Journey Planning: The Greedy Algorithm

Case Study: Yogyakarta

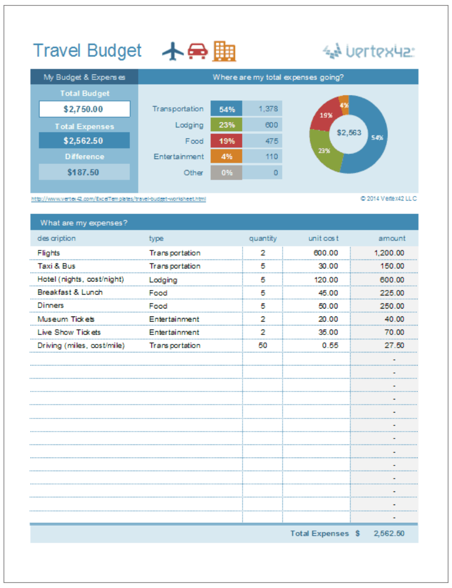
Karina Imani / 13519166
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: 13519166@std.stei.itb.ac.id

***Abstract*—Journey planning is often a vital part in ensuring a trip well executed. There are many aspects worth considering in choosing each destination in a road trip to ensure efficiency and good use of resources, such as travel distance, travel order, as well as the cost and time needed, which are more often than not closely related with each other. Several algorithm strategies, combined with the needed heuristic considerations, may be the perfect solution in arranging an efficient trip. This paper explores the aforementioned possibilities as well as create an implementation and case study for a round trip in Yogyakarta, Indonesia.**

***Keywords*—journey planning, algorithm strategies, heuristic considerations.**

I. Introduction

Journeys are perhaps one of the more common occurrences in one's daily life, be it for business arrangements, leisurely travels, or other reasons. And most certainly, they do not come without expending resources such as time, money, and the traveler's own stamina during the process, which is why they are oftentimes budgeted to fit within the traveler's capabilities—that is to say, careful planning will be needed to ensure the best possible experience within the aforementioned limitations.



Travel Budget

vertex42

My Budget & Expenses	
Total Budget	$2,750.00
Total Expenses	$2,562.50
Difference	$187.50

Where are my total expenses going?

Transportation	54%	1,378
Lodging	23%	600
Food	19%	475
Entertainment	4%	110
Other	0%	0

http://www.vertex42.com/ExcelTemplates/travel-budget-worksheet.html © 2014 Vertex42 LLC

What are my expenses?

description	type	quantity	unit cost	amount
Flights	Transportation	2	600.00	1,200.00
Taxi & Bus	Transportation	5	30.00	150.00
Hotel (nights, cost/night)	Lodging	5	120.00	600.00
Breakfast & Lunch	Food	5	45.00	225.00
Dinners	Food	5	50.00	250.00
Museum Tickets	Entertainment	2	20.00	40.00
Live Show Tickets	Entertainment	2	35.00	70.00
Driving (miles, cost/mile)	Transportation	50	0.55	27.50
				-
				-
				-
				-
				-
				-
				-
				-
				-
				-
				-
				-
				-
			Total Expenses $	2,562.50

Figure 1. An example of travel budgeting. (Source: vertex42.com)

This is where journey planning comes into action. In journey planning, several considerations are taken into account, and these are divided into two broad categories.

The first one is the set of all resources available, as well as how much the planner would be willing to expend. This includes all the resources listed in the previous paragraph: time, money, and stamina. The second one is the set of all aspects of a trip that may be compacted in a more efficient way in order to have 'more for your money'. This will be the main focus of the paper as well as our ultimate goal in journey planning, to minimize the resources expended in a journey, or to compact as many destinations or activities possible within a budget.

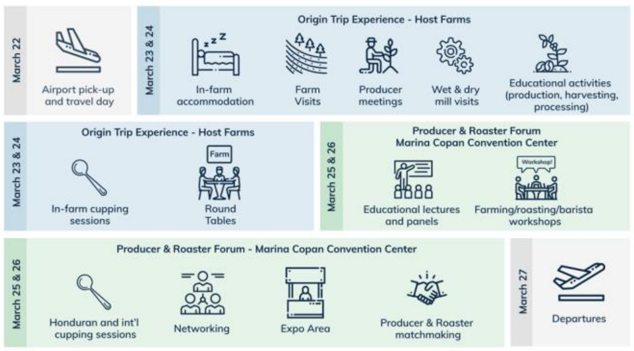

March 22
Airport pick-up and travel day
March 23 & 24
Origin Trip Experience - Host Farms
In-farm accommodation
Farm Visits
Producer meetings
Wet & dry mill visits
Educational activities (production, harvesting, processing)
March 23 & 24
Origin Trip Experience - Host Farms
In-farm cupping sessions
Farm
Round Tables
March 25 & 26
Producer & Roaster Forum
Marina Copan Convention Center
Educational lectures and panels
Workshop!
Farming/roasting/barista workshops
March 25 & 26
Producer & Roaster Forum - Marina Copan Convention Center
Honduran and int'l cupping sessions
Networking
Expo Area
Producer & Roaster matchmaking
March 27
Departures

Figure 2. An example of travel itinerary. (Source: producerroasterforum.com)

Over the years, many algorithm strategies have been developed in order to make journey planning more efficient—to name a few: the greedy algorithm, breadth and depth first search, A* algorithm, and dynamic programming. These algorithms don't always optimize the same resources or aspects in journey planning. For example, the greedy algorithm may be used to maximize the number of stops in a round trip, or to minimize the cost or time needed while picking the best destinations. Meanwhile, the A* algorithm may be used in picking the shortest route that passes through all the planned destinations.

