Aufgabe 5: Stadtführung

Team-ID: 00779
Team: tiantianquan
Autor: Florian Werth
15. November 2023

Inhaltsverzeichnis

Anmerkungen zur Aufgabenstellung:

Im Aufgabentext steht, dass der Start beliebig sein kann. Das habe ich ignoriert, weil man **den Start nicht wegkürzen kann**, den es gibt keine Teiltour in der sich der Start befinden kann. Teiltouren sind für mich chronologisch, da die ursprüngliche Tour auch chronologisch ist, daher müsste eine Teiltour, die wiederum eigentlich auch eine Tour ist, auch chronologisch sein.

Lösungsidee

- finden aller **Teiltouren (start, ende distanz)**, die keinen essenziellen Tourpunkt enthalten
- eine Auswahl an Teiltouren finden, sodass die summierte Distanz größtmöglich ist und keine der Teiltouren sich überschneiden:
  - naiver Ansatz: alle möglichen Kombinationen (2^n) ausprobieren
  - gute Ansatz: Problem lässt sich als weighted activity selection erkennen. Google nach der Lösung fragen. Keinen blassen Schimmer haben. Die Lösung ist zu schwer. Ein Link zu StackOverflow muss her.
    https://stackoverflow.com/questions/39693094/weighted-activity-selection

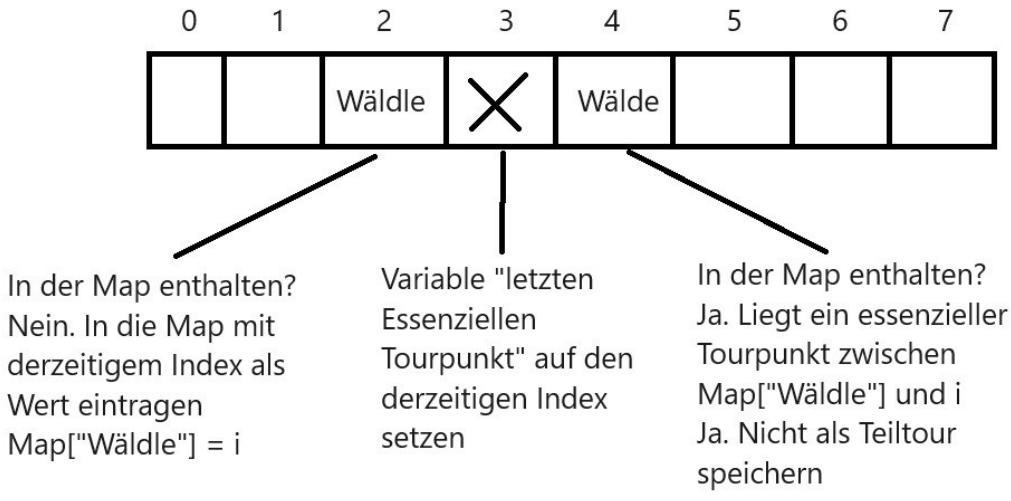
Umsetzung

Die Lösung wurde in C++ geschrieben.

Finden der Teiltouren

1. Die Zeilen werden eingelesen, aufgespalten und der **Tourpunkt als struct mit Name, Jahr, Essenziel und Kulminiert** in einem Vektor gespeichert
2. Beim Einlesen wird gleichzeitig für jeden Eintrag mit einer **unordered_map (Key: Name, Value: letzter Auftrittsort)** überprüft, ob der Tourpunkt schon einmal auftauchte. Liegt zwischen dem letzten Auftreten und dem jetzigen kein essenzieler Tourpunkt, werden die beiden als Teiltour in einem Vektor mit (Anfang = letztes Auftreten (index), Ende = derzeitiger Index, Distanz = die Differenz der beiden kulminierten Distanzen) gespeichert. Dadurch liegen im Endeffekt alle Teiltouren in einem Vektor nach aufsteigendem Ende sortiert.

