**923403630780**

**Algorithms and data structures**

preface

This document describes a number of user stories. It is up to you to sort these user stories

to deliver.

Background and reason

The company “BittyBite” (from now on shortened to “BB”) is an internet service provider. "BB" focuses

especially on the 50+ market within Amsterdam. This target group states, in the case of installation and / or

problems, a personal approach is more appreciated than contact by phone and / or email.

For that reason "BB" employs a number of customer-friendly technicians. Every “BB” technician has

the task of visiting customers every day who have problems with their internet connection.

Furthermore, “BB” puts (possible) profit mainly in improving and making hair more sustainable

services. For example, “BB” aims, among other things, to only allow technicians to use within the city

making public transport and specifically metro and tram lines (so bus lines are out of scope; after all

pollutant).

The “BB” planner schedules the work for the technicians one day in advance based on

"First come, first served". However, it often appears on the day itself that there are appointments

fall out, be moved and / or recover. The planner must then switch on again during the day

the battle to distribute the remaining appointments as well as possible among those who are active on that day

mechanics.

Scope and objective

The disadvantage of this method of planning is that a technician does not always follow the most convenient route

one day. This takes time that the technician could have used for another customer

to assist.

Management knows about the existence of AI-like techniques at other companies in the world

where a planner is no longer necessary at all. But they are skeptical. There will also be in the near

future a CRM will be purchased to keep track of appointments. In that sense, your solution is one

extension of a yet to be chosen CRM system where customer data and agreements are made

kept up. So it is necessary that the planning system can handle these agreements but

is not the source of these agreements.

For that reason, the management of “BB” asks you to show a number of “techniques”

showing that it is possible to help a planner; not so much to replace.

Your solution therefore offers a number of functions which will soon be used by another system

is called (sort of black-box approach) and is able to extract appointments from the CRM system

to come up with the best schedule for a mechanic.

Approach and expected results

“BB” has expressed its wishes in a number of user stories; including the corresponding

acceptance criteria. In addition, the “AlgoDS Final Assignment Manual” mentions general criteria /

guidelines, which apply to all user stories.

“BB” expects you to implement a minimum number of user stories.

So “BB” does not, at this stage, appreciate a user interface but more a (black box)

“System” that provides an interface that makes it possible to call various functions from

outside.

User stories

BB wants as “pure” as possible an implementation of the system in order not to depend (in the future)

to be of all kinds of modules which (possibly) entail all kinds of costs and / or deployment issues

bring over. Therefore (this is a precondition) “BB” imposes the restriction that you do NOT use

may make (external) modules / packages. This also makes it possible to (in the future) use the

transfer code to a programming language other than Python (should “BB” decide to do so).

You must therefore also adhere to the guidelines (see the document “AlgoDS Final Assignment

instruction manual")

**US01 Reading GVB data from tram line 1 (proof of concept)**

As a planner I want to be able to use data from the Amsterdam Municipal Transport Company

(GVB) so that I always have the most current metro / tram line information.

Background

Initially, “BB” wants to see that you are technically able to read information from the GVB /

processing. In total there are X tram and metro lines. The tram lines start at 1 and the metro lines

starting at 50. As an example, the GVB has supplied a JSON file. This is data related to tram line

1. See the file “GVB\_1\_1.json” (the structure of this file is explained in “AlgoDS

Final assignment manual ”).

Note: For the sake of convenience, we will now use the term “line” to refer to a tram or

metro line to indicate!

Acceptance criteria

The ads\_numberOfStationsInTrack () function

• is equal to the number of stations on line 1;

• output type: integer

The ads\_nameOfStartStationInTrack () function:

• is equal to the name of the 1st

station of line 1;

• output type: string

The ads\_nameOfEndStationInTrack () function:

• is the same as the name of the terminal of line 1;

• output type: string

The ads\_listOfStationsInTrack () function:

• output type: list or string;

o is the same as the names of the stations of line 1;

o sequence from start to end station

Method / tips

This user story is an extension of the previous one. The functions created in US01 will continue to work

almost the same, except that the number of a line can be assigned to the functions. You can

1 Crash Course Programming in Python (Publisher: Visual Steps)

using the datatype “Dictionary” create a kind of look-up table to, via a line number, the name of

retrieve a JSON file.

