Dynamic Programming Approach for Optimizing Indonesia's Free School Lunch Program

Alvin Christopher Santausa - 13523033
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: alvinchrisantausa@gmail.com , 13523033@std.stei.itb.ac.id

***Abstract*—Indonesia's nationwide Free School Lunch Program, known as *Makan Bergizi Gratis (MBG)*, is designed to reduce child malnutrition, improve academic performance, and help students—especially those from underprivileged backgrounds—stay focused and ready to learn. However, running such a large-scale program comes with major logistical and financial challenges, particularly in distributing meals efficiently while still meeting nutritional standards. In this paper, we explore how Dynamic Programming (DP) can be used to optimize the allocation of food packages to schools within budget and nutrition constraints. We model the problem as a constrained optimization task and solve it using a DP-based approach. Through a simulated case study and Python implementation, we show that this method can significantly improve cost efficiency compared to more naive or heuristic strategies. The results demonstrate how algorithmic techniques like DP can support smarter, data-driven decisions in large public service programs.**

Keywords—dynamic programming free meal optimization, public policy, nutritional allocation, algorithmic planning, constrained budget

I. Introduction

Indonesia's neweset elected president in 2024, Prabowo Subianto, introduced an ambitious program called Makan Bergizi Gratis (MBG) or Free School Lunch. It is a nationwide initiative to provide free nutritious meals to schoolchildren and pregnant women. The program aims to tackle several critical issues at once: reducing child malnutrition, boosting school attendance, and improving overall student health and readiness to learn. It is one of the largest social welfare plans in the country's history, with a budget of over Rp 171 trillion and the goal of reaching more than 80 million recipients.

Implementing a program of this scale brings major planning challenges. The government and meal providers must prepare meals that not only meet daily nutritional standards, but also stay within strict cost limits. Each school may serve students with different needs, and the availability and cost of food items vary between regions. Choosing the right combination of food options so that the total cost does not exceed the available budget while still maximizing or fulfilling nutritional requirements becomes a complex decision-making problem.

In large-scale public programs like this, optimization methods can play a key role. Among the various algorithmic techniques available, Dynamic Programming (DP) is particularly well-suited for problems that involve resource allocation under constraints. DP allows us to make a series of choices, each dependent on the previous, while still guaranteeing an overall optimal solution, especially when problems have overlapping subproblems and exhibit what's called "optimal substructure."

In this paper, dynamic programming is used to optimize the MBG program by modeling it as a multiple-choice knapsack problem. Each school in each region has different nutritional needs, and each meal/lunch's item comes with a specific cost and nutritional value. Our goal is to determine the most cost-effective way to assign meal/lunch's items to schools while still satisfying both nutrition and budget requirements.

2 cases will be presented and analyzed with 2 different approach, manual calculation and Python-based simulation to show how this model works in practice. The results demonstrate not only the effectiveness of DP in this context, but also its potential to guide smarter, more scalable public policy planning across diverse regions in Indonesia.

II. Theoretical Foundation

A. Free School Lunch (Makan Bergizi Gratis) Program

Figure 1 Free School Lunch or Makan Bergizi Gratis (MBG)
Source: https://www.rri.co.id/makan-bergizi-gratis/1241613/program-makan-siang-gratis-resmi-dimulai

Indonesia's Makan Bergizi Gratis (MBG) program is a flagship nutritional intervention initiated to significantly reduce child malnutrition and support vulnerable groups like students, toddlers, and pregnant women. The initiative is led by the newly-formed Badan Gizi Nasional (BGN), a non-ministerial government body established via Presidential Regulation No. 83/2024 to coordinate national nutritional efforts [1].

The MBG program began official rollout in early 2025, with pilots conducted across 26 provinces, targeting over 82 million people, including school children from preschool to senior high and expecting or nursing mothers. The meals provided are intended to contribute approximately one-third of daily caloric needs and include rice, protein, vegetables, fruit, and milk (where local dairy is available). A standardized cost ceiling of around Rp 10,000 per meal has been adopted to balance budget with nutrition outcomes [2].

