# Creating a Helper Bot to Generate a User Shopping List Recommendation Based on their Budget

Mahameru Ds 13519014
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail (gmail): erutan.ds20@gmail.com

***Abstract*—We're often being faced to the difficulties of choice when shopping on an online shop, finding ourself wondering how could we maximize the potential value of the items that we bought given that we only have a certain budget? This paper tries to solve that problem by using the classic example of The Knapsack Problem, more specifically, the 0/1 Knapsack Problem.**

***Keywords*—Knapsack, Shopping List, Pattern Matching, Dynamic Programming**

## I. Introduction (*Heading 1*)

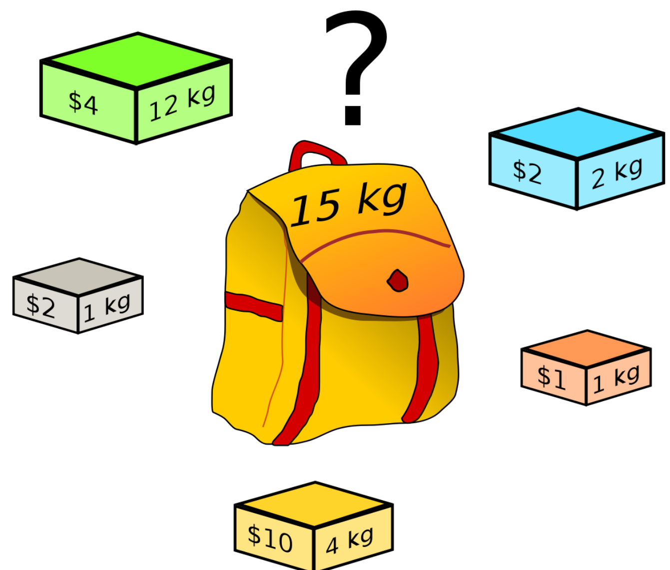
The Knapsack problem is one of the most classic combinatorial problems in computer science. To put it simply, this problem consists of many items, a bag, and a limit, and we need to find out what is the best combination within the limit that we can include in the bag with the most value possible. In this paper, we will try to apply this problem to recommending a shopping list to our user given the scenario : let's say we are running an online shop with a certain amount of items. Each item has their own popularity, which is how many people bought them and what are their reviews towards the product. We will try to create a program that will recommend the most optimal combination of items that the user can purchase given their budget, that is, how can the user buy as much popular item as they can within a certain budget?

Our application will also be supported by a bot that can parse and understand human language, using the functionality of regular expression string matching within javascript to provide users with ease of use.

## II. Theoretical Foundation

### *A. The Knapsack Problem*

The knapsack problem is a problem in combinatorial optimization: Given a set of items, each with a weight and a value, determine the number of each item to include in a collection so that the total weight is less than or equal to a given limit and the total value is as large as possible. It derives its name from the problem faced by someone who is constrained by a fixed-size knapsack and must fill it with the most valuable items. The problem often arises in resource allocation where the decision makers have to choose from a set of non-divisible projects or tasks under a fixed budget or time constraint, respectively.




The knapsack problem has been studied for more than a century, with early works dating as far back as 1897. The name "knapsack problem" dates back to the early works of the mathematician Tobias Dantzig (1884–1956), and refers to the commonplace problem of packing the most valuable or useful items without overloading the luggage.

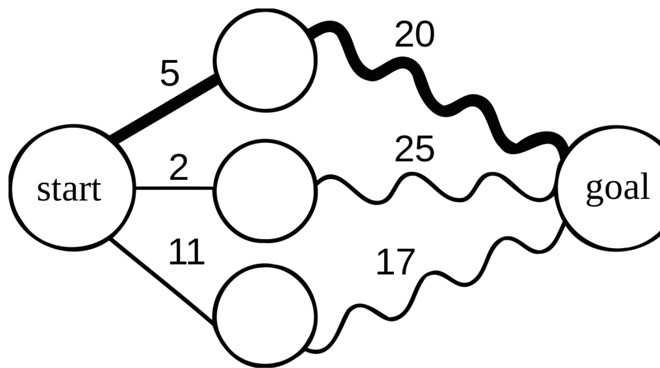
Our discussion will specifically focus on the 0/1 Knapsack Problem. Given weights and values of n items, put these items in a knapsack of capacity W to get the maximum total value in the knapsack. In other words, given two integer arrays val[0..n-1] and wt[0..n-1] which represent values and weights associated with n items respectively. Also given an integer W which represents knapsack capacity, find out the maximum value subset of val[] such that the sum of the weights of this subset is smaller than or equal to W. You cannot break an item, either pick the complete item or don't pick it (0-1 property).