**US02 Reading GVB data regarding all metro and tram lines**

As a planner I want to be able to use all tram and metro data from the GVB so that I always

have the most up-to-date line information.

Background

See the other list of JSON files.

Acceptance criteria

The ads\_numberOfStationsInTrack (<track>) function

• input type <track>: integer (the number of the line)

• output type: integer

o equals the number of stations within the line;

o equals “-1” if the line does not exist

The ads\_nameOfStartStationInTrack (<track>) function:

• input type <track>: integer (just number of the line)

• output type: string

o equals the name of the 1st

station of the line;

o equals “(unknown)” if the line does not exist

The ads\_nameOfEndStationInTrack (<track>) function:

• input type <track>: integer (the number of the line)

• output type: string

o is the same as the name of the terminal of the line;

o equals “(unknown)” if the line does not exist

The ads\_listOfStationsInTrack (<track>) function:

• input type <track>: integer (the number of the line)

• output type: list or string;

o equals the names of the stations within the line;

o equals “(unknown)” if the line does not exist;

Method / tips

This user story is an extension of the previous one. The functions created in US01 will continue to work

almost the same, except that the number of a line can be assigned to the functions. You can

1 Crash Course Programming in Python (Publisher: Visual Steps)

using the datatype “Dictionary” create a kind of look-up table to, via a line number, the name of

retrieve a JSON file.

**US03 Search line (s) based on station name**

As a planner I want to know which lines a technician can use to get to a particular station

so that I can make more efficient planning.

Background

A planner within “BB” deals with the planning of a number of technicians. As a customer

indicates that you live at a certain station in the area, the planner can use this function to

to pick up the line (s) passing that station. Certain lines are already in the day

near a mechanic's work area. Then it is more convenient to have the same technician do that

do it and not another.

Acceptance criteria

The ads\_listOfTracksGivenStation (<station>) function:

• input type <station>: string (name of the station)

• output type: list of integers;

o equals the numbers of the lines serving the station;

o equals an empty list if the drive does not exist

o order of numbers is NOT important.

The ads\_isStationOnTrack (<track>, <station>) function:

• input type <track>: integer (number of the line)

• input type <station>: string (name of the station)

• output type: boolean (true / false);

o is TRUE if station is part of the line; otherwise FALSE

NOTE: it is not allowed to use standard functions from Phyton to search. With

in other words, it must be made yourself. Think of the search algorithms which

during the lecture.

Method / tips

You are already able to read the GVB information. The same drive can contain multiple files

appearance. So it's time to find a way to keep track of which stations per station

files it prevents. Suppose the “Central Station” occurs in lines 1, 4, 13 and 50, then

(and this is one way) you create some sort of list for that station containing the numbers of the lines.

The outcome could be:

• “Central Station” → 1, 4, 13, 50

• “Ferdinand Bolstraat” → 1, 5

•…

So if you go through all the files you can update the list of songs per station.

Note: In this example, line 1 will pass both “Ferdinand Bolstraat” and “Central”

Station ”

Now you (probably) have a list of stations with, per station, a list of numbers. The number of

stations is not very large, so to search for a station, you can suffice with a simple search algorithm.

**US04 Processing travel time information (proof of concept)**

As a planner, I want to have access to travel times (between stations) so that I can make better promises

towards customers.

Background

The JSON files supplied by GVB do not only contain information about the structure of

the metro / tram network, but also the travel times between stations. As a prelude to the automatic

plans, “BB” initially wants to see that you are technically able to calculate travel times between stations

processing.

As a planner, this helps me, in the first instance, to determine how long it takes a technician

get from station A to station B within a specific line.

Note: you do NOT need to use the current information from the GVB files yet,

but you can suffice with average values ​​(see section “Starting points…” at the bottom of this

document).