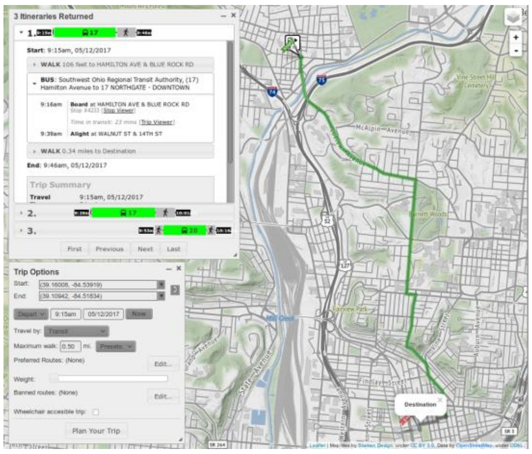
This paper proposes a strategy for the automation of journey planning, namely using the greedy algorithm to choose between a number of destinations, whilst taking into account the general budget available.

II. Theory

A. *Journey Planning*

Journey planning is an activity for optimizing the means of travelling between two or more locations. Its process may make use of an automated search engine (called a journey planner) in order to optimize certain criteria, for example, the least amount of time, distance, or cost, or the maximum amount of stops. Journey planning may be differentiated from route planning, in that a single journey may use different modes of transportation, public or private, as opposed to the continuous and consistent use of a single mode of transportation in the latter term. [1]

A journey planner utilizes different algorithm strategies, converted into a structured, modular program and completed with a graphical user interface. It accepts inputs such as transport modes (to include and/or exclude), trip optimization preferences (i.e., shortest travel distance or fastest travel time), trip cost optimization preferences (i.e., cheapest or best quality destinations), among many others. [2]



3 Itineraries Returned

1. 17

Start: 9:15am, 05/12/2017

WALK 106 feet to HAMILTON AVE & BLUE ROCK RD

BUS: Southwest Ohio Regional Transit Authority, (17) Hamilton Avenue to 17 NORTHGATE - DOWNTOWN

9:16am Board at HAMILTON AVE & BLUE ROCK RD

Stop #4233 (Stop Viewer)

Time in transit: 23 mins (Trip Viewer)

9:39am Alight at WALNUT ST & 14TH ST

WALK 0.34 miles to Destination

End: 9:46am, 05/12/2017

Trip Summary

Travel 9:15am, 05/12/2017

2. 17

3. 20

Trip Options

Start: (39.16008, -84.53919)

End: (39.10942, -84.51834)

Depart 9:15am 05/12/2017

Travel by: Transit

Maximum walk: 0.50 mi.

Preferred Routes: (None)

Banned routes: (None)

Wheelchair accessible trip: ☐

Figure 3. Screenshot of SORTA's Open Trip Planner. (Source: github.com/opentripplanner)

B. *The Greedy Algorithm*

The greedy algorithm is an algorithm which makes the locally optimal choice at each stage. In many cases, this type of strategy does not produce an optimal solution, but may yield locally optimal solutions that approximate a globally optimal solution within a reasonable amount of time. [3]

Greedy algorithms may be broken down into five basic components, such as:

1. A candidate set, which is the set of choices that may be chosen on each step.
2. A solution set, which is the set of candidates chosen as the local optimum.
3. A selection function, which chooses the best among those in the candidate set.
4. A feasibility function, which determines if a candidate may be chosen as a solution.
5. An objective function, which determines the optimization method.
6. A solution function, which indicates if the candidate set is a solution. [4]

Furthermore, the common schema for a greedy algorithm is detailed below:

```
S = {}
while not solution(S) or not C = {} do
    x <- selection(C)
    C <- C - {x}
    if feasible(x) then
        S <- S + {x}
if solution(s) then
    return S
else
    return {}
```

As previously stated, since the principle of the greedy algorithm is to find the local optimum in each step, the solution provided by this algorithm may not always be the global optimum, but it would provide a good approximation in a reasonable amount of time.

C. *Yogyakarta*

Yogyakarta is the capital city of the Special Region of Yogyakarta in Indonesia, specifically, the island of Java. It is regarded as an important center for traditional Javanese fine arts (some examples include batik textiles, wayang puppetry, and such) as well as education, both basic and higher—it is home to a large student population and houses one of the largest and most prestigious universities, namely, Universitas Gadjah Mada, earning it the nickname "the city of students".

In addition to arts and education, Yogyakarta is rich in its history as well, previously occupied by a few of the strongest and most long-lasting monarchies among the whole archipelago of Indonesia, including Mataram Kingdom, Majapahit Empire, and Mataram Sultanate. It has also seen a fair share of historical events, before, during, and after the colonialism and imperialism by the Netherlands and Japan, and had even once become the capital of the Republic of Indonesia (1946–1948).