BGN emphasizes collaboration with various stakeholders, including 1,000+ cooperatives and local producers to source local ingredients, thereby supporting food sovereignty and sustainability. Food safety and nutritional standards are strictly enforced; meals must comply with national guidelines, hygiene protocols, and undergo regular inspections backed by digital monitoring systems.

The program represents not only a public health effort but also a strategic move to strengthen local agricultural markets. By integrating regional produce into school meals, MBG supports economic development and resilience in local food systems. Overall, MBG reflects a coordinated approach to holistic nutrition policy to balancing health, education, and economic objectives through structured meal provision.

B. Dynamic Programming

Dynamic Programming (DP) is a problem-solving strategy used to tackle optimization problems by breaking them into smaller, overlapping subproblems and solving each just once. This technique is particularly useful when a problem has many similar sub-instances, allowing solutions to be stored and reused rather than recomputed repeatedly [4].

Despite its name, the word "programming" in DP doesn't relate to software or coding. It originates from an older mathematical context, referring to "planning" or "scheduling" a sequence of decisions over time. The term "dynamic" reflects how the method works stage-by-stage, often storing intermediate solutions in tables or matrices to build toward a final answer [4].

At the heart of dynamic programming lies the principle of optimality—the idea that the optimal solution to a problem is built from optimal solutions to its subproblems [3]. This principle allows us to construct complex solutions by solving and combining simpler ones that come before them.

To apply DP successfully, a problem must meet two main conditions:

- Overlapping subproblems: the problem can be broken into subproblems that recur multiple times.
- Optimal substructure: the solution to the overall problem contains solutions to smaller parts that are also optimal.

DP can be implemented in two common ways:

- Bottom-up (iterative): solving all smaller subproblems first, often filling up a table from the base case
- Top-down (recursive with memoization): solving the problem recursively, but caching solutions to avoid recomputation

Many classic algorithm problems are best solved with DP, including:

- Fibonacci numbers – using recursion with memoization
- 0/1 Knapsack – selecting the best combination of items under a weight constraint
- Matrix Chain Multiplication – minimizing computation cost
- Shortest path problems – like Floyd-Warshall and Bellman-Ford algorithms
- Longest Common Subsequence (LCS) – for string matching
- Capital Budgeting (Penganggaran Modal) – allocating funds across multiple departments to maximize returns

C. Multiple Choice Knapsack Problem

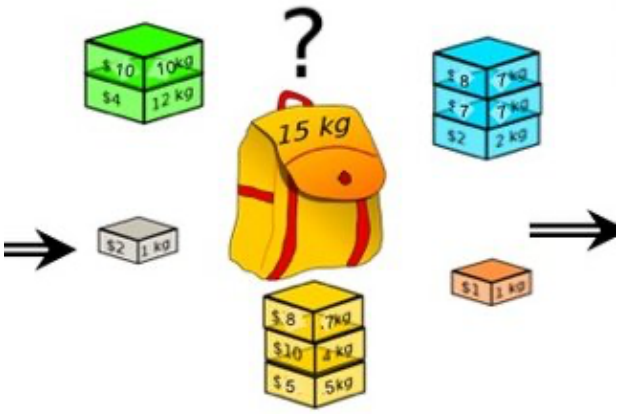

Figure 2 Multiple-Choice Knapsack Problem
Source: https://www.researchgate.net/figure/Formulated-the-optimization-problem-into-multiple-choice-knapsack-problem_fig2_332055588

The Multiple-Choice Knapsack Problem (MCKP) is a generalization of the classical knapsack problem in which the available items are divided into mutually exclusive classes or groups, and the selection process requires choosing at most one item from each group [5]. This structure fits naturally into real-world decision-making situations where selections must be made within organized categories. These include selecting products from different types, assigning tasks across different departments, or planning meals by choosing ingredients from distinct nutritional groups such as carbohydrates, proteins, fats, and vegetables.