Acceptance criteria

The ads\_travelTimeBetweenStationsGivenTrack (<track>, <station\_departure>,

<station\_arrival>):

• input type <track>: integer (the number of the line)

• input type <station\_departure>: string (name of the departure station)

• input type <station\_arrival>: string (name of the arrival station)

• output type: integer

o equals the travel time in minutes from departure to final station

o equals the value -1 if the end station from the departure station is not in 1 step / stop

accessible

**Method** / **tips**

So far you have created different data structures to make up the previous user stories.

Perhaps this user story is for you a small extension of what you already have. But let's be there

assume that this is not the case yet.

In this user story you work on a data structure in which you will keep track of which for each station X.

other stations you can reach from station X AND how long a tram / metro takes.

The outcome could be:

**• “Frederiksplein” → list of adjacent stations:**

**o “Johan Huizingalaan”, line 1, z minutes;**

**o “Central Station”, line 5, y minutes;**

**• “Central Station” → list of adjacent stations:**

**o “Ferdinand Bolstraat”, line 7, w minutes;**

**o “Frederiksplein”, line 5, q minutes;**

**o “Ferdinand Bolstraat”, line 6, hours**

**•…**

Note that the travel time of 2 adjacent stations A and B can depend on the direction (also

suggests the average time not; because it is the same in all directions!). So from station A.

for example, to station B can be 3 minutes but 2 minutes in the opposite direction! In the

above example q may be different from y!

Also pay attention to the fact that (see the example) you can get to the

“Ferdinand Bolstraat” can travel; namely via line 7 and via line 6. Although for this user story the

travel times are the same, they may differ later!

**US05 Process current travel time information**

As a planner I want to have the most accurate travel times (between stations) so that I can schedule schedules

of mechanics can adjust if necessary.

Background

The current set of files contain information about the planned travel time between stations. But

due to work and / or accidents, this information may no longer be correct. “BB” goes

therefore purchase a subscription from the GVB to notify “BB” if the GVB provides

that (planned) travel times are no longer met. This information can therefore be obtained at any time of the

day in.

For now “BB” wants you to demonstrate that you can take this into account by creating a function

with which the travel time between 2 adjacent stations within the same line can be adjusted

AFTER the normal GVB information has been read.

Acceptance criteria

The ads\_changeTravelTimeBetweenStationsGivenTrack (<track>, <station\_departure>,

<station\_arrival>, <travel\_time>):

• input type <track>: integer (the number of the line)

• input type <station\_departure>: string (name of the departure station)

• input type <station\_arrival>: string (name of the arrival station)

• input type <travel\_time>: integer (travel time in minutes)

• output type: integer

o equals the travel time in minutes from departure to final station before the adaptation

o equals the value -1 if the end station from the departure station is not in 1 step / stop

reachable

Note: The travel time can also be a high value (e.g. 1000)! This could be later

used to indicate that the travel time between these 2 stations is so long that there is actually none

“Normal” connection is possible.

Method / tips

You are already able to record travel times between 2 adjacent stations. So with this user

story you just need to reset it. You give back the “old” travel time as output; unless

the 2 stations are not adjacent (then you return the value -1).

**US06 Check for delay**

As a planner I want to know whether a technician can NOT be at his / her next appointment in time so that I can

can arrange alternative transport for the technician and inform the customer about the delay.

Background

After the service desk has recorded an appointment in the CRM system, the planner already has one

technician selected. After updating the travel times information, it may be that a certain

connection between adjacent stations A and B is no longer possible and / or will take too long. So

the system must offer a function that checks whether a technician cannot make an appointment

fulfill. In that case, the planner arranges an alternative means of transport and the customer is responsible for the

mechanic a little later.

For clarification… The planner uses the function by specifying the departure and end station and

also the maximum acceptable travel time by the planner between 2 adjacent stations. Like a

the connection between 2 adjacent stations exceeds this “threshold”, the planner sees this as

an unacceptable connection between 2 stations.