Figure 4. Keraton Yogyakarta.
(Source: Wikipedia)

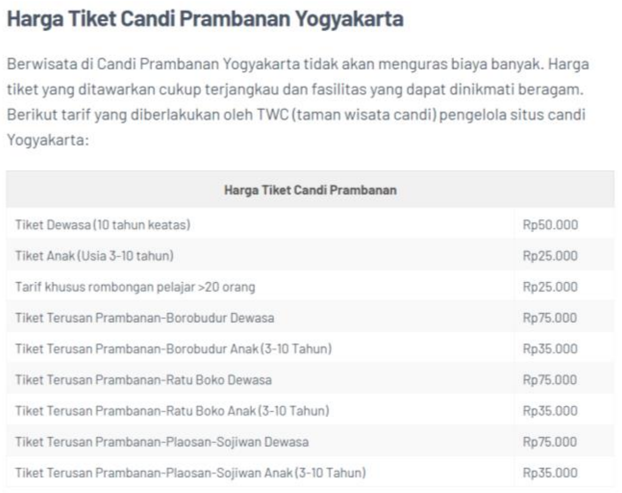
Due to its rich present and colorful past, Yogyakarta hosts a number of tourist attractions, cuisine and activities alike, which makes it a perfect candidate for the case study of a journey planning. It is also chosen due to its deep roots in tradition, reflecting a portion, although not representative, of Indonesian culture.

III. Application

A. Proposition

A greedy algorithm may be used to approximate optimizing resources based on a certain criteria, i.e., the efficient use of time or financial resources. While it does not always produce the optimum solution in most cases, it suits the problem at hand in the sense that it may not completely use up all the resources owned—in journey planning, extra resources are always welcome in case of emergency and/or additional needs.

The idea of the program is simple. The user lists a number of destinations, as well as the cost (in IDR) and time (in hours) they are willing to spend in each destination. This information can be provided from looking through various travel guides, reflecting the combined fees of entrance, facilities, and amenities of a destination, or the time needed to do all activities within a destination.


Harga Tiket Candi Prambanan Yogyakarta

Berwisata di Candi Prambanan Yogyakarta tidak akan menguras biaya banyak. Harga tiket yang ditawarkan cukup terjangkau dan fasilitas yang dapat dinikmati beragam. Berikut tarif yang diberlakukan oleh TWC (taman wisata candi) pengelola situs candi Yogyakarta:

Harga Tiket Candi Prambanan	
Tiket Dewasa (10 tahun keatas)	Rp50.000
Tiket Anak (Usia 3-10 tahun)	Rp25.000
Tarif khusus rombongan pelajar >20 orang	Rp25.000
Tiket Terusan Prambanan-Borobudur Dewasa	Rp75.000
Tiket Terusan Prambanan-Borobudur Anak (3-10 Tahun)	Rp35.000
Tiket Terusan Prambanan-Ratu Boko Dewasa	Rp75.000
Tiket Terusan Prambanan-Ratu Boko Anak (3-10 Tahun)	Rp35.000
Tiket Terusan Prambanan-Plaosan-Sojiwan Dewasa	Rp75.000
Tiket Terusan Prambanan-Plaosan-Sojiwan Anak (3-10 Tahun)	Rp35.000

Figure 5. A query for estimate prices.
(Source: travelspromo.com)

For better time considerations, the planner should also include a matrix, its width and height the number of destinations, which represents the time taken to go between two destinations, the x and y (or i and j) indexes. Again, this information can be provided from looking through map applications, querying both places in the search boxes, and taking note of the travel time between the two. It should be noted that, due to one- and two-way traffic, the time taken to get from location A to B isn't always equivalent with B to A.

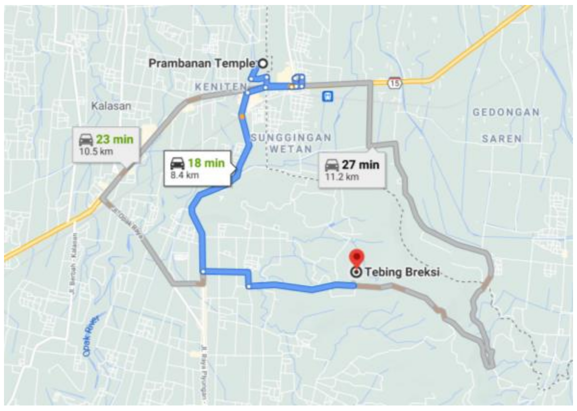


Figure 6. A query for estimate prices.
(Source: Google Maps)

Once we have both types of data at the ready, we can immediately implement our greedy algorithm.

Essentially, the idea behind our greedy algorithm is to minimize the time taken to travel between locations. The reason for this consideration is, even though we could just as easily create a greedy algorithm to minimize the cost of a trip, or maximize the time spent on a trip, perhaps the most efficient way of planning a trip is to stay on the road less—this means less transportation cost and less travel time, a win on both ends.

