Application for Arranging Travel Itinerary in the City of Bandung Using Greedy Algorithm

Kintan Sekar Adinda
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13517102@std.stei.itb.ac.id

This paper explains a little about how to apply the greedy algorithm to arrange an itinerary that contains two attractions and a place to eat that can be done on a one-day vacation in Bandung. The greedy algorithm is one algorithm for optimizing problems. Making this paper aims to facilitate application users during the holidays and help us understand the greedy algorithm.

Keywords—greedy; itinerary; profit; weight

I. Introduction

Bandung is one of the favorite destinations of many people during holiday. So many places that can be visited, starting from nature, shopping and culinary tourism. Too many choices might make us confused, which place is the most recommended and closest to our position.

If we search in the search engines, we can find a variety of tourist destinations in Bandung. However, of course we cannot visit all these destinations. In one day maybe we can only visit one to two tourist attractions. Therefore, we must be smart in choosing as many tourist attractions as possible with high ratings and located close to our current position. For example, now we are at ITB. We have three tourism options, namely Tangkuban Perahu, Cisangkuy, and Braga where all three have the same rating. Of the three choices, we better choose Cisangkuy and Braga because the distance tends to be closer to our current position. Likewise, by choosing a place to eat. We can choose culinary tours that are close to Cisangkuy and Braga.

In this case, the author designed an application to arrange a traveling itinerary in Bandung. This application detects tourist locations that are close to our current position and helps choose which destinations are worth visiting. Not only tourist attractions, this application also helps us to determine culinary tours that are located close to the tourist attractions we choose. To determine which tourist attractions are worth visiting, a greedy algorithm is used. Greedy algorithm is commonly used to solve the knapsack problem. There are several variables that will be used as benchmarks in implementing the greedy algorithm. This application will make it easier for us to optimize vacation time.

II. Theoretical Basis

A. *Greedy Algorithm*

The greedy method suggests that one can devise an algorithm that works in stages, considering one input at a time. At each stage, a decision is made regarding whether a particular input is in an optimal solution. This is done by considering the inputs in an order determined by some selection procedure. If the inclusion of the next input into the partially constructed optimal solution will result in an infeasible solution, then this input is not added to the partial solution. Otherwise, it is added. The selection procedure itself is based on some optimization measure. This measure may be the objective function. In fact, several different optimization measures may be plausible for a given problem. Most of these, however, will result in algorithms that generate suboptimal solutions. This version of the greedy technique is called the subset paradigm.

We can describe the subset paradigm abstractly, but more precisely than above, by considering the control abstraction in following code.

```
Algorithm Greedy($a, n$)
// $a[1:n]$ contains the $n$ inputs.
{
    $solution := \emptyset$; // Initialize the solution.
    for $i := 1$ to $n$ do
    {
        $x :=$ Select($a$);
        if Feasible($solution, x$) then
            $solution :=$ Union($solution, x$);
    }
    return $solution$;
}
```

B. Knapsack Problem

Let us try to apply the greedy method to solve the knapsack problem. We are given *n* objects and a knapsack or bag. Object *i* has a weight and the knapsack has a capacity *m*. If a fraction x_i, $0 \leq x_i \leq 1$, of object i is placed into the knapsack, then a profit of $p_i x_i$ is earned. The objective is to obtain a filling of the knapsack that maximizes the total profit earned. Since the knapsack capacity is *m*, we require the total weight of all chosen objects to be at most *m*. Formally, the problem can be stated as

$$\text{maximize} \sum_{1 \leq i \leq n} p_i x_i$$
$$\text{subject to} \sum_{1 \leq i \leq n} w_i x_i \leq m$$
$$\text{and } 0 \leq x_i \leq 1, \quad 1 \leq i \leq n$$

The profits and weights are positive numbers.

Insert objects one by one into knapsack. Once an object is entered into knapsack, the object cannot be taken out again.

There are some greedy strategies that are heuristic which can be used to select objects that are will be entered into knapsack:

1. Greedy by profit

   At each step, the knapsack is filled with objects which has the biggest profit. This strategy tries to maximize profits by choosing the most profitable object first.

2. Greedy by weight

   At each step, the knapsack is filled with objects that have the lightest weight. This strategy tries to maximize profits by entering as many objects as possible into the knapsack.

3. Greedy by density

   At each step, the knapsack is filled with objects that have the largest number of density. This strategy tries to maximize profits by selecting objects that have the greatest profit per unit weight.

The choice of object based on one of the three strategies above does not guarantee that it will provide an optimal solution. There is even a possibility that the three strategies do not provide optimum solutions. The following example illustrates these cases.

