# Aufgabe 4: Urlaubsfahrt

Team-ID: 00485

Team-Name: FXC3

Bearbeiter/-innen dieser Aufgabe:
Leonardo Benini

10. November 2019

## Inhaltsverzeichnis



## Lösungsidee

Im Kern der Lösung liegt ein Brute Force Attacke, jedoch gibt es einige Optimierungen. Zuerst findet das Programm die minimale Stopp-Anzahl heraus. Daraufhin wird eine rekursive Methode in Gang gesetzt, die alle Möglichkeiten durchprobiert. Diese kürzt ab, wenn die noch zu fahrende Distanz geteilt durch die maximale Reichweite des Autos mit einer Tankfüllung größer ist als die minimale Anzahl an Stopps minus die schon gemachten Stopps.

Sobald eine Route mit der minimalen Stopp-Anzahl gefunden worden ist, wird diese Route durch eine Methode geführt, die die Menge an Benzin optimiert, die getankt werden soll, sodass möglichst geringe Kosten entstehen. Dies wird wie folgt gemacht: Wenn Tankstelle 1 billiger ist als Tankstelle 2, wird der Tank vollgemacht. Wenn dies anders herum ist, wird der Tank genauso gefüllt, sodass das Auto zur nächsten Tankstelle gelangen kann. Der Preis der neuen Route wird dann mit dem Preis der bis jetzt besten Route verglichen und falls sie billiger ist, wird sie die neue beste Route.

Am Schluss bleibt die billigste Route übrig und wird dann zurückgegeben.

## Umsetzung

Die Lösungsidee wird im Python implementiert. Das Modul copy erlaubt es einem in Python eine Kopie von einem Array zu erstellen, der in sich noch weitere Arrays trägt.

Zuerst wird die Methode „find_best_route“ aufgerufen, die zuerst die Methode „find_min_amount_of_stopping“ aufruft.

```
def find_min_amount_of_stopping(cons, gas_stations, fill, distance): ## finds out the minimum amount of stops you have to do
    global stations, route, tsize
    stops = 0
    while check_destination_in_range(cons, fill, distance) == False:
        furthest_station = copy.deepcopy(stations_in_range(cons,fill, gas_stations)[-1])
        amount_of_fuel = tsize - (fill - calc_fuel(cons, furthest_station[0]))
        stops += 1
        distance, fill, gas_stations = fuel(furthest_station, amount_of_fuel, fill, distance, gas_stations)
        route.append([amount_of_fuel, stations[len(stations) - len(gas_stations) - 1]])
    return stops
```

In dieser Methode wird eine while Schleife solange ausgeführt bis das Ziel ohne einen weiteren Stopp erreichbar ist. In dieser Schleife werden mit Hilfe einer anderen Methode die erreichbaren Tankstellen errechnet und die entfernteste Tankstelle wird dann als nächstes Ziel angesteuert. Außerdem wird der Tank immer vollgemacht, um zu garantieren, dass so wenige Stopps wie möglich gemacht werden müssen. Die Methode „fuel“ wird immer dann aufgerufen, wenn das Auto tankt und es passt eigentlich nur alle Werte den neuen Umständen an.

```
def loop(gas_stations, r, fill, cons, tsize, distance, min_stops, b_fill, b_dis): ## brute force function
    c1 = copy.deepcopy(gas_stations)
    c2, c3, c4 = (r, fill, distance)
    global route, stations, beste_route
    for i in range(len(stations_in_range(cons,fill, gas_stations))):
        station = copy.deepcopy(gas_stations[i])
        route[r][1] = stations[len(stations) - len(gas_stations) + i]
        fuel_amount = tsize - (fill - station[0] / 100 * cons)
        route[r][0] = fuel_amount
        distance, fill, gas_stations = fuel(station, fuel_amount, fill, distance, gas_stations)
        if check_destination_in_range(cons, fill, distance) == True:
            new_route = copy.deepcopy(opt_money_spent(copy.deepcopy(route), cons, b_dis, b_fill))
            if beste_route == []:
                beste_route = copy.deepcopy(new_route)
            elif new_route[-1] < beste_route[-1]:
                beste_route = copy.deepcopy(new_route)
            gas_stations = copy.deepcopy(c1)
            r, fill, distance = (c2, c3, c4)
            continue
```