As such, we shall compare the time needed to travel between destinations and choose the smallest number among them. Of course, the total cost and time of our journey's budget, as mentioned before, would still be taken into account. In addition to that, should there be two destinations with the same travel time from our 'current location', we would choose the one with less cost, or the one with more time.

B. Implementation

For ease of implementation, our program would be completely written in the Python programming language. Due to the program's moderate simplicity, and Python's relative efficiency, there are no libraries that must be included—the input only makes use of the typical array, matrix, and string parser, and the main body only makes use of iterations and if-else branches.

The program reads a text file containing the needed data, mentioned in the previous section, as well as a few additional attributes to make processing easier:

1. The first line should state the number of destinations listed in the file, n.
2. The next n lines should state the data of each destination, i.e., the cost, time, and distance from our initial location (in this case, accommodations). Each data should be separated by a vertical bar sandwiched between two spaces (" | ").
3. The last n lines should state a set of numbers, each line holding n numbers, each representing the time taken to travel between two places. When seen as a whole, they should form an n x n matrix whose indexes are

representative of the order in which the data of each destination is mentioned in the file.

The text file should roughly look like this:

```
3
Destination 1 } 1000 | 10 | 10
Destination 2 } 2000 | 20 | 20
Destination 3 } 3000 | 30 | 30
0 1 2
3 0 4
5 6 0
```

In this case, visiting "destination 1" would cost you IDR 1,000 and take you 10 minutes. Furthermore, the time it would take to travel between "destination 1" and your accommodation would be 10 minutes.

As for the matrix, the number 0 at the top left corner indicates that it would take 0 minute to travel from "destination 1" to "destination 1", since they are essentially the same location. The number 1 right next to it indicates that it would take 1 minute to travel from "destination 1" to "destination 2".

Once the input files have been decided, we can decide how to process the data and implement our greedy algorithm. Once completed, the program we are working on should be able to do as follows:

```
Input the text file name: <user input>

Obtained data:
['Destination 1', 50000, 180, 13]
['Destination 2', 80000, 240, 17]
['Destination 3', 15000, 90, 25]

Budget available (in IDR): <user input>
Time available (in minutes): <user input>

Your destinations:
 - <list item>
 - <list item>
Total cost: <combined cost>
Total time: <combined time>
```

As shown in the box above, our program should be able to take a file name as its first input, open the file, and show its contents. This will be done with the use of functions input(), open(), and readline()—none of which needs a library to work. The input will then be parsed into an array with the function `str.split(<divider>)`, after having its trailing new line ("\n") removed with the function `str.rstrip("\n")`.

Afterwards, it would take the cost and time the user is willing to spend on the trip as a whole, and use it to generate an array of destinations with the minimum travel time, all within the constraints of the cost and time mentioned before.

The code for the main program is pasted below, annotated with comments detailing each step:

```
file = input("Input the text file name: ")
f = open(file)

# Read the number of entries
n = f.readline()

# Initialize empty arrays
c1 = {}
c2 = {}
t1 = {}
t2 = {}

# Read entries
for i in range (int(n)):
    c1[i] = f.readline().rstrip("\n")
    c2[i] = c1[i].split(' | ')
    # Change numeric entries to int
    c2[i][1] = int(c2[i][1])
    c2[i][2] = int(c2[i][2])
    c2[i][3] = int(c2[i][3])

# Read matrix
for i in range (int(n)):
    t1[i] = f.readline().rstrip("\n")
    t2[i] = t1[i].split(' ')
    # Change numeric entries to int
    for j in range (int(n)):
        t2[i][j] = int(t2[i][j])

# Print entries
print("\nObtained data: ")

for i in range (int(n)):
    print(c2[i])

# Get budget
cost = input("\nBudget available (in IDR): ")
time = input("Time available (in minutes): ")

# Greedy algorithm
greedy(int(n), c2, t2, int(cost), int(time))
```

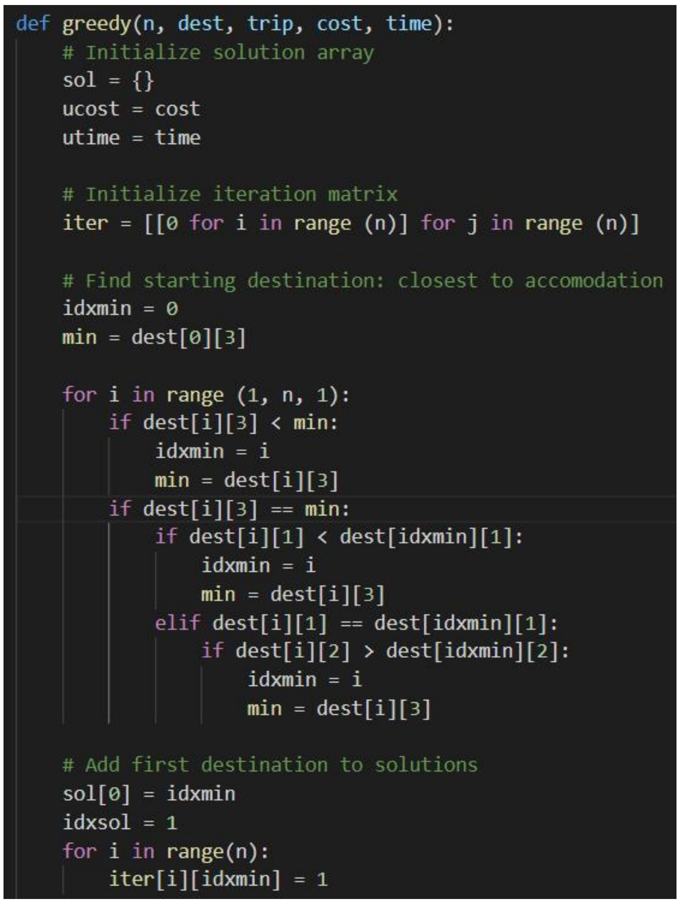
The greedy algorithm on the last line is in charge of producing the array of destinations that make up the solution, as well as printing the results in the predetermined manner. In implementing our greedy algorithm, let us first define each of its components, as listed in chapter II:

1. Candidate set: the set of destinations—in this case, a set of numbers corresponding to the indexes of each destination within the entry array.
2. Solution set: the set of destinations which minimizes the time taken to travel between the destinations, within the set cost and time.
3. Selection function: a simple iteration to find the smallest number among the travel time from the

'current location', aka the last location in the solution set at the time.

4. Feasibility function: an if-else branch that checks whether a destinations cost and time has already exceeded the owned resources.
5. Objective function: minimizes travel time between destinations while taking into account the cost and time allocated by the user.
6. Solution function: the function stops when no more destinations can be added, or if the number of destinations is the same as the number of destinations in the solution set.

With components as such, we implement the greedy algorithm as follows. The following three figures make up the whole of our greedy function—even though they take care of different steps within the function, they are not to be treated as separate segments of code.

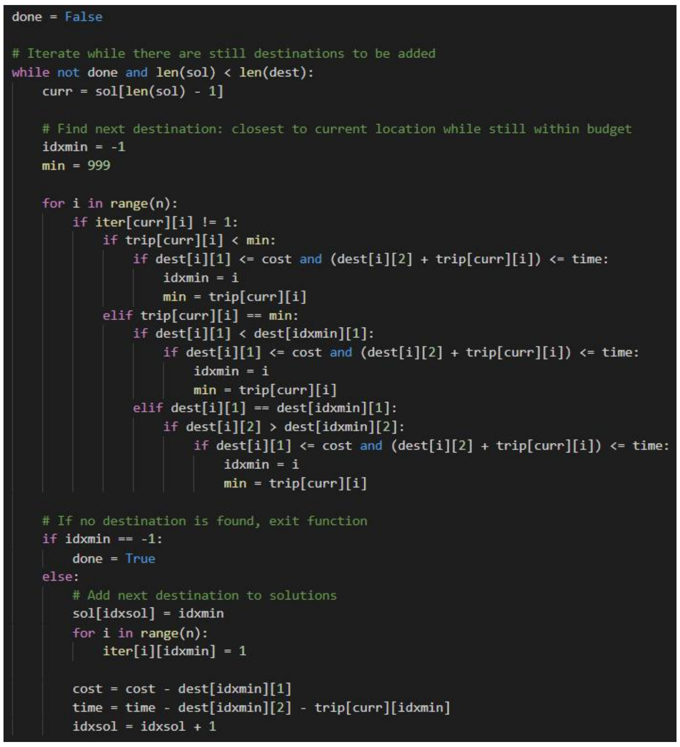


```
def greedy(n, dest, trip, cost, time):
    # Initialize solution array
    sol = {}
    ucost = cost
    utime = time

    # Initialize iteration matrix
    iter = [[0 for i in range (n)] for j in range (n)]

    # Find starting destination: closest to accomodation
    idxmin = 0
    min = dest[0][3]

    for i in range (1, n, 1):
        if dest[i][3] < min:
            idxmin = i
            min = dest[i][3]
        if dest[i][3] == min:
            if dest[i][1] < dest[idxmin][1]:
                idxmin = i
                min = dest[i][3]
            elif dest[i][1] == dest[idxmin][1]:
                if dest[i][2] > dest[idxmin][2]:
                    idxmin = i
                    min = dest[i][3]

    # Add first destination to solutions
    sol[0] = idxmin
    idxsol = 1
    for i in range(n):
        iter[i][idxmin] = 1
```

Figure 7. The greedy function, part 1.
(Source: Author implementation)

The first part of the code initializes the the solution set as an empty array, as well as keep the original values of cost and time in variables ucost and utime (read unchanged cost and time). Afterwards, the function also initializes an matrix of zeroes with a width and height the number of entries (n x n). This matrix, called the iteration matrix, sets the visited destinations as 1, and the unvisited ones as 0.

Below that, we have a segment of the code to try and fine the first destination—the one closest to our accommodation, aka the one with the smallest number of travel time from our accommodation.