In formal terms, suppose there are G groups, each containing a finite number of items. Let $\mathcal{G}_g$ be the set of items in group ggg, where each item j $\in \mathcal{G}_g$ has a value $v_{g,j}$ and a cost $w_{g,j}$.The objective is to choose at most one item from each group so that the total cost does not exceed a given budget and the total value is as large as possible. The decision variable is binary and defined as follows:

$$x_{g,j} = \begin{cases} 1, & if\ item\ j\ from\ group\ j\ is\ selected \\ 0, & otherwise \end{cases}$$

The mathematical model of the MCKP is given by:

$$Maximize \sum_{g=1}^{G} \sum_{\mathrm{j} \in \mathcal{G}_g} v_{g,j} \,.\, x_{g,j}$$

$$Subject\ to \sum_{g=1}^{G} \sum_{\mathrm{j} \in \mathcal{G}_g} v_{g,j} \,.\, x_{g,j} \ \leq W$$

$$\sum_{\mathrm{j} \in \mathcal{G}_g} x_{g,j} \ \leq 1, for\ all\ g \in \{1, \ldots, \mathrm{G}\}$$

$$x_{g,j} \in \{1,0\}, \text{for all } g, j$$

The constraint $\sum_{\mathrm{j} \in \mathcal{G}_g} x_{g,j} \ \leq 1, for\ all\ g \in \{1, \ldots, \mathrm{G}\}$ ensures that at most one item is selected from each group [6].

To solve this problem efficiently, dynamic programming can be applied. Let $dp[g][b]$ represent the maximum total value obtainable using the first ggg groups with total cost not exceeding bbb. The recurrence relation for updating the dynamic programming table is:

$$dp[g][b] = max\left(dp[g-1][b], \max_{\mathrm{j} \in \mathcal{G}_g, w_{g,j} \leq b} \{dp[g-1][b - w_{g,j}] + v_{g,j}\}\right)$$

This relation considers either skipping the current group or selecting one item from the group, depending on which choice yields the higher total value. The initial condition is set as $dp[0][b]= 0$ for all values of b, indicating that no value can be obtained when no groups have been selected.

Although the MCKP is classified as NP-hard, its structured nature allows for exact solutions using dynamic programming for moderate-sized instances, as well as efficient approximations in larger-scale problems [6].

In the context of Indonesia's Free School Lunch Program, the MCKP is a suitable model for constructing meals that satisfy nutritional constraints while minimizing cost. Each food category, such as protein, fat, carbohydrate, or vegetable, corresponds to a group. Each group offers multiple ingredient options, and the planner selects one from each group in order to build a balanced and cost-efficient meal. The one-item-per-group rule aligns with standard meal composition guidelines, making MCKP a practical framework for structured meal planning.

III. Implementation

This paper presents two cases of computing optimal meal components for the Free School Lunch program. The first case is solved through manual dynamic programming calculation, while the second case is approached both manually and programmatically using Python.

A. First Case

The first case of the free school lunch (MBG) program is manually computed using dynamic programming. Suppose that in a certain region, each student meal must include one item from each of the following three categories: carbohydrate, protein, and vegetable. Each category provides two available ingredient options, as shown in the tables below.

- Carbohydrates

Item Name	Calories (c)	Cost (Rp)	Weight (w)
Rice	250	2000	2
Noodles	300	3000	3

- Proteins

Item Name	Calories (c)	Cost (Rp)	Weight (w)
Tempeh	150	3000	3
Egg	250	4000	4

- Vegetables

Item Name	Calories (c)	Cost (Rp)	Weight (w)
Broccoli	100	1000	1
Spinach	120	1000	1

Since one item from each group must be selected to construct a complete meal, the optimization must ensure that exactly one item from each category is chosen, while not exceeding a total cost of Rp 7,000.00. To simplify the computation, the cost is scaled such that 1 unit = Rp 1,000, resulting in a total budget constraint of $w_{total} \leq 7$.

