    hmPunkte ← HashMap(string, arr(int))
    for i ← 0 to route.size do
        hmPunkte[route[i].name].push(i)
    end for
    result ← arr(TeilRoute)
    for j ← 0 to route.size do
        gleichePunkte ← hmPunkte[route[i].name]
        if gleichePunkte.size > 2 then
            startIndex ← gleichePunkte[j]
            endIndex ← gleichePunkte[j + 1]
            if startIndex > 1 then
                distance ← route[endIndex].accDistance − route[startIndex].accDistance
                teilRoute ← newTeilRoute(startIndex, endIndex, distance)
                if IsValidTeilRoute(route, teilroute) then
                    result.push(teilroute)
                end if
            end if
        end if
    end for
    return result
end function
```

4 Beispiele

Tour 1

```
cat examples/tour1.txt | ./a.out
Brauerei,1613,X,0
Karzer,1665,X,80
Rathaus,1678,X,150
Rathaus,1739,X,150
Euler-Br cke,1768, ,330
Fibonacci-Gaststt te,1820,X,360
Schiefes Haus,1823, ,480
Theater,1880, ,610
Emmy-Noether-Campus,1912,X,740
Emmy-Noether-Campus,1998,X,740
Euler-Br cke,1999, ,870
Brauerei,2012, ,1020
```

Tour 2

```
cat examples/tour2.txt | ./a.out
Brauerei,1613, ,0
Karzer,1665,X,80
Rathaus,1678, ,150
Rathaus,1739, ,150
Euler-Br cke,1768, ,330
Fibonacci-Gaststt te,1820,X,360
Schiefes Haus,1823, ,480
Theater,1880, ,610
Emmy-Noether-Campus,1912,X,740
Emmy-Noether-Campus,1998,X,740
Euler-Br cke,1999, ,870
Brauerei,2012, ,1020
```

Tour 3

```
cat examples/tour3.txt | ./a.out
Talstation,1768, ,0
W ldle,1805, ,520
Mittlere Alp,1823, ,1160
Observatorium,1833, ,1450
Observatorium,1874,X,1450
Piz Spitz,1898, ,1920
Panoramasteg,1912,X,2140
Panoramasteg,1952, ,2140
Ziegenbr cke,1979,X,2390
Talstation,2005, ,2670
```

Tour 4

```
cat examples/tour4.txt | ./a.out
Blaues Pferd,1523, ,0
Alte M hle,1544, ,110
Marktplatz,1549, ,210
Marktplatz,1562, ,210
Springbrunnen,1571, ,290
Dom,1596,X,360
Bogensch tze,1610, ,480
Bogensch tze,1683, ,480
Schnecke,1698,X,630
Fischweiher,1710, ,810
Reiterhof,1728,X,930
```

```
Schnecke,1742, ,1070
Gro e Gabel,1874, ,960
Fingerhut,1917,X,1030
Stadion,1934, ,1150
Marktplatz,1962, ,1240
Baumschule,1974, ,1320
Polizeipr sidium,1991, ,1460
Blaues Pferd,2004, ,1610
```

Tour 5

```
cat examples/tour5.txt | ./a.out
Gabelhaus,1638, ,0
Gabelhaus,1699, ,0
Hexentanzplatz,1703,X,160
Eselsbr cke,1711, ,280
Dreibannstein,1724, ,390
Dreibannstein,1752, ,390
Schmetterling,1760,X,540
Dreibannstein,1781, ,620
M rchenwald,1793,X,700
Fuchsbau,1811, ,780
Torfmoor,1817, ,890
Gartenschau,1825, ,1030
Riesenrad,1902, ,880
Dreibannstein,1911,X,1010
Olympisches Dorf,1924, ,1170
Haus der Zukunft,1927,X,1300
Stellwerk,1931, ,1420
Stellwerk,1942, ,1420
Labyrinth,1955, ,1630
Gauklerstadl,1961, ,1710
Planetarium,1971,X,1790
K nguruhfarm,1976, ,1840
Balzplatz,1978, ,1920
Dreibannstein,1998,X,2010
Labyrinth,2013, ,2140
CO2-Speicher,2022, ,2330
Gabelhaus,2023, ,2400
```

5 Quellcode

In diesem Kontext wird die C++20-Version verwendet, um das Verhalten von Vergleichsoperatoren für eigene Strukturen zu definieren.

```
struct RoutePunkt {
    string name;
    int year;
    bool essential;
    int acc_distance;
};

struct TeilRoute {
    int start_index;
    int end_index;
    int distance;

    bool operator<(const TeilRoute&  other) {
        return other.distance > distance;
    }
};

RoutePunkt parse_line(string line) {
    vector<string> values;
    stringstream ss(line);
    for (int i = 0; i < 4; i++) {
        string substr;
        getline(ss, substr, ',');
        values.push_back(substr);
    }

    RoutePunkt result;
    result.name = values[0];
    result.year = stoi(values[1]);
    result.essential = (values[2] == "X");
    result.acc_distance = stoi(values[3]);

    return result;
}

bool is_valid_teilroute(vector<RoutePunkt> route, TeilRoute tr) {
    for (int i = tr.start_index + 1; i < tr.end_index; i++) {
        RoutePunkt punkt = route[i];
        if (punkt.essential) return false;
    }
    return true;
}

vector<TeilRoute> find_teilrouten(vector<RoutePunkt> route) {
    unordered_map<string, vector<int>> hm_punkte;
    for (int i = 0; i < route.size(); i++) {
        hm_punkte[route[i].name].push_back(i);
    }

    vector<TeilRoute> result;
    for (int i = 0; i < route.size(); i++) {
        vector<int> gleiche_punkte = hm_punkte[route[i].name];
        if (gleiche_punkte.size() < 2) continue;
        for (int j = 0; j < gleiche_punkte.size() - 1; j++) {
            int start_index = gleiche_punkte[j];
            int end_index = gleiche_punkte[j + 1];
            if (start_index < i) continue;
            RoutePunkt start = route[start_index];
            RoutePunkt end = route[end_index];
            int distance = end.acc_distance - start.acc_distance;
            TeilRoute teilroute = { start_index, end_index, distance };
            if (is_valid_teilroute(route, teilroute)) result.push_back(teilroute);
        }
    }
    return result;
}

vector<TeilRoute> filter_best_teilroutes(vector<RoutePunkt> route, vector<TeilRoute> trs) {
```

```
    if (trs.size() < 2) return trs;
    vector<TeilRoute> best;
    for (int i = 0; i < trs.size(); i++) {
        if (trs[i + 1].start_index <= trs[i].end_index) {
            best.push_back(trs[i + 1] < trs[i] ? trs[i] : trs[i + 1]);
            i++;
            continue;
        }
        best.push_back(trs[i]);
    }
    return best;
}

void print_best(vector<RoutePunkt> &route, vector<TeilRoute> &trs) {
    int i = 0;
    while (i < route.size()) {
        auto it = find_if(trs.begin(), trs.end(), [&i] (const TeilRoute& tr) {
            return i > tr.start_index && i < tr.end_index;
        });

        if (it != trs.end()) {
            for (int j = it->end_index; j < route.size(); j++) {
                route[j].acc_distance -= it->distance;
            }
            i = it->end_index;
        } else {
            cout << route[i].name << "," << route[i].year << "," << (route[i].essential ? 'X' : '␣') <<
            i++;
        }
    }
}

int main() {
    vector<TeilRoute> trs = find_teilrouten(initial_route);
    vector<TeilRoute> best_trs = filter_best_teilroutes(initial_route, trs);
    print_best(initial_route, best_trs);
  return 0;
}
```