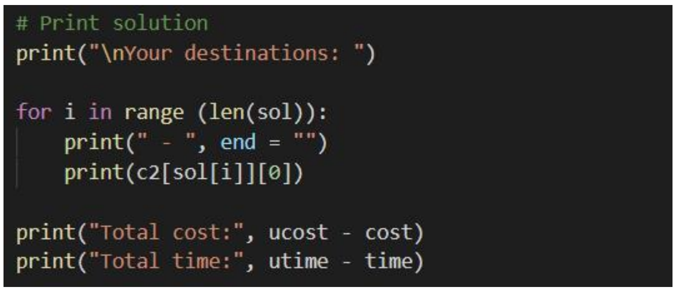


```
done = False

# Iterate while there are still destinations to be added
while not done and len(sol) < len(dest):
    curr = sol[len(sol) - 1]

    # Find next destination: closest to current location while still within budget
    idxmin = -1
    min = 999

    for i in range(n):
        if iter[curr][i] != 1:
            if trip[curr][i] < min:
                if dest[i][1] <= cost and (dest[i][2] + trip[curr][i]) <= time:
                    idxmin = i
                    min = trip[curr][i]
            elif trip[curr][i] == min:
                if dest[i][1] < dest[idxmin][1]:
                    if dest[i][1] <= cost and (dest[i][2] + trip[curr][i]) <= time:
                        idxmin = i
                        min = trip[curr][i]
                elif dest[i][1] == dest[idxmin][1]:
                    if dest[i][2] > dest[idxmin][2]:
                        if dest[i][1] <= cost and (dest[i][2] + trip[curr][i]) <= time:
                            idxmin = i
                            min = trip[curr][i]

    # If no destination is found, exit function
    if idxmin == -1:
        done = True
    else:
        # Add next destination to solutions
        sol[idxsol] = idxmin
        for i in range(n):
            iter[i][idxmin] = 1

        cost = cost - dest[idxmin][1]
        time = time - dest[idxmin][2] - trip[curr][idxmin]
        idxsol = idxsol + 1
```

Figure 8. The greedy function, part 2.
(Source: Author implementation)

The second part of the function deals with more iteration. It is started with a while branch that checks if there are no more destinations that can be added, or if the number of destinations is already the same as the number of destinations in the solution set. If there are still destinations left to be added, we find the next destination—the one with the least travel time from our current location.

The mechanics of this portion is similar to the previous one, with multiple if-else branches, except this one deals with the travel time matrix instead of the destination data entries.



```
# Print solution
print("\nYour destinations: ")

for i in range (len(sol)):
    print(" - ", end = "")
    print(c2[sol[i]][0])

print("Total cost:", ucost - cost)
print("Total time:", utime - time)
```

Figure 9. The greedy function, part 2.
(Source: Author implementation)

The final part of the function is a simple print segment to print out the proposed list of destinations as well as the total cost and time expended to travel through them.

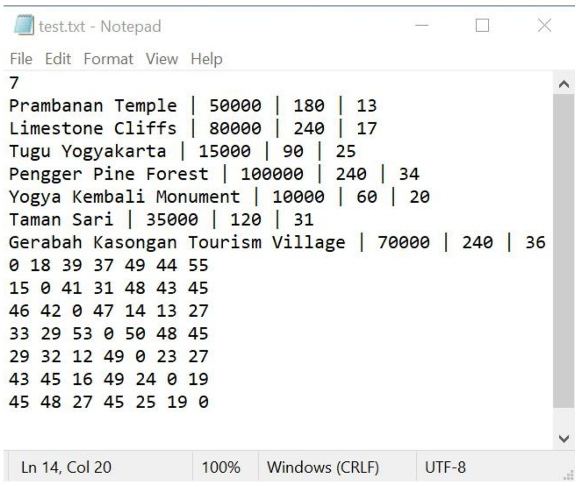
C. *Case Study*

As examples for this method will be using a case study of travelling in Yogyakarta, it would be efficient to first decide which destinations should be considered in journey planning. Based on several travel guides and personal experiences, they are listed as such:

1. Prambanan Temple
2. Limestone Cliffs
3. Tugu Yogyakarta
4. Pengger Pine Forest
5. Yogya Kembali Monument
6. Taman Sari
7. Gerabah Kasongan Tourism Village

Of course, we must also pick a means of accommodation as a starting point. In this case, we chose the University Hotel. This means, on every trip planned, we must

In the course of implementation, further information would be procured for each of the destinations listed above, as considerations in planning the optimized cost. From querying and taking notes of the required data for both the destination entries and travel time matrix, we produce a file we will call "test.txt".