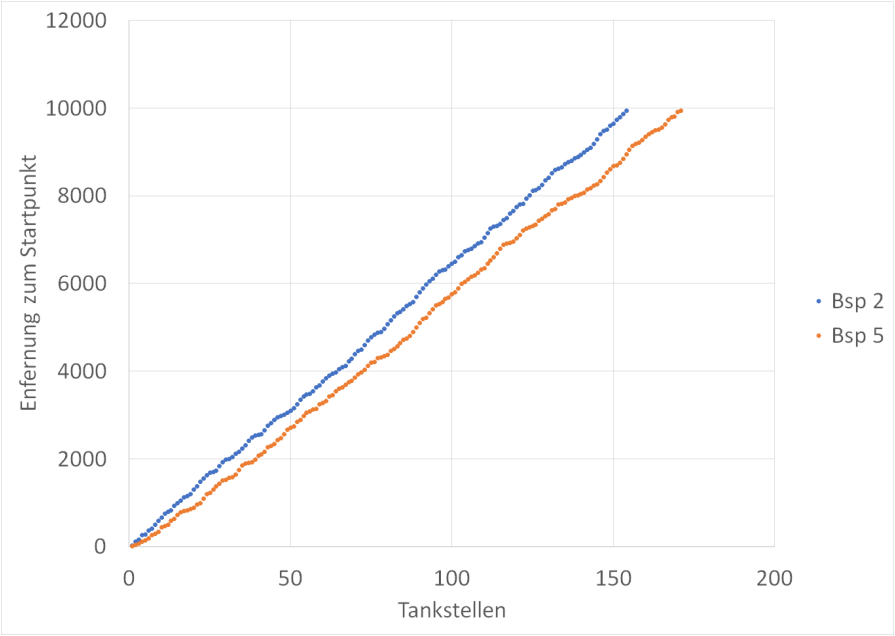
Sobald die minimale Anzahl an Stopps herausgefunden ist, wird die rekursive Methode „loop“ aufgerufen. Diese Methode beinhaltet eine for Schleife, die durch die erreichbaren Tankstellen iteriert. Außerdem befinden sich in der for Schleife einige Bedingungen, die zum Beispiel die Route aufgeben, wenn das Ziel gar nicht mehr in den minimalen Stopps erreicht werden kann. Wenn dies aber nicht der Fall ist wird dieselbe Methode nochmals aufgerufen mit den angepassten Werten. Sobald das Ziel ohne einen weiteren Stopp erreichbar ist, wird die Methode „opt_money_spent“ aufgerufen, die die Menge an Geld, das fürs Tanken ausgegeben wird, ans Minimum bringt. Sobald dies geschehen ist, wird die Geldsumme mit der bis jetzt niedrigsten Geldsumme verglichen und falls diese niedriger ist, wird die neue Route die beste Route.

```python
def opt_money_spent(route, cons, dis, fill): ## function that optimizes the money you spent fueling up
    global tsize
    am = float(fill)
    route.append(0.0)
    for i in range(len(route) -1):
        if i == 0:
            am -= calc_fuel(cons, route[i][1][0])
        else:
            am -= calc_fuel(cons, route[i][1][0] - route[i - 1][1][0])
        if i == len(route) -2:
            route[i][0] = calc_fuel(cons, dis - route[i][1][0]) - am
            route[len(route) -1] += route[i][0] * route[i][1][1]
        else:
            if route[i][1][1] < route[i + 1][1][1]:
                #if the first gas station is cheaper, make the tank full
                route[i][0] = tsize - am
                am = tsize
            else:
                route[i][0] = calc_fuel(cons, route[i + 1][1][0] - route[i][1][0]) - am
                am += route[i][0]
            route[len(route) - 1] += route[i][0] * route[i][1][1]
    return route
```

Die Methode „opt_money_spent“ funktioniert, wie in der Lösungsidee beschrieben: Eine for Schleife geht alle Tankstellen durch und tankt voll auf, wenn die jetzige Tankstelle billiger ist als die nächste. Wenn dies andersrum ist tankt er nur so viel auf, sodass das Auto genau zur nächsten Tankstelle kommt.

Nach all diesen Schritten bleibt die beste Route übrig und wird ausgegeben.

Die Laufzeit des Programms ist abhängig von der Anzahl der Tankstellen und braucht in den Beispielen 2 und 5 ziemlich lange. Jedoch benötigt Beispiel 2 deutlich länger als Beispiel 5, obwohl die Anzahl der Tankstellen, deren Verteilung (siehe Graph), der Verbrauch und die Tankgröße vergleichbar sind.




Lediglich die Menge an Benzin, die anfangs im Auto ist, ist unterschiedlich. Eine Lösungsidee für dieses Problem wäre von hinten für jede Tankstelle die beste Route zum Ziel zu berechnen und diese

Berechnungen in sich zu verwenden. Durch diese Herangehensweise könnten extrem viele Berechnungen erspart werden.

## Beispiele

Die Ausgabe der Tankstellen funktioniert wie folgt: In einer Liste werden alle Tankstopps aufgeführt. Jeder Tankstopp ist besteht aus zwei Einträgen: der zu tankenden Menge und der entsprechenden Tankstelle, identifiziert durch die Entfernung zum Startpunkt und dem Spritpreis.

Beispiel 1:

```
Anzahl der durchgeführten Tankstopps: 2

Preis für die ganze Reise: 81.70€

[[10.0, [100.0, 1.45]], [48.0, [400.0, 1.4]]]
```

Beispiel 2:

```
Anzahl der durchgeführten Tankstopps: 9

Preis für die ganze Reise: 2530.33€

[[269.32, [1118.0, 1.15]], [192.95999999999998, [1922.0, 1.17]],
[184.63999999999996, [2762.0, 1.19]], [274.0, [3833.0, 1.18]], [197.12,
[4874.0, 1.19]], [274.0, [5796.0, 1.17]], [237.91999999999996, [6912.0,
1.31]], [274.0, [7929.0, 1.15]], [223.04, [8987.0, 1.21]]]
```

Beispiel 3:

```
Anzahl der durchgeführten Tankstopps: 1

Preis für die ganze Reise: 99.20€

[[80.0, [465.0, 1.24]]]
```

Beispiel 4:

```
Anzahl der durchgeführten Tankstopps: 2

Preis für die ganze Reise: 236.88€

[[88.2, [264.0, 1.2]], [100.79999999999998, [607.0, 1.3]]]
```

Beispiel 5:

```
Anzahl der durchgeführten Tankstopps: 9

Preis für die ganze Reise: 2494.32€

[[209.57999999999998, [107.0, 1.16]], [244.0, [1198.0, 1.15]], [229.76,
[2344.0, 1.17]], [244.0, [3454.0, 1.16]], [234.14999999999998, [4569.0,
1.17]], [202.01999999999998, [5531.0, 1.31]], [232.07, [6692.0, 1.34]],
[244.0, [7798.0, 1.16]], [218.42, [8944.0, 1.31]]]
```

## Quellcode

Das Projekt ist auf Github einsehbar.

Link: https://github.com/FloxiGH/BWINF2019