### *B. Knapsack Problem with Dynamic Programming*

Dynamic Programming (DP) is an algorithmic technique for solving an optimization problem by breaking it down into simpler subproblems and utilizing the fact that the optimal solution to the overall problem depends upon the optimal solution to its subproblems.

Dynamic programming is both a mathematical optimization method and a computer programming method.

The method was developed by Richard Bellman in the 1950s and has found applications in numerous fields, from aerospace engineering to economics.




In both contexts it refers to simplifying a complicated problem by breaking it down into simpler sub-problems in a recursive manner. While some decision problems cannot be taken apart this way, decisions that span several points in time do often break apart recursively. Likewise, in computer science, if a problem can be solved optimally by breaking it into sub-problems and then recursively finding the optimal solutions to the sub-problems, then it is said to have optimal substructure.

If sub-problems can be nested recursively inside larger problems, so that dynamic programming methods are applicable, then there is a relation between the value of the larger problem and the values of the sub-problems.[1] In the optimization literature this relationship is called the Bellman equation.

In the Dynamic programming we will work considering the same cases as mentioned in the recursive approach. In a DP[][] table let's consider all the possible weights from '1' to 'W' as the columns and weights that can be kept as the rows.

The state DP[i][j] will denote the maximum value of 'j-weight' considering all values from '1 to ith'. So if we consider 'wi' (weight in 'ith' row) we can fill it in all columns which have 'weight values > wi'. Now two possibilities can take place:

Fill 'wi' in the given column.

Do not fill 'wi' in the given column.

Now we have to take a maximum of these two possibilities, formally if we do not fill 'ith' weight in 'jth' column then DP[i][j] state will be same as DP[i-1][j] but if we fill the weight, DP[i][j] will be equal to the value of 'wi'+ value of the column weighing 'j-wi' in the previous row.

### C. *Regular Expression*

A regular expression (shortened as regex or regexp;[1] also referred to as rational expression[2][3]) is a sequence of characters that specifies a search pattern. Usually such patterns are used by string-searching algorithms for "find" or "find and replace" operations on strings, or for input validation. It is a technique developed in theoretical computer science and formal language theory.

The concept arose in the 1950s when the American mathematician Stephen Cole Kleene formalized the description of a regular language. The concept came into common use with Unix text-processing utilities. Different syntaxes for writing regular expressions have existed since the 1980s, one being the POSIX standard and another, widely used, being the Perl syntax.

Regular expressions are used in search engines, search and replace dialogs of word processors and text editors, in text processing utilities such as sed and AWK and in lexical analysis. Many programming languages provide regex capabilities either built-in or via libraries, as it has uses in many situations.

### D. *Pattern Matching with Regular Expression*

In computer science, pattern matching is the act of checking a given sequence of tokens for the presence of the constituents of some pattern. In contrast to pattern recognition, the match usually has to be exact: "either it will or will not be a match." The patterns generally have the form of either sequences or tree structures. Uses of pattern matching include outputting the locations (if any) of a pattern within a token sequence, to output some component of the matched pattern, and to substitute the matching pattern with some other token sequence (i.e., search and replace).

Tree patterns are used in some programming languages as a general tool to process data based on its structure, e.g. C#,[1] F#,[2] Haskell, ML, Rust,[3] Scala,[4] Swift[5] and the symbolic mathematics language Mathematica have special syntax for expressing tree patterns and a language construct for conditional execution and value retrieval based on it.

Often it is possible to give alternative patterns that are tried one by one, which yields a powerful conditional programming construct. Pattern matching sometimes includes support for guards.Regular expressions are patterns used to match character combinations in strings. In JavaScript, regular expressions are also objects. These patterns are used with the exec() and test() methods of RegExp, and with the match(), matchAll(), replace(), replaceAll(), search(), and split() methods of String.

A regular expression pattern is composed of simple characters, such as /abc/, or a combination of simple and special characters, such as /ab*c/ or /Chapter (\d+)\.\d*/. The last example includes parentheses, which are used as a memory device. The match made with this part of the pattern is remembered for later use.