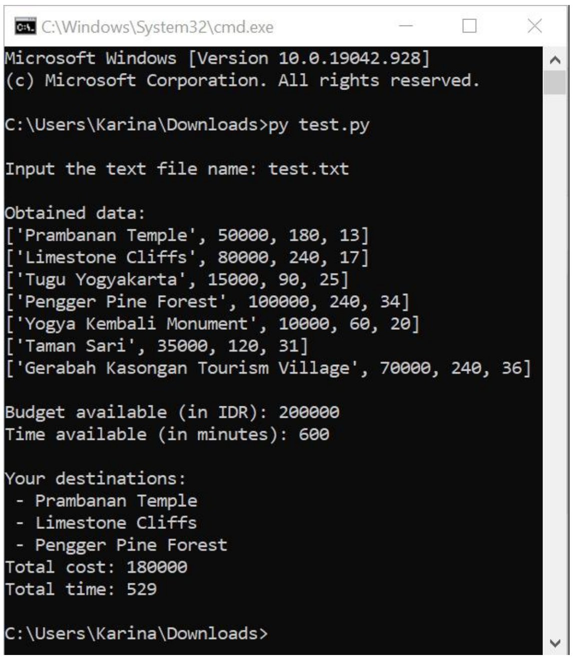


test.txt - Notepad
File Edit Format View Help
Ln 14, Col 20 100% Windows (CRLF) UTF-8

```
7
Prambanan Temple | 50000 | 180 | 13
Limestone Cliffs | 80000 | 240 | 17
Tugu Yogyakarta | 15000 | 90 | 25
Pengger Pine Forest | 100000 | 240 | 34
Yogya Kembali Monument | 10000 | 60 | 20
Taman Sari | 35000 | 120 | 31
Gerabah Kasongan Tourism Village | 70000 | 240 | 36
0 18 39 37 49 44 55
15 0 41 31 48 43 45
46 42 0 47 14 13 27
33 29 53 0 50 48 45
29 32 12 49 0 23 27
43 45 16 49 24 0 19
45 48 27 45 25 19 0
```

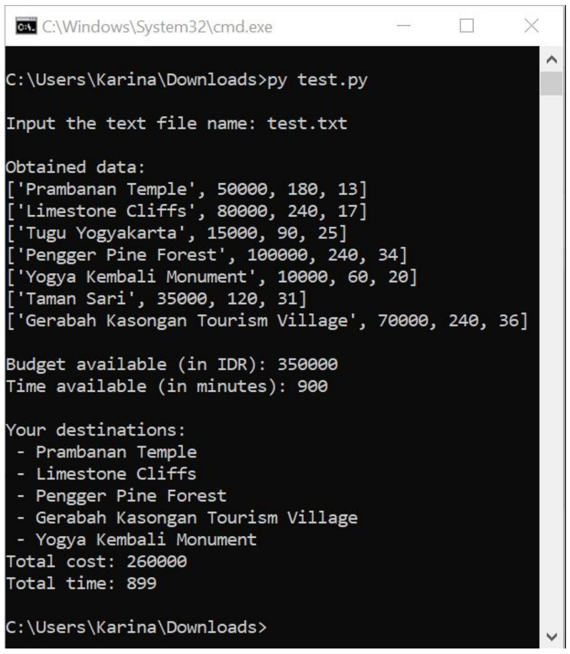
Figure 10. The contents of test.txt.
(Source: Author implementation)

The following are several tests for the program in the previous section, using the same input text file, but with different cost and time limits.



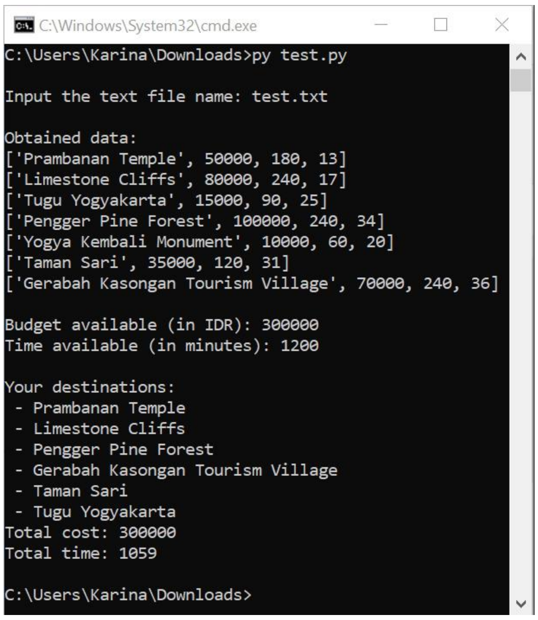
```
Microsoft Windows [Version 10.0.19042.928]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Karina\Downloads>py test.py

Input the text file name: test.txt

Obtained data:
['Prambanan Temple', 50000, 180, 13]
['Limestone Cliffs', 80000, 240, 17]
['Tugu Yogyakarta', 15000, 90, 25]
['Pengger Pine Forest', 100000, 240, 34]
['Yogya Kembali Monument', 10000, 60, 20]
['Taman Sari', 35000, 120, 31]
['Gerabah Kasongan Tourism Village', 70000, 240, 36]

Budget available (in IDR): 200000
Time available (in minutes): 600

Your destinations:
 - Prambanan Temple
 - Limestone Cliffs
 - Pengger Pine Forest
Total cost: 180000
Total time: 529

C:\Users\Karina\Downloads>
```