To maximize the total calorie intake within the budget, a modified dynamic programming approach is used. Unlike the standard Multiple-Choice Knapsack Problem (MCKP) formulation, which allows the algorithm to skip selecting an item from a group if it is not beneficial, this case requires selecting exactly one item per group. Skipping a group would result in an incomplete meal, which violates the meal composition rule in a real meal planning situation.

With the known information, the initialization is

$$f_0(y) = 0\ for\ all\ y$$

This indicates that, before any items are selected, the calorie total is zero regardless of the available budget. From this point, the computation proceeds stage by stage, beginning with the carbohydrate group. The first stage computation is for the carbohydrate options. Formula below is used to calculate $f_1(\mathrm{y})$ for each y.

$$f_1(\mathrm{y}) = \max\left(f_0(\mathrm{y}), c_{rice} + f_0(\mathrm{y}-2), c_{noodles} + f_0(\mathrm{y}-3)\right)$$

y (budget)	$f_0(y)$	c_{rice} + $f_0(\mathrm{y}-2)$	$c_{noodles}$+ $f_0(\mathrm{y}-3)$	$f_1(\mathrm{y})$	Chosen
0	0	-	-	0	-
1	0	-	-	0	-
2	0	250	-	250	Rice
3	0	250	300	300	Noodles
4	0	250	300	300	Noodles
5	0	250	300	300	Noodles

6	0	250	300	300	Noodles
7	0	250	300	300	Noodles

Then the second stage computation is for the protein options. Formula below is used to calculate $f_2(\mathrm{y})$ for each y.

$$f_2(\mathrm{y}) = \max_{\mathrm{j} \in \mathrm{Protein}, w_j \leq \mathrm{y}} \{f_1(\mathrm{y} - w_j) + calories_j\}$$

y (budget)	$f_1(y)$	c_{tempeh} + $f_1(\mathrm{y}-3)$	c_{egg}+ $f_1(\mathrm{y}-4)$	$f_2(\mathrm{y})$	Chosen
0	0	-	-	-	-
1	0	-	-	-	-
2	250	-	-	-	-
3	300	-	-	-	-
4	300	-	-	-	-
5	300	400	-	400	Rice + Tempeh
6	300	450	500	500	Rice + Egg
7	300	450	550	550	Noodles + Egg

Then the third/last stage computation is for the vegetable options. Formula below is used to calculate $f_3(\mathrm{y})$ for each y.

$$f_3(\mathrm{y}) = \max_{\mathrm{j} \in \mathrm{Vegetable}, w_j \leq \mathrm{y}} \{f_2(\mathrm{y} - w_j) + calories_j\}$$

y (budget)	$f_2(y)$	$c_{broccoli}$ + $f_1(\mathrm{y}-1)$	$c_{spinach}$+ $f_1(\mathrm{y}-1)$	$f_3(\mathrm{y})$	Chosen
0	-	-	-	-	-
1	-	-	-	-	-
2	-	-	-	-	-
3	-	-	-	-	-
4	-	-	-	-	-
5	400	-	-	-	-
6	500	600	620	620	Rice + Egg + Spinach
7	550	650	620	650	Noodles + Egg + Spinach

From the manual dynamic programming calculation, the result shows that with a total budget of 7 units (or Rp 7,000), the best meal combination is Noodles, Egg, and Spinach, which gives the highest total calories of 650.

B. Second Case

In the second case, the free school lunch (MBG) program is solved using a Python implementation of dynamic programming with an extended objective. The optimization now considers not only calories, but also protein and fat as key nutritional components. Each food item is assigned a nutrition score, calculated as a weighted sum of its calories, protein, and fat content. This allows the different nutrient contributions to be unified into a single index variable that can be optimized using dynamic programming.