The latter is easy to imagine ... Suppose the travel time between 2 adjacent stations

“Central Station” to “Ferdinand Bolstraat” takes 35 minutes via line 7, and the “treshold” at 30

minutes, the system should consider this connection as “does not exist”; in fact ignore

that this is possible. Because which technician will sit on a tram or metro for longer than 30 minutes at 1

make a stop?

Acceptance criteria

The function ads\_shortestRouteBetweenStations (<station\_departure>, <station\_arrival>,

<treshold\_traveltime>)

• input type <station\_departure>: string (name of the departure station)

• input type <station\_arrival>: string (name of the arrival station)

• input type <treshold\_traveltime>: integer:

o> = 0; default = 10

o number of minutes traveling time above which a connection between 2 adjacent stations

within the same line is seen as temporarily unreachable / blocked

• output type: list or string

o equals a list of (the least number of) stages (see below) which are the route

Form the mechanic to come from the departure station

(<station\_departure>) to arrival station (<station\_arrival>)

o a stage is described by “line:“ + <line number> + “# boarding:“ +

<departure station> + “# alight:“ + <arrival station> + “# number of stops:

“+ Number of stops from boarding to getting off

o is empty if <station\_departure> equals <station\_arrival>

o equals “unreachable” if <station\_arrival> cannot be reached from

<station\_departure> (taking into account <treshold\_traveltime>)

o equals “error” if either drive does not exist / does not exist for the system

is known

Note: The departure and end station can be on different lines!

The ads\_isEngineerDelayed function (<station\_departure>, <station\_arrival>, <treshold\_traveltime>)

• input type <station\_departure>: string (name of the departure station)

• input type <station\_arrival>: string (name of the arrival station)

• input type <treshold\_traveltime>: integer:

o> = 0; default = 10

o number of minutes traveling time above which a connection between 2 adjacent stations

within the same line is seen as temporarily unreachable / blocked

• output type: boolean

o equals TRUE if the technician is delayed from time to time; different

FALSE

o There is a delay if, taking into account the travel time information between stations, the technician has to make more stops than if the system

ignores this travel time information

Method / **tips**

You can divide this user story into 3 steps.

First of all, you implement a shortest path algorithm (BFS or DFS). You then ignore the

travel times between 2 adjacent stations; so all connections are possible regardless of the may

high travel time.

Then you adjust that algorithm so that a connection between 2 adjacent stations A and B “does not

exists / is possible ”if the travel time is“ too long ”. In other words if the travel time is between the 2

adjacent stations come out above the “treshold”.

Finally, you use this algorithm to make function “isEngineerDelayed”. Suppose you act like you

does not take travel times / delays into account at all (threshold is 0), then the algorithm delivers

you on a specific route. By using the same function “ads\_shortestRouteBetweenStations” again

to use, but with the entry “threshold” it may turn out that a different route comes out.

Then you can answer the question whether there is a delay.

NOTE: The shortest route calculates the route from A to B where the number of stops the technician has

passes / visits is as small as possible. You are NOT looking for the fastest route yet; so you do

not yet take into account the total duration of a journey and / or transfer times from tram to

metro (or vice versa). It is therefore possible that the shortest route is a combination of a mix of trams / or metro lines.

***US07 Find fastest route mechanic***

As a planner I want to determine the fastest route so that there may be room for an extra later in the day

appointment.

Background

As a planner, I still initially determine the order of appointments on my own

technician and I estimate the travel times and transfer times myself. As a planner, I would like the system

want to know the fastest route per appointment; from station A to B (the system does not have to

NOT yet to take transfer times into account). There may still be some at the end of the day

there is room for an extra appointment (if another customer calls with an urgent job).