Veranschaulichung des Algorithmus zum Finden von Teiltouren

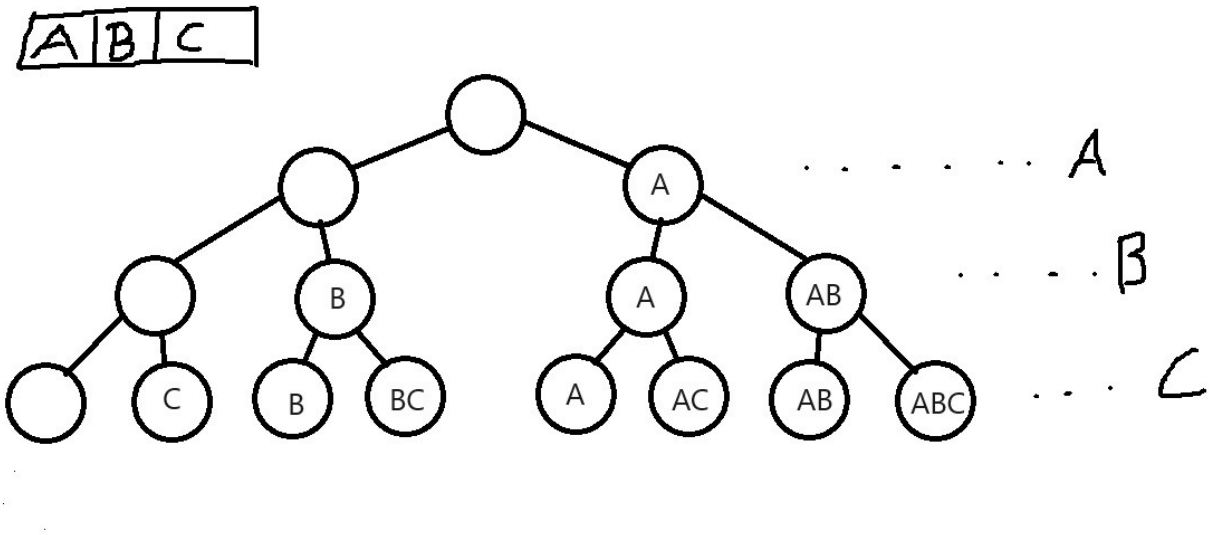


Auswahl treffen (Bruteforce, dabei wird es auch bleiben)

Mit einer rekursiven Funktion werden alle Kombination ermittelt. Der Funktion werden die Teiltouren, ein Vektor in dem die Kombinationen gespeichert werden, die derzeitige Kombination, sowie ein Index, der sich auf den Teiltouren-Vektor bezieht übergeben. Für jeden Funktionsaufruf wird die Funktion wiederum zweimal aufgerufen. Einmal wird nur der Index erhöht und das andere Mal wird der Index erhöht und das aktuelle Element der derzeitigen Kombination hinzugefügt. Ist der Index gleich der Länge der Teiltouren wird die derzeitige Kombination gespeichert.

In der folgenden Grafik ist der Algorithmus veranschaulicht, dabei ist der linke Teilbaum der Funktionsaufruf bei dem die derzeitige Teiltour nicht genommen wird und bei dem rechten Teilbaum wird die Teiltour in die derzeitige Kombination aufgenommen. Die Blätter sind die

Ergebnisse bzw. Kombinationen.



Man kann auch gleich beim Hinzufügen überprüfen, ob der Start der hinzuzufügenden Teiltour kleiner als das Ende der letzten Teiltour ist, wenn ja ignoriert man diese dann auch.

Die optimale Auswahl ist nun diejenige Kombination, deren Summe an Teiltourdistanzen maximal ist.

Für die Ausgabe werden alle gekürzten Tourpunkte markiert. In einer Schleife werden alle Tourpunkte ausgegeben. Tourpunkte die markiert sind werden in Klammern ausgegeben.

Anmerkung zur Eingabe: Die txt. Dateien werden nur richtig gelesen, wenn die oberste Zeile leer ist und n also in der zweiten Zeile steht.

Beispiele

Die "Grafik" zeigt die gefunden Teilrouten mit Distanz und Initialen. Darunter stehen die gekürzten Routen und noch tiefer gelegen ist die endgültige Route. Tourpunkte in Klammern werden nur kurz erwähnt und nicht besucht.

tour3.txt

```
1180  W  W
1290     O  O
600         P  P
410            Z Z
```

(Observatorium, Panoramasteg)
Aus der Grafik wird ersichtlich, dass das wahrscheinlich die optimale Lösung ist.

Talstation 1768
Wäldle 1805
Mittlere Alp 1823
Observatorium 1833
(Wäldle 1841)
(Bergstation 1866)
Observatorium 1874
Piz Spitz 1898
Panoramasteg 1912
(Bergstation 1928)
(Ziegenbrücke 1935)
Panoramasteg 1952
Ziegenbrücke 1979
Talstation 2005

tour1.txt

(Rathaus, Emmy-Noether-Campus)

Brauerei 1613
Karzer 1665
Rathaus 1678
(Gründungsstein 1685)
(Wallanlage 1690)

Rathaus 1739
Euler-Brücke 1768
Fibonacci-Gaststätte 1820
Schiefes Haus 1823
Theater 1880
Emmy-Noether-Campus 1912
(Fibonacci-Gaststätte 1923)
(Hilbert-Raum 1945)
(Schiefes Haus 1950)
(Gauß-Turm 1952)
Emmy-Noether-Campus 1998
Euler-Brücke 1999
Brauerei 2012

tour2.txt

(Rathaus, Emmy-Noether-Campus)