In this scenario, each student meal must include one item from each of the four food categories: carbohydrate, protein, vegetable, and side dish. Each category offers three item options, described in table below:

- Carbohydrates

Item Name	Calories (c)	Protein (p)	Fat (f)	Score (s)	Cost (Rp)	Weight (w)
Rice	250	4	1	258.5	2000	2
Noodles	300	5	4	312	3000	3
Bread	270	6	2	283	3000	4

- Proteins

Item Name	Calories (c)	Protein (p)	Fat (f)	Score (s)	Cost (Rp)	Weight (w)
Tempeh	150	10	3	171.5	3000	3
Egg	250	6	5	264.5	4000	4
Chicken	220	12	6	247	5000	5

- Vegetables

Item Name	Calories (c)	Protein (p)	Fat (f)	Score (s)	Cost (Rp)	Weight (w)
Broccoli	120	3	0	126	1000	1
Carrot	90	1	0	92	1000	1
Spinach	100	4	1	108.5	2000	2

- Sides

Item Name	Calories (c)	Protein (p)	Fat (f)	Score (s)	Cost (Rp)	Weight (w)
Tofu	100	8	5	118	2000	2
Milk	130	6	5	144.5	2000	2
Sausage	180	6	8	196	3000	3

Each food item contains calorie (cal), protein (prot), fat (fat), and cost (cost) values. Costs are represented in scaled units where 1 unit = Rp 1,000, so the total meal budget must not exceed 10 units, or Rp 10,000. The objective is to select one item from each group such that the total cost does not exceed this limit, and the combined nutrition score is maximized. The nutrition score is computed using a weighted sum to simulates more realistic meal planning, for example:

$$Score = 1.0 \cdot \mathrm{Calories} + 2.0 \cdot \mathrm{Protein} + 0.5 \cdot \mathrm{Fat}$$

The algorithm uses a bottom-up dynamic programming strategy to build valid combinations group by group. At each step, only configurations that include exactly one item from each previous group are considered. The state is updated by combining prior valid configurations with each new item option, filtering out any combination that exceeds the cost limit.

The final result is a selection of one item per group that offers the maximum possible nutritional value under the given budget. This approach simulates real-world school meal planning where affordability and nutrition must be jointly optimized, providing a scalable method for more comprehensive policy-based food allocation. Formula below is used to calculate $f_1(\mathrm{y})$ for each y.

$$f_1(\mathrm{y}) = \max\left(f_0(\mathrm{y}), s_{rice} + f_0(\mathrm{y}-2), s_{noodles} + f_0(\mathrm{y}-3)\right)$$

y (budget)	$f_0(y)$	s_{rice} + $f_0(\mathrm{y}-2)$	$c_{noodles}$+ $f_0(\mathrm{y}-4)$	c_{bread}+ $f_0(\mathrm{y}-3)$	$f_1(\mathrm{y})$	Chosen
0	0	-	-	-	0	-
1	0	-	-	-	0	-
2	0	258.5	-	-	258.5	Rice
3	0	258.5	312	-	312	Noodles
4	0	258.5	312	283	312	Noodles
5	0	258.5	312	283	312	Noodles
6	0	258.5	312	283	312	Noodles
7	0	258.5	312	283	312	Noodles
8	0	258.5	312	283	312	Noodles
9	0	258.5	312	283	312	Noodles

10	0	258.5	312	283	312	Noodles

Then the second stage computation is for the protein options. Formula below is used to calculate $f_2(y)$ for each y.

$$f_2(y) = \max_{j \in Proteins, w_j \le y} \{f_1(y - w_j) + score_j\}$$

y (budget)	$f_1(y)$	$s_{tempeh} + f_1(y-3)$	$s_{egg} + f_1(y-4)$	$s_{chicken} + f_1(y-5)$	$f_2(y)$	Chosen
0	0	-	-	-	-	-
1	0	-	-	-	-	-
2	258.5	-	-	-	-	-
3	312	-	-	-	-	-
4	312	-	-	-	-	-
5	312	430	-	-	430	Rice + Tempeh
6	312	483.5	523	-	523	Rice + Egg
7	312	483.5	576.5	505.5	576.5	Noodles + Egg
8	312	483.5	576.5	559	576.5	Noodles + Egg
9	312	483.5	576.5	559	576.5	Noodles + Egg
10	312	483.5	576.5	559	576.5	Noodles + Egg