Review the 0/1 Knapsack problem with n = 4.

w1 = 2; p1 = 12

w2 = 5; p1 = 15

w3 = 10; p1 = 50

w4 = 5; p1 = 10

Knapsack capacity W = 16

Solution by greed algorithm:

Object Property				Greedy by			Optimal Solution
i	w_i	p_i	p_i/w_i	profit	weight	density	
1	6	12	2	0	1	0	0
2	5	15	3	1	1	1	1
3	10	50	5	1	0	1	1
4	5	10	2	0	1	0	0
Total weight				15	16	15	15
Total profit				65	37	65	65

In this example, the greedy algorithm with the object selection strategy based on profit and density provides the optimal solution, while the selection of objects based on weight does not provide an optimal solution.

C. *Java Languange*

Java is a general-purpose programming language that is class-based, object-oriented, and specifically designed to have as few implementation dependencies as possible. It is intended to let application developers "write once, run anywhere" (WORA), meaning that compiled Java code can run on all platforms that support Java without the need for recompilation. Java applications are typically compiled to "bytecode" that can run on any Java virtual machine (JVM) regardless of the underlying computer architecture. The syntax of Java is similar to C and C++, but it has fewer low-level facilities than either of them. As of 2018, Java was according to GitHub one of the most popular programming languages in use, particularly for client-server web applications, with a reported 9 million developers.

There are quite a number of academics in Indonesia who use the Java language as a tool to complete a thesis or final assignment with various topics. Therefore, the author wants to use java language as a reference to apply the greedy algorithm to knapsack problems.

Like any programming language, the Java language has its own structure, syntax rules, and programming paradigm. The Java language's programming paradigm is based on the concept of OOP, which the language's features support.

The Java language is a C-language derivative, so its syntax rules look much like C's. For example, code blocks are modularized into methods and delimited by braces ({ and }), and variables are declared before they are used.

Structurally, the Java language starts with *packages*. A package is the Java language's namespace mechanism. Within packages are classes, and within classes are methods, variables, constants, and more.

```
public class HelloWorld {

    public static void main(String[] args) {
         // prints "Hello, World!"
         System.out.println("Hello, World!");
    }
}
```

III. Sampling

In using this application, the location of our current position is needed. This is useful for finding the location of the nearest tourist destination and place to eat. For example our position is now at Bandung Institute Technology. The five closest tourist objects from ITB are

- Bandung Zoo
- Upside Down World
- Bandung Geological Museum
- Gedung Sate
- Museum Konferensi Asia Afrika

The five closest place to eat from ITB are

- Warung Pasta
- Midori Japanese Restaurant
- Kapulaga Indonesian Bistro
- Gyu-Kaku Japanese BBQ Restaurant
- Pizza Hut

From all tourist attractions and places to eat, we determine the weight and profit of each place as a characteristic of the 0/1 Knapsack problem. Determination of weight and profit for this tourist destination is determined from several aspects that are subjectively considered influential and can be a differentiating factor in reviewing the issue of the selection of tourist visits. The process of determining weight and profit is also carried out under several assumptions that can help define the problem to be modeled.

To determine the profit from these places, we need data in the form of ratings from the place along with the price (in the form of the price of admission for tourism objects and food prices for places to eat). Will benefit users, if the rating is high, but the price is cheap. So, the calculation that can be done is if we divide the rating by price, then we will get a profit. The higher the rating, the higher the profit. The lower the price, the higher the profit.

$$\boxed{profit = \frac{rating}{price}}$$

To determine the weight from these places, we need data in the form of the distance of the tourist attractions from our current position and data in the form of the length of trip that must be taken to arrive at that destination. To do a calculation, we can multiply the distance by 50% and multiply the duration of the trip by 50% then add it up, then we will get the weight. The distance and travel time are directly proportional to weight.

$$\boxed{\begin{gathered} weight \\ = 50\% \times distance + 50\% \times time \end{gathered}}$$

To get the value of density, we can divide weight by its profit.

$$\boxed{density = \frac{weight}{profit}}$$

By implementing these two formulas for all tourist attractions and restaurants, we get the following data.

Profit Table for Tourist Attraction			
Tourist Attraction	**Rating**	**Price for two people (approx.) × 1,000,000**	**Profit**
Bandung Zoo	3.8	0.04	95
Upside Down World	4.0	0.1	40
Bandung Geological Museum	4.6	0.01	460
Bosscha	4.5	0.02	225
NuArt Sculpture Park	4.6	0.05	92

Profit Table for Places to Eat			
Places to Eat	**Rating**	**Ticket Price**	**Profit**
Warung Pasta	4.2	0.08	52.5
Midori Japanese Restaurant	4.3	0.2	21.5
Kapulaga Indonesian Bistro	4.2	0.12	35
Gyu-Kaku Japanese BBQ Restaurant	4.7	0.3	15.67
Pizza Hut	4.4	0.2	22

Weight Table for Tourist Attraction			
Tourist Attraction	**Distance (km)**	**Duration of Trip by Car (minute)**	**Weight**
Bandung Zoo	3.9	2	2.95
Upside Down World	5.1	9	7.05
Bandung Geological Museum	5.7	15	10.35
Bosscha	8.4	47	27.7
NuArt Sculpture Park	0.95	30	15.48