Acceptance criteria

Function ads\_fastestRouteBetweenStations (<station\_departure>, <station\_arrival>)

• type <station\_departure>: string (name of the departure station)

• type <station\_arrival>: string (name of the arrival station)

• output type: list or string

o equals a list of legs (see below) which are the fastest route for the

technician to come from the departure station (<station\_departure>) to

arrival station (<station\_arrival>)

o a stage is described by “line:“ + <line number> + “# boarding:“ +

<departure station> + “# alight:“ + <arrival station> + “# number of stops:

“+ Number of stops from boarding to getting off +“ # travel time: “+ total time that

necessary to complete the stage

o is empty if <station\_departure> equals <station\_arrival>

o equals “unreachable” if <station\_arrival> cannot be reached from

<station\_departure>

o equals “error” if either drive does not exist / does not exist for the system

is known

Method / **tips**

By using the Dijkstra algorithm you can determine the fastest route; after all, the travel time information is known. There is no need to take into account a “threshold” (as in the

previous user story) because the Dijkstra algorithm eventually finds the fastest route (if there is one)

taking into account travel times between stations.

NOTE: the fastest route calculates the route from A to B which takes the least time. It could be that

more stops are visited in comparison to the shortest route from A to B. You still have to

not taking into account possible transfer times from tram to metro (or vice versa). So it is possible

are that the fastest route is a combination of a mix of tram and / or metro lines (while if you take the

transfer times would result in a different route).

***US08 Find nearest station***

As a planner, I want to determine the nearest station based on a zip code so that the

technician does not have to walk too far

Background

Service desk always asks customers what their estimated distance is to the nearest station,

but that's not always true. Sometimes customers are used to name a particular station while there

in reality is a station that is closer. So “BB” wants to be based on a customer's zip code

determine the nearest station. For this purpose “BB” has purchased a data set containing

the corresponding “position format” per postal code.

Acceptance criteria

Ads\_nearestStation (<zip\_code>) function

• input type <zip\_code>: string (zip code in format 1111AA)

• output type: string

o equals the name of the station closest to the specified postcode

• output type: integer

o equals the distance (rounded in whole meters) from the postcode to the

found station

Note, this function returns 2 values!

Furthermore, it is NOT allowed to SEARCH by zip code, that is much too slow… so make it

a custom HASH TABLE using a custom HASH FUNCTION.

Method / **tips**

The zip code file contains geographic coordinates; expressed in degrees and minutes (a

degree is 60 minutes) to three decimal places (eg N51 ° 58.127 E005 ° 40.273). In your GPS

this position format is referred to as hddd ° mm.mmm'2

.

The GVB uses the same kind of coordinates to indicate the location of its stations. You should

so create an algorithm that looks up the zip code, gets its coordinates, and then

determine which of the stations is closest. This will create a hash table where the key is

the postcode and the corresponding value is the nearest station.

***US09 Finding fastest route engineer (version 2)***

As a planner I want to determine the fastest route taking into account transfer times so that there are later on the

day there is room for an extra appointment.

Background

This is an extension of existing functionality. As a planner, I still decide at first

the order of appointments with a technician on my own and I estimate the travel times and

transfer times in. As a planner I would like to know from the system what the fastest route of the

one to another appointment where the system now takes transfer times into account.

Acceptance criteria

Function ads\_fastestRouteBetweenStationsV2 (<station\_departure>, <station\_arrival>)

• type <station\_departure>: string (name of the departure station)

• type <station\_arrival>: string (name of the arrival station)

• output type: list or string

o equals a list of legs (see below) which are the fastest route for the

technician to come from the departure station (<station\_departure>) to

arrival station (<station\_arrival>), taking into account transfer times

o a stage is described by OR:

▪ “line:“ + <line number> + “# boarding:“ + <departure station> + “

# alighting: “+ <arrival station> +“ # number of stops: “+ number of stops

from boarding to disembarking + “# travel time:“ + total time required

to complete the stage

▪ “transfer:“ + time in minutes to transfer to the next line

o is empty if <station\_departure> equals <station\_arrival>

o equals “unreachable” if <station\_arrival> cannot be reached from

<station\_departure>

o equals “error” if either drive does not exist / does not exist for the system

is known

Method / **tips**

2 The N and the E indicate that it is a location north of the equator and east of the

prime meridian. The distance between two latitudes is 111 km. The distance between two longitudes varies,

because the meridians towards the poles get closer and closer together. At the equator is the distance between

two longitudes 111 km, in the Netherlands 64 km and 0 km at the poles.