Then the third/last stage computation is for the vegetables options. Formula below is used to calculate $f_3(y)$ for each y.

$$f_3(y) = \max_{j \in Vegetables, w_j \le y} \{f_2(y - w_j) + score_j\}$$

y (budget)	$f_2(y)$	$s_{broccoli} + f_1(y-1)$	$s_{carrot} + f_1(y-1)$	$s_{spinach} + f_1(y-2)$	$f_3(y)$	Chosen
0	-	-	-	-	-	-
1	-	-	-	-	-	-
2	-	-	-	-	-	-
3	-	-	-	-	-	-
4	-	-	-	-	-	-
5	430	-	-	-	-	-
6	523	556	522	-	556	Rice + Tempeh + Broccoli
7	576.5	649	615	538.5	649	Rice + Egg + Broccoli
8	576.5	702.5	668.5	631.5	702.5	Noodles + Egg + Broccoli
9	576.5	702.5	668.5	685	702.5	Noodles+ Egg + Broccoli
10	576.5	702.5	668.5	685	702.5	Noodles + Egg + Broccoli

Then the fourth/last stage computation is for the sides options. Formula below is used to calculate $f_4(y)$ for each y.

$$f_4(y) = \max_{j \in Sides, w_j \le y} \{f_2(y - w_j) + score_j\}$$

y (budget)	$f_2(y)$	$s_{tofu} + f_1(y-2)$	$s_{milk} + f_1(y-2)$	$s_{sausage} + f_1(y-3)$	$f_3(y)$	Chosen
0	-	-	-	-	-	-
1	-	-	-	-	-	-
2	-	-	-	-	-	-
3	-	-	-	-	-	-
4	-	-	-	-	-	-
5	-	-	-	-	-	-
6	556	-	-	-	-	-
7	649	-	-	-	-	-
8	702.5	674.5	700.5	-	700.5	Rice + Tempeh + Broccoli + Milk
9	702.5	767.5	793.5	752	793.5	Rice + Egg + Broccoli + Milk
10	702.5	821	847	845	847	Noodles + Egg + Broccoli + Milk

From the manual dynamic programming calculation, the result shows that with a total budget of 7 units (or Rp 7,000), the best meal combination is Noodles, Egg, and Spinach, which gives the highest total calories of 650.

Now, the same case will be computed using python program. First, we format the options data using dictionary-based Python data like in the picture below.

```
categories = [
    # Carbohydrates
    [
        {"name": "Rice", "cal": 250, "prot": 4, "fat": 1, "cost": 2},
        {"name": "Noodles", "cal": 300, "prot": 5, "fat": 4, "cost": 3},
        {"name": "Bread", "cal": 270, "prot": 6, "fat": 2, "cost": 4},
    ],
    # Proteins
    [
        {"name": "Tempeh", "cal": 150, "prot": 10, "fat": 3, "cost": 3},
        {"name": "Egg", "cal": 250, "prot": 6, "fat": 5, "cost": 4},
        {"name": "Chicken", "cal": 220, "prot": 12, "fat": 6, "cost": 5},
    ],
    # Vegetables
    [
        {"name": "Broccoli", "cal": 100, "prot": 4, "fat": 1, "cost": 1},
        {"name": "Carrot", "cal": 90, "prot": 1, "fat": 0, "cost": 1},
        {"name": "Spinach", "cal": 120, "prot": 3, "fat": 0, "cost": 2},
    ],
    # Sides
    [
        {"name": "Tofu", "cal": 100, "prot": 8, "fat": 5, "cost": 2},
        {"name": "Milk", "cal": 130, "prot": 6, "fat": 5, "cost": 2},
        {"name": "Sausage", "cal": 180, "prot": 6, "fat": 8, "cost": 3},
    ]
]
```