Das Ergebnis bleibt im Vergleich zum vorherigen Beispiel gleich. In der Aufgabenstellung steht zwar, dass der Anfangspunkt nicht der Ursprüngliche sein muss, doch ich habe mich entschieden, dass zu ignorieren, da man den Anfang meiner Auffassung nach nicht weg kürzen kann, weil er sich nicht innerhalb einer geschlossenen Teiltour befindet.

tour4.txt

(Marktplatz, Bogenschütze, Große Gabel)

Blaues Pferd 1523
Alte Mühle 1544
Marktplatz 1549
(Zeughaus 1559)
Marktplatz 1562
Springbrunnen 1571
Dom 1596
Bogenschütze 1610
(Marktplatz 1622)
(Ruhiges Eck 1625)
(Frauentor 1636)
(Ruhiges Eck 1651)
(Springbrunnen 1675)
Bogenschütze 1683
Schnecke 1698
Fischweiher 1710
Reiterhof 1728
Schnecke 1742
Schmiede 1765
Große Gabel 1794
(Schnecke 1829)
(Europapark 1852)
Große Gabel 1874
Fingerhut 1917
Stadion 1934
Marktplatz 1962
Baumschule 1974
Polizeipräsidium 1991
Blaues Pferd 2004

tour5.txt

(Gabelhaus, Dreibannstein, Märchenwald, Eselsbrücke, Riesenrad, Stellwerk)

Gabelhaus 1638
(Burgruine 1654)
(Labyrinth 1667)
(Hängepartie 1672)
(Hexentanzplatz 1681)
Gabelhaus 1699
Hexentanzplatz 1703
Eselsbrücke 1711
Dreibannstein 1724
(Alte Wache 1733)
(Palisadenhaus 1740)
Dreibannstein 1752
Schmetterling 1760
Dreibannstein 1781
Märchenwald 1793
(Fuchsbau 1811)
(Torfmoor 1817)
(Gartenschau 1825)
Märchenwald 1840
Eselsbrücke 1855
(Heimatmuseum 1863)
Eselsbrücke 1877
Reiterdenkmal 1880
Riesenrad 1881
(Hochsitz 1885)
(Gartenschau 1898)
Riesenrad 1902
Dreibannstein 1911
Olympisches Dorf 1924
Haus der Zukunft 1927
Stellwerk 1931
(Dreibannstein 1938)
Stellwerk 1942
Labyrinth 1955
Gauklerstadl 1961
Planetarium 1971
Känguruhfarm 1976
Balzplatz 1978
Dreibannstein 1998
Labyrinth 2013

CO2-Speicher 2022
Gabelhaus 2023

Quellcode (Auschnitte)

```
struct tourpoint
{
	std::string name;
	int year;
	bool essential;
	int culminated;
};
`
`
struct tour
{
	int start;
	int end;
	int distance;
};

// Rekursive Funktion
void generate_combinations(const std::vector<tour>& tours, std::vector<std::vector<tour>>&
combinations, std::vector<tour> curr, int i)
{
	if (i >= tours.size())
	{
		if (curr.size() > 0)
		{
			combinations.push_back(curr);
		}
		return;
	}

	// leave it
	generate_combinations(tours, combinations, curr, i + 1);



	//take it
	if (curr.size() > 0)
	{
		// no intersection
		if (tours[i].start >= curr[curr.size()-1].end)
		{
			curr.push_back(tours[i]);
		}
	}
	else
	{
		curr.push_back(tours[i]);
```

```
        }
        generate_combinations(tours, combinations, curr, i + 1);
}

// Einlesen
std::unordered_map<std::string, int> tourMap;
std::vector<tour> tours;

int last_essential = -1;
for (int i = 0; i < n; i++)
{
        // Aufspalten
        std::getline(example, line);
        std::vector<std::string> split;
        std::stringstream ss(line);
        while (ss.good()) {
                std::string substr;
                std::getline(ss, substr, ',');
                split.push_back(substr);
        }

        tourpoints[i] = { split[0], std::stoi(split[1]), split[2] == "X", std::stoi(split[3]) };

        // Teiltouren finden
        if (!tourMap.contains(tourpoints[i].name))
        {
                tourMap[tourpoints[i].name] = i;
        }
        else
        {
                int start = tourMap[tourpoints[i].name];
                int end = i;
                tourMap[tourpoints[i].name] = end;
                if (last_essential <= start || last_essential > end)
                {
                        tours.push_back({ start, end, tourpoints[end].culminated -
                        tourpoints[start].culminated });
                }
        }

        if (tourpoints[i].essential)
        {
                last_essential = i;
        }
}

// Beste Kombination finden

std::vector<std::vector<tour>> combinations;
std::vector<tour> curr{};
generate_combinations(tours, combinations, curr, 0);
```

```
int max_index = 0;
int max = 0;
for (int c = 0; c < combinations.size(); c++)
{
	int curr{};
	for (auto x : combinations[c])
	{
		curr += x.distance;
	}
	if (curr > max)
	{
		max = curr;
		max_index = c;
	}
}
```