Regex book | Version History | Feedback | Blog

Options | Quick Reference

| | |
|---|---|
| . | Any character except newline. |
| \. | A period (and so on for *, \(, \\, etc.) |
| ^ | The start of the string. |
| $ | The end of the string. |
| \d,\w,\s | A digit, word character [A-Za-z0-9_], or whitespace. |
| \D,\W,\S | Anything except a digit, word character, or whitespace. |
| [abc] | Character a, b, or c. |
| [a-z] | a through z. |
| [^abc] | Any character except a, b, or c. |
| aa\|bb | Either aa or bb. |
| ? | Zero or one of the preceding element. |
| * | Zero or more of the preceding element. |
| + | One or more of the preceding element. |
| {n} | Exactly *n* of the preceding element. |
| {n,} | *n* or more of the preceding element. |
| {m,n} | Between *m* and *n* of the preceding element. |
| ??,*?,+?, {n}?, etc. | Same as above, but as few as possible. |
| (*expr*) | Capture *expr* for use with \1, etc. |
| (?:*expr*) | Non-capturing group. |
| (?=*expr*) | Followed by *expr*. |
| (?!*expr*) | Not followed by *expr*. |

Near-complete reference

## III. Implementation

This program will be implemented using javascript and typescript by creating a REST API using the help of Node.js and Express. The Idea is to have an endpoint that will receive an input message. The program will later parse this input message and try to extract the important bits of information that is included in the input message to later be forwarded into another function that will process the items. This endpoint will later returns the necessary data that the user requires, which is the list of items that has been processed to be the optimal shopping list, given a budget that the user has provided.

In the application, this process will be executed in the "/itemlist" endpoint of the REST API. This endpoint are stored in the "index.ts" file in our program.

```
app.get("/itemlist", (req, res) => {
    res.json(recomendation(req.body.message, data));
});
```

Most of our program logic will be handled by the module "utilities/item_recommendation.ts"

Our applications are using a user -friendly interface in which the user can just type what they're thinking and the program will try to recognize what the user is trying to do. Because of that, before we can process the data, we need to make sure that we are processing the right data that the user asked. This process is being executed by the functions getItemCategory and getBudget. These two functions will try to find what kind of item the user wants and what their budget. These to values will later be passed to the getBestValue function that will later process the optimized result.

```
function getItemCategory(inputMessage: any) {
    const findCategoryRegex = /Books|Furniture|Music|Electronics/gi;
    const itemCategory = inputMessage.match(findCategoryRegex);
    return itemCategory;
}

function getBudget(inputMessage: any) {
    let findBudget = /budget.?\d+/gi;
    let budget = inputMessage.match(findBudget)[0].split(" ")[1];
    // let index = findBudget.findIndex(budget)
    // budget = findBudget.;
    return budget;
}
```

getBestValue function are used to pick the best combination of item given certain itemCategories and budget that the user requires. This function works by using the idea of Dynamic programming, which is solving and memorizing subproblems for later use.

```
function getBestValue(items: item[], itemCategory: string[], budget: number) {
    const filteredItems = items.filter((item) =>
        itemCategory.includes(item.category.toLowerCase())
    );
    let optimalCombination: number[][] = [];
    for (let i = 0; i < filteredItems.length; i++) {
        const { id, popularity, price } = filteredItems[i];
        optimalCombination.push([]);
        for (let currentTotal = 0; currentTotal * 1000 < budget;
currentTotal++) {
            if (i == 0 || currentTotal == 0) {
                optimalCombination[i].push(0);
            } else if (currentTotal < Math.round(price /
1000)) {

const topPopularity = optimalCombination[i - 1][currentTotal];

optimalCombination[i].push(topPopularity);
            }
 else {

const prevPrice = currentTotal - Math.round(price / 1000);

const prevPopularity = optimalCombination[i - 1][prevPrice];

const topPopularity = optimalCombination[i - 1][currentTotal];

const bestValue = max(topPopularity, prevPopularity + popularity);
```

```
optimalCombination[i].push(bestValue);
                }
            }
        }
        let recomendedItems: item[] = [];
        let currentCapacity = optimalCombination[1].length - 1;
        // console.log(currentCapacity);
        // console.log(optimalCombination.length);
        for (let i = optimalCombination.length - 1; i > 0; i--) {
            if (currentCapacity <= 0) {
                break;
            }

const current = optimalCombination[i][currentCapacity];

const topValue = optimalCombination[i - 1][currentCapacity];
            if (current != topValue) {

recomendedItems.push(filteredItems[i]);

const itemCapacity = Math.round(filteredItems[i].price / 1000);

currentCapacity -= itemCapacity;
            }
        }
        return recomendedItems;
}
```

This recommendation function is used as a main hub that connects the "utilities/item_recommendation.ts" module to the "index.ts" file that controls our REST API.

```
export function recomendation(inputMessage: string, items: item[]) {
        const budget = getBudget(inputMessage);
        const itemCategory = getItemCategory(inputMessage);
        const recommendedItems = getBestValue(items, itemCategory, budget);
        return { message: inputMessage, items: recommendedItems };
}
```

## IV. Test Cases

The first case is to test what the program will return if we give a budget that is way below any item that is available in the database. As expected, the program returns an empty array.