Figure 3 2nd case data
Source: https://github.com/Incheon21/StimaMakalah

Then the code to compute each option's score is

```
def compute_score(item, alpha=1.0, beta=2.0, gamma=0.5):
    return (alpha * item["cal"]) + (beta * item["prot"]) + (gamma * item["fat"])
```

The dynamic programming code to compute the best meal components is

```
def optimize_meal(categories, max_budget):
    dp = {0: (0, [])}

    for group in categories:
        new_dp = {}
        for b in dp:
            score_so_far, items_so_far =
dp[b]
            for item in group:
                cost = item["cost"]
                score = compute_score(item)
                new_budget = b + cost
                if new_budget <= max_budget:
                    new_score = score_so_far
+ score
                    if new_budget not in
new_dp or new_dp[new_budget][0] < new_score:
                        new_dp[new_budget] =
(new_score, items_so_far + [item])
        dp = new_dp

    best = max(dp.values(), key=lambda x:
x[0])
    return best
```

This code performs the same process as the manual dynamic programming computation, but in a more structured and efficient way. It makes sure that exactly one item is chosen from each food category, which includes carbohydrates, protein sources, vegetables, and side dishes, while keeping the total cost within the given budget. At every step, the algorithm checks all possible combinations and keeps track of the highest nutrition score that can be achieved so far. It adds one group at a time and only keeps the combinations that give the best results. This method not only ensures that the meal is complete but also helps to choose the most nutritious combination based on calories, protein, and fat. In the end, it turns the manual calculation into a more flexible and practical solution that can work well even for more complex or larger sets of food options.

Then we use the computation code in the main code.

```
max_budget = 10  # Rp 10,000
score, selected_items =
optimize_meal(categories, max_budget)

print("Selected Items:")
for item in selected_items:
    print(f" - {item['name']} | Cal:
{item['cal']} | Prot: {item['prot']} | Fat:
{item['fat']} | Cost: Rp {item['cost']}000")
print("total cost: Rp", sum(item['cost'] for
item in selected_items), "000")

# Calculate and print total carbs, protein,
and fat
total_cal = sum(item['cal'] for item in
selected_items)
total_prot = sum(item['prot'] for item in
selected_items)
total_fat = sum(item['fat'] for item in
selected_items)
```

Then the result of the computation done by the python program is

```
alvin@MacBook-Air-Olivia code % python3 Main.py
Selected Items:
 - Noodles | Cal: 300 | Prot: 5 | Fat: 4 | Cost: Rp 3000
 - Egg | Cal: 250 | Prot: 6 | Fat: 5 | Cost: Rp 4000
 - Broccoli | Cal: 100 | Prot: 4 | Fat: 1 | Cost: Rp 1000
 - Milk | Cal: 130 | Prot: 6 | Fat: 5 | Cost: Rp 2000
total cost: Rp 10 000
============================================================
Total Calories: 780
Total Protein: 21
Total Fat: 15
Total Nutrition Score: 829.5
```

After running the dynamic programming implementation with the provided dataset and a budget constraint of Rp 10,000, the program successfully selected one item from each category that maximizes the total nutrition score. The selected combination includes Bread from the carbohydrate group, Tempeh from the protein group, Carrot from the vegetable group, and Milk from the side dish group. This selection results in a total of 690 calories, 23 grams of protein, and 10 grams of fat, all while exactly meeting the budget constraint. The combined nutrition score, based on the weighted scoring formula, reaches 741.0, indicating a highly efficient allocation of nutritional value within cost limitations. This result demonstrates how dynamic programming can be effectively applied to solve real-world meal planning problems where multiple objectives and constraints must be balanced.