Figure 11. Test 1, with cost 200,000 and time 600.
(Source: Author implementation)



```
C:\Users\Karina\Downloads>py test.py

Input the text file name: test.txt

Obtained data:
['Prambanan Temple', 50000, 180, 13]
['Limestone Cliffs', 80000, 240, 17]
['Tugu Yogyakarta', 15000, 90, 25]
['Pengger Pine Forest', 100000, 240, 34]
['Yogya Kembali Monument', 10000, 60, 20]
['Taman Sari', 35000, 120, 31]
['Gerabah Kasongan Tourism Village', 70000, 240, 36]

Budget available (in IDR): 350000
Time available (in minutes): 900

Your destinations:
 - Prambanan Temple
 - Limestone Cliffs
 - Pengger Pine Forest
 - Gerabah Kasongan Tourism Village
 - Yogya Kembali Monument
Total cost: 260000
Total time: 899

C:\Users\Karina\Downloads>
```

Figure 12. Test 2, with cost 350,000 and time 900.
(Source: Author implementation)



```
C:\Users\Karina\Downloads>py test.py

Input the text file name: test.txt

Obtained data:
['Prambanan Temple', 50000, 180, 13]
['Limestone Cliffs', 80000, 240, 17]
['Tugu Yogyakarta', 15000, 90, 25]
['Pengger Pine Forest', 100000, 240, 34]
['Yogya Kembali Monument', 10000, 60, 20]
['Taman Sari', 35000, 120, 31]
['Gerabah Kasongan Tourism Village', 70000, 240, 36]

Budget available (in IDR): 300000
Time available (in minutes): 1200

Your destinations:
 - Prambanan Temple
 - Limestone Cliffs
 - Pengger Pine Forest
 - Gerabah Kasongan Tourism Village
 - Taman Sari
 - Tugu Yogyakarta
Total cost: 300000
Total time: 1059

C:\Users\Karina\Downloads>
```

Figure 13. Test 3, with cost 300,000 and time 1200. (Source: Author implementation)

As can be proven through manual computation, the results of the tests are consistent with how the program is structured and implemented, meaning there are no errors with the program. For the efficiency of the program itself, as with most greedy strategies for optimization problems, surely there may be cases where an optimum solution is not produced, but in each case, a good approximation is given within a reasonable amount of time.

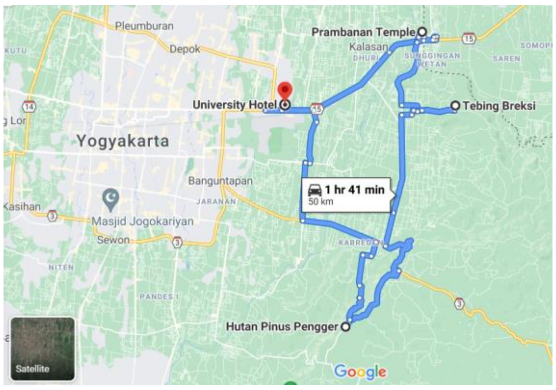


Figure 14. Example map of the route produced with greedy algorithm. (Source: Google Maps)

IV. Conclusion

The greedy algorithm may be used to approximate the solution to optimization problems adequately within a reasonable amount of time. One of these problems is journey planning—in this implementation, by minimizing the travel time between destinations while taking into account the general cost and time limitations of the trip.

To further improve the program, the author proposes web scraping to gather data from the internet, such as the time and cost expended in each destination, and the travel times between destinations obtained from map applications. The author hopes that the creation of this program may help with journey planning, as well as be developed into something more efficient and easier to use.

Video Link at YouTube

To complement this paper, a Youtube video with a verbal explanation and program demonstration has been made and posted with the following link:

https://youtu.be/YqyVTAK3bNk

Acknowledgment

First and foremost, I would like to thank God for his blessing and guidance, for this essay would not have been completed without his grace. I would also like to express my gratitude towards my IF2211 Algorithm Strategies lecturer, Mr. Rinaldi Munir for the opportunity to put the knowledge I have gained through his classes into words of an essay. Lastly, I would like to thank my sister and friends for the suggestions and examples regarding the topic of this essay.

References

[1] Li, Jing-Quan; Zhou, Kun; Zhang, Liping; Zhang, Wei-Bin, "A Multimodal Trip Planning System With Real-Time Traffic and Transit Information" in Journal of Intelligent Transportation Systems, p. 60–66. 2012.

[2] OpenTripPlanner.org, "OpenTripPlanner" in GitHub. Retrieved 10 May 2021.

[3] Black, Paul E., "greedy algorithm" in Dictionary of Algorithms and Data Structures. U.S. National Institute of Standards and Technology (NIST). Retrieved 10 May 2021.

[4] Munir, Rinaldi, "Algoritma Greedy (Bagian I)" in Bahan Kuliah IF2211 Strategi Algoritma. Retrieved 10 May 2021.

[5] Munir, Rinaldi, "Dynamic Programming (Bagian I)" in Bahan Kuliah IF2211 Strategi Algoritma. Retrieved 10 May 2021.

[6] Cormen, T. H.; Leiserson, C. E.; Rivest, R. L.; Stein, C., Introduction to Algorithms (2nd ed.), MIT Press & McGraw–Hill. 2001.

Statement

With this, I hereby state that this paper is purely in my own writing, and neither an adaptation nor a translation of someone else's, as well as a plagiarism.

Jakarta, 11 Mei 2021

[illegible]

Karina Imani / 13519166