Expansion of user story 07. If you want to go from station A to station B but these are not on

the same line, you have to add the transfer time to the travel time. This may result in a different route.

***US10 Organize appointments for technician***

As a planner, I want the appointments assigned to a technician to be the most efficient

order so that a technician may still have room for an extra appointment

Background

Until now, the planner has determined the order himself. By making functionality earlier, it already is

possible to determine the fastest route (from one appointment to another) based on a

manual classification; taking into account transfer time. The goal now is to change the order of the

appointments to be determined by the system so that the total time is in a certain order of

agreements is the least compared to another combination of agreements.

As a person, the planner also takes into account the fact that:

• a technician leaves from a home address;

• a technician must walk from his / her home address to a departure station;

• a technician has to walk to get to an appointment from a terminal station;

• a technician must walk back to the same station after an appointment has been completed

travel to a next station close to the next appointment;

• a technician needs time for the repair

To test the operation of the system, “BB” provides a list of appointments

of the CRM system. The following information is (in principle) available for each appointment:

• Name and address / zip code of the customer;

• Desired time with regard to an appointment with the technician;

• Reason for the appointment;

• Name of the technician.

Acceptance criteria

The ads\_efficientRouteEngineer (<engineer>) function

• input type <engineer>: string (name of an engineer)

• output type: list or string

o equals a list of stages which is the fastest route for the engineer to

to make all appointments, taking the mechanic's home address into account,

walking, transfer and assembly times

o a stage is described by OR:

▪ “walking from:“ + starting position of the technician (home address, station or appointment) +

“#To:“ + final position of fitter (home address, station or appointment) + “#distance:

“+ Distance +“ # travel time: “+ total time needed to walk the distance

laying OR

▪ “line:“ + <line number> + “# boarding:“ + <departure station> + “

# alighting: “+ <arrival station> +“ # number of stops: “+ number of stops

from boarding to disembarking + “# travel time:“ + total time required

to complete the stage

▪ “transfer:“ + time in minutes to transfer to the next line

▪ "assembly time:" + number of minutes of assembly at an appointment

o is empty if there are no appointments for the technician

• output type: integer

o is equal to the total number of minutes that the technician is on the road

Pay attention to the following:

• See the starting points / assumptions (see end of this document)!

• You do not have to take into account the lunch break or the maximum working time yet!

• The function therefore returns 2 values!

The figure above shows an example with 2 agreements:

• the technician walks from a home address to the nearest station to get to the 1st

appointment (stages A, B and C);

• the technician needs time for the repair (stage D)

• the technician walks back to the station to go to the nearest station of appointment 2

to come and walk there (stages E, F, G)

• the technician needs time for the repair (stage H)

• the technician returns home (stages I, J and K)

Method / **tips**

The functions you have created so far are now easy to use. This is an extension of user

story 09 in which you have already found a way to get from station A to station B

to come; taking into account the transfer time. But more needs to be done now.

First of all, you have a list of appointments. So basically you have to go combinations of appointments

try to find out in which order of appointments a technician spends the least time.

In fact, this resembles the Traveling Travelers problem. And because the number of appointments is small enough, you can

you can use the Traveling Traveler algorithm to examine all combinations; so you still have to

not directly use a GREEDY algorithm.

First of all, the departure station is fixed for all appointments; because this is the station closest to

the mechanic's home address is. To do this, use the mechanic's zip code to enter the

nearest station (see the hash table with postcodes from user story 08). You can then

determine what the 1st stage is; namely walking from the home address to the 1st

departure station (in the

figure stage A).