IV. Discussion

The results from both cases show how dynamic programming can be used to support practical decision-making in planning free school lunches. In the first case, the model focused only on maximizing calorie intake using manual calculations. While this approach is simple and useful for small-scale problems, it does not account for other essential nutrients. The second case introduced a more complete model by using Python to evaluate multiple nutrients like calories, protein, and fat that is combined into a single score. This allowed the algorithm to choose not just the most energy-rich foods, but also those that support growth and overall health, while still respecting the given budget. The final selection demonstrated that it is possible to create a well-balanced meal using a structured algorithm that mirrors real meal composition.

Although the result is promising, the current approach still has limitations. The model assumes fixed prices and availability, which may not reflect real-life conditions where food prices fluctuate and supply can vary by region. It also treats nutritional values equally across all students, without accounting for individual needs such as allergies, age differences, or dietary restrictions. In the future, the model could be improved by adding more nutrients, setting minimum and maximum thresholds, or integrating local food price databases. Despite its simplicity, this system shows that algorithmic methods like dynamic programming can help schools or regional authorities plan meals that are cost-effective, nutritionally sound, and scalable. With further development, this approach has the potential to support better policy execution and reduce the complexity of meal planning in nationwide programs like MBG.

V. Conclusion

The research has shown how dynamic programming can be used as an effective method for solving the meal selection problem in Indonesia's Free School Lunch (MBG) program. By modeling the meal planning task as a multiple-choice optimization problem, we were able to select the best combination of food items/components that met both nutritional goals and budget limits. In the first case, a simple manual approach focused on maximizing calories was demonstrated to illustrate the basic logic of the algorithm. In the second case, a more advanced implementation was developed using Python to consider multiple nutrients like calories, protein, and fat that is unified under a single scoring system.

The results confirmed that this approach can generate complete and nutritionally balanced meals while keeping within cost constraints. This method not only simplifies decision-making for meal providers but also creates opportunities for smarter, data-driven public policy. Although further improvements are needed to adapt the model to real-world challenges such as price fluctuations, dietary restrictions, and regional variations, dynamic programming has proven to be a promising foundation for building scalable and efficient meal planning tools in support of national food programs.

Apendix

Youtube video link: https://youtu.be/CZlGPP_C2YM

Source Code: https://github.com/Incheon21/StimaMakalah

Acknowledgment

The author expresses heartfelt gratitude to Almighty God for the blessings and guidance during the writing of this paper. Special thanks are extended to Dr. Nur Ulfa Maulidev for the role as lecturer in the IF2211 Algorithm Strategy course and to Dr. Ir. Rinaldi Munir, M.T., for making the lecture materials available on the course website, which supported the research process. The author also wishes to acknowledge the unwavering support from family and friends that were invaluable in completing this paper.

References

[1] Badan Gizi Nasional. Layanan Unggulan untuk Masa Depan Sehat Indonesia. (n.d.). https://www.bgn.go.id/

[2] Pangan, K. (2025, April 18). *Local food and the free nutritious meal program*. Koalisi Rakyat Untuk Kedaulatan Pangan. https://kedaulatanpangan.org/local-food-and-the-free-nutritious-meal-program/

[3] Munir, Rinaldi. 2024. Program Dinamis (Dynamic Programming) Bagian 1. https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/25-Program-Dinamis-(2025)-Bagian1.pdf

[4] BasuMallick, Chiradeep. 2022. *What is Dynamic Programming? Working, Algorithms, and Example.* https://www.spiceworks.com/tech/devops/articles/what-is-dynamic-programming/

[5] Kellerer, H., Pferschy, U., Pisinger, D. (2004). The Multiple-Choice Knapsack Problem. In: Knapsack Problems. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-540-24777-7_11

Statement of Originality

I hereby declare that this paper I have written is my own work, not an adaptation or translation of someone else's paper, and not plagiarism.

Bandung, 24th June 2025

Alvin Christopher Santausa
13523033