Weight Table for Places to Eat			
Places to Eat	**Distance (km)**	**Duration of Trip by Car (minute)**	**Weight**
Warung Pasta	0.45	4	2.23
Midori Japanese Restaurant	1.6	12	6.8
Kapulaga Indonesian Bistro	0.35	6	3.18
Gyu-Kaku Japanese BBQ Restaurant	1.6	12	6.8
Pizza Hut	0.7	9	4.85

Density Table for Tourist Attraction			
Tourist Attraction	**Weight**	**Profit**	**Density** × 100
Bandung Zoo	2.95	95	3.1
Upside Down World	7.05	40	17.63
Bandung Geological Museum	10.35	460	2.25
Bosscha	27.7	225	12.31
NuArt Sculpture Park	15.48	92	16.83

Density Table for Place to Eat			
Places to Eat	**Weight**	**Profit**	**Density** × 100
Warung Pasta	2.23	52.5	4.25
Midori Japanese Restaurant	6.8	21.5	31.63
Kapulaga Indonesian Bistro	3.18	35	9.09
Gyu-Kaku Japanese BBQ Restaurant	6.8	15.67	43.4
Pizza Hut	4.85	22	22.05

This application will help us to choose in one day two attractions and one place to eat. Here is a general holiday schedule before the application chooses a tourist attraction.

1	Tourist Attraction
2	Place to Eat (for lunch)
3	Tourist Attraction

IV. Analysis

In the analysis section, a greedy by density approach will be used because greedy by density tends to lead to optimal solutions compared to greedy by weight or greedy by profit.

A. Result of Greedy Algorithm by Density

Result of Greedy Algorithm for Tourist Attraction		
Number	**Tourist Attraction**	**Density** × 100
1	Upside Down World	17.63
2	NuArt Sculpture Park	16.83
3	Bosscha	12.31
4	Bandung Zoo	3.1
5	Bandung Geological Museum	2.25

Result of Greedy Algorithm for Places to Eat		
Number	**Tourist Attraction**	**Density** × 100
1	Gyu-Kaku Japanese BBQ Restaurant	43.4
2	Midori Japanese Restaurant	31.63
3	Pizza Hut	22.05
4	Kapulaga Indonesian Bistro	9.09
5	Warung Pasta	4.25

Based on the result of greedy algorithm, so the application will display the following itinerary for a day.

1	Upside Down World
2	Gyu-Kaku Japanese BBQ Restaurant
3	NuArt Sculpture Park

B. Code that Implements a Greedy Algorithm

```
class Knapsack {

    static int max(int a, int b) { return
(a > b) ? a : b; }

    static int knapSack(int W, int wt[],
int val[], int n)
    {
        if (n == 0 || W == 0)
            return 0;

        if (wt[n - 1] > W)
            return knapSack(W, wt, val, n
- 1);

        else
            return max(val[n - 1] +
knapSack(W - wt[n - 1], wt, val, n - 1),
                       knapSack(W, wt,
val, n - 1));
    }

    public static void main(String
args[])
    {
        int val[] = new int[] { 60, 100,
120 };
        int wt[] = new int[] { 10, 20, 30
};
        int W = 50;
        int n = val.length;
        System.out.println(knapSack(W,
wt, val, n));
    }
}
```

Conclusion

Various attractions and places to eat are already very much on the Internet. However, so that we can enjoy the holidays optimally, we better arrange the itinerary first. Therefore, we can use the application that the author made to solve the problem. Greedy Algorithm works well as a technology that helps writers to arrange the itinerary properly.

References

[1] Ellis Horowitz, Sartaj Sahni, and Sanguthevar Rajasekaran, "Computer Algorithms," W. H. Freeman and Company 1997.

[2] Munir, Rinaldi. 2006. Diktat Strategi Algoritma.

[3] Binstock, Andrew (May 20, 2015). "Java's 20 Years of Innovation". Forbes. Archived from the original on March 14, 2016. Retrieved March 18, 2016.

[4] https://developer.ibm.com/tutorials/j-introtojava1/, accessed on April 26 2019 at 10.13 am.

[5] https://www.geeksforgeeks.org/java-program-for-dynamic-programming-set-10-0-1-knapsack-problem/, accessed on April 26 2019 at 3.24 pm.

[6] Y. Yorozu, M. Hirano, K. Oka, and Y. Tagawa, "Electron spectroscopy studies on magneto-optical media and plastic substrate interface," IEEE Transl. J. Magn. Japan, vol. 2, pp. 740-741, August 1987 [Digests 9th Annual Conf. Magnetics Japan, p. 301, 1982].

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 29 April 2012

[illegible]

Kintan Sekar Adinda
13517102