JSON Auth Query Header 1 Docs


{
  "message" : "Recommends me books to buy with the budget 3000 Rupiah"
}


Beautify JSON

200 OK 20.3 ms 79 B

Preview Header 6 Cookie Timeline


{
  "message": "Recommends me books to buy with the budget 3000 Rupiah",
  "items": []
}


The next test case are to retrieves book items with a reasonable budget. This time, the application are returning the books, which are the optimal combination of books that the user should get if they want to maximize buying as much popular books as they can.


{
  "message" : "Recommends me books to buy with the budget 20000 Rupiah"
}


Beautify JSON

200 OK 35.2 ms 494 B

Preview Header 6 Cookie Timeline


{
  "message": "Recommends me books to buy with the budget 20000 Rupiah",
  "items": [
    {
      "id": 19,
      "name": "Fresh",
      "popularity": 7,
      "category": "Books",
      "price": 6600,
      "description": "Advanced mission-critical attitude"
    },
    {
      "id": 11,
      "name": "In the Company of Men",
      "popularity": 6.3,
      "category": "Books",
      "price": 5600,
      "description": "Up-sized motivating workforce"
    },
    {
      "id": 7,
      "name": "Manrape (Män kan inte våldtas) ",
      "popularity": 1.2,
      "category": "Books",
      "price": 4600,
      "description": "Ameliorated tertiary open architecture"
    }
  ]


The applications are not only working with one category of item, therefore we can test the case where we asked our application to retrieve the recommended item combination if we want to include books, furniture, and electronics in our shopping list.

```
JSON  Auth  Query  Header 1  Docs
{
  "message" : "Recommends me books, furniture, electronics to buy with the budget 450000 Rupiah"
}
Beautify JSON
200 OK  27 ms  1911 B
Preview  Header 6  Cookie  Timeline
{
  "message": "Recommends me books, furniture, electronics to buy with the budget 450000 Rupiah",
  "items": [
    {
      "id": 24,
      "name": "Bola Lampu",
      "popularity": 8.4,
      "category": "Electronics",
      "price": 9280,
      "description": null
    },
    {
      "id": 23,
      "name": "Meja 2",
      "popularity": 9.2,
      "category": "Furniture",
      "price": 190700,
      "description": null
    },
    {
      "id": 19,
      "name": "Fresh",
      "popularity": 7,
      "category": "Books",
      "price": 6600,
      "description": "Advanced mission-critical attitude"
    },
```


In case that the user inputs a category that we do not have in our shop, we will simply return an empty array.


```
{
  "message" : "Recommends me breakfast to buy with the budget 20000 Rupiah"
}
Beautify JSON
200 OK  65.4 ms  84 B
Preview  Header 6  Cookie  Timeline
{
  "message": "Recommends me breakfast to buy with the budget 20000 Rupiah",
  "items": []
}
```


## Video Link at Youtube

https://youtu.be/fiO9hoHVSOw

## Github Repository

https://github.com/eruds/Shop-Item-Recomendation

## Acknowledgment


In this section I would like to express my gratitude towards God, my friends, and my family for supporting me all the way up to this point of my life where I finish this paper. I would also like to express my gratitude to Ir. Rila Mandala,M.Eng.,Ph.D. as my lecturer and also to the entire ITB informatics study lecturers. It is thanks to their efforts that I am able to finish this paper and understand the various algorithm strategies commonly used in programming.Lastly I would like to apologize for any mistakes that occured in the process of making this paper.


## References


[1] https://www.geeksforgeeks.org/0-1-knapsack-problem-dp-10/

[2] https://en.wikipedia.org/wiki/Regular_expression

[3] https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Program-Dinamis-2020-Bagian1.pdf

[4] https://en.wikipedia.org/wiki/Pattern_matching

[5] https://en.wikipedia.org/wiki/Knapsack_problem

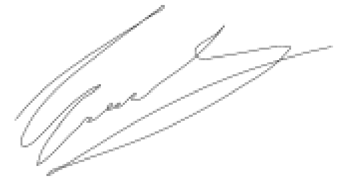
[6] https://www.educative.io/courses/grokking-dynamic-programming-patterns-for-coding-interviews/m2G1pAq0OO0

[7] https://en.wikipedia.org/wiki/Dynamic_programming


## 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.

Medan, 11 Mei 2021



Mahameru Ds - 13519014