Then you grab a first appointment. Use the zip code of this appointment (and the hash table with

postcodes from user story 08) to find out what the nearest station X is (stage B) and

how far it is to walk from station X to this appointment (stage C). Then the technician starts the

battle (stage D). Then he / she walks back to station X to go to the next appointment

to go. This convention also has a zip code and a station Y closest to this zip code. This

leads to stages E, F and G). Etc.… A first combination then yields a total time. After this you grab

another appointment as the first appointment and repeat the process as described above.

Afterwards you have looked at all combinations and a minimum time rolls out with 1 combination.

**US11 Organize all appointments**

As a planner, I want the appointments to be assigned to the

available technicians so that all technicians are deployed as well as possible and I do not need to

many mechanics deployed.

Background

By making US10 you have already managed to determine the order of appointments. However, if more

agreements are then there in 1 day passes for 1 technician then that is no longer possible. However, "BB" has

several technicians employed. But “BB” doesn't want too many technicians working every day

put. On the one hand this costs money and on the other hand there are fewer days (due to free time / illness)

technicians available. So a planner wants to estimate or list appointments at a particular

day can all be handled with the number of technicians available for that day.

Acceptance criteria

The ads\_planAppointments (<list of engineers>) function

• Input type <number of engineers>: list of string (the number of engineers available)

• output type: list or string

o equals a list of stages, PER MECHANIC, which, seen throughout the day,

is the fastest route for a technician to come from his / her home address

up to and including the last appointment of that day, taking home address into account,

walking, transfer and assembly times and a break moment

o a stage is described by OR:

▪ “technician:” + technician + “#walk from:“ + starting position technician

(home address, station or appointment) + “# to:“ + final position of fitter (home address,

station or appointment) + “#distance:“ + distance + “# travel time:“ + total time needed

to walk the distance OR

▪ “technician:” + technician + “# line:“ + <line number> + “# boarding:“ +

<departure station> + “# alight:“ + <arrival station> + “#number

stops: “+ number of stops from boarding to getting off +“ # travel time: “

+ total time needed to complete the leg

▪ “technician:” + technician + “#switching:“ + time in minutes to switch

steps to the next line

▪ “technician:” + technician + “#break:“ + number of minutes break

o is empty if there are no appointments on that day

• output type: integer

o equals the total number of appointments that could NOT be scheduled

Note the following:

• See the starting points / assumptions (see end of this document)!

• You do not have to take the lunch break into account, but you DO have a maximum working time;

• The function therefore returns 2 values!

Method / **tips**

You can solve this problem by using a GREEDY algorithm. In fact, you are kind of going

Apply a “clever-sack” algorithm in which you try every mechanic's backpack as well as possible

to fill.

You know you're putting too much in a mechanic's backpack when the maximum work time gets

overwritten. So if a technician leaves home, ends 5 appointments, returns home and

then stay within working hours then it is good. But if it turns out that you do with an extra appointment

exceeds the maximum working time, that combination is not allowed. If you still have agreements about it

If you have one, pick up another technician and do the same. Etc.… Ultimately you have

or no more agreements about it.

Starting points, assumptions and preconditions

You can use the following parameters for the system:

• Mechanic: Average repair time = 35 minutes;

• Mechanic: A mechanic works a maximum of 8 hours a day;

• Mechanic: A mechanic walks at a speed of 5 km / h;

• Average travel time from one tram station to another = 3 minutes;

• Average travel time from one metro station to another = 1.5 minutes;

• Average transfer time from one tram or metro line to the next tram or metro line = 4

minutes.

• Mechanic "Harry" lives at "Sam van Houtenstraat 3, 1067JA in Amsterdam"

• Mechanic “Peter” lives at “Daalwijkdreef 27, 1103AD in Amsterdam”

• Mechanic “Kees” lives at “Arent Janszoon Ernststraat 11, 1083GP in Amsterdam”

Tram and metro lines GVB

For a visual overview / tool for the tram and metro network, see:

https://www.gvb.nl/sites/default/files/lijnkaart2019-2.pdf

Note: the rail map is sometimes adjusted by the GVB. So you may find these on some

points does not match the GVB\_X\_1 files. It's more about being a resource

to visually display the GVB\_X\_1 files