Math Extended Essay

Maximize the Value of a knapsack

How to maximize the value of a knapsack given the knapsack's weight capacity and a set of items, each with a weight and a value?

Word count: 3810

Table of Contents

1. Introduction

A common challenge for programming competitions is the knapsack problem – “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 while maximizing the total value” (Knapsack problem - 2021). The problem’s simple conception and its complexity in real analysis fascinated me, so I choose this type of problem to investigate. My main focus for this essay would be a proposal of methods to solve the problems, and optimization processes to make my methods more efficient, and testing them using randomly generated data.

2. State Research Question

2.1. Definition

To describe the knapsack problem in plain language:

“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.” (Knapsack problem - 2021)

To describe the knapsack problem in mathematical terms:

Given n items, numbered from 1 up to n. The ith item has a weight of w_i, a value of v_i and a maximum amount of m_i; x_i is the number of the $i$rh item to include in a knapsack of weight capacity W in order to

$$\text{Maximize } \sum_{i=1}^{n} v_i x_i$$

$$\text{Subject to } \sum_{i=1}^{n} w_i x_i \leq W \text{ and } x_i \leq m_i \;\; \forall i \in [1, \text{n}].$$

2.2. Different Cases of the Knapsack Problem

When $m_i = 1 \;\; \forall i \in [1, \text{n}]$, it is the “0-1 knapsack problem” (Knapsack problem - 2021), meaning that there is only one for each item, and it can either be chosen or not chosen.

When $m_i = c_i \ \forall i \in [1, \mathrm{n}]$, where c_i is an arbitrary positive integer, it is the "bounded knapsack problem" (Knapsack problem - 2021). It is the most common case in real life.

When $m_i = \infty \ \forall i \in [1, \mathrm{n}]$, it is the "unbounded knapsack problem" (Knapsack problem - 2021). There is no limit on the number of items.

Because the three cases are similar, I would only analyze the bounded knapsack problem because the solution to the two other cases are similar. The 0-1 knapsack problem is a special case of the bounded knapsack problem when $c = 1$, and the unbounded knapsack problem is another special case when $c = \infty$.

To begin with, I would like to introduce two basic examples of the question and develop a preliminary method. Then I will gradually extend and optimize the method.

2.3 Definition of Key Terms

1. In this essay, all variables are non-negative integers. The only exception is $aveVal$s, which can be any rational number.

2. A "scheme" refers to a way of selecting different quantities of different items to completely fill a knapsack of a given weight capacity. In mathematical language, a "scheme" is a way of filling the knapsack using x_i number of item i $\forall i \in [1, n]$ to completely fill a knapsack of weight w.

3. If the variables of max function involves i or j, e.g. max $(w_i x_j)$, it means the maximum value of $w_i x_j$ for all possible i and j.

4. "Memory cost" measures the approximate memory required to store a function as a matrix in a computer. The matrix has the same dimension as the function, and its value is same as the function value when all variables are the same. The memory cost is the product of the number of choices for each dimension. It is most directly affected by the dimension of the function.

5. “Number of steps” of a method measures the approximate effort required to calculate the result using the method. Usually, we are interested in the approximate number of steps, so simple arithmetic such as plus 1 on a variable is ignored.

3. Examples and Methods

3.1. Example 1

Three items are given, labeled A, B, and C respectively. There are 4 As, each with a value of 6 and a weight of 2. There are 3 Bs, each with a value of 5 and a weight of 2. There are two Cs, each with a value of 10 and weight 3. The maximum weight capacity is eleven. Find the maximum total value available.

3.1.1. Solution

It is straightforward to think that only when the value per unit weight is large can the total value be large. Then we calculate the average value per unit weight of an item, and call it $aveVal_i$ for item i.

We get the following data:

$$aveVal_A = \frac{6}{2} = 3$$

$$aveVal_B = \frac{5}{2} = 2.5$$

$$aveVal_C = \frac{10}{3} = 3.3333\ldots$$

$aveVal_c > aveVal_A > aveVal_B$, so we first choose as many Cs as possible, then as many As as possible, finally as many Bs as possible. We first choose all the Cs to get a value of 20 and a weight of 6. Then, for $11 - 6 = 5$ units of weight left, we choose $\left\lfloor \frac{5}{2} \right\rfloor = 2$ As to produce the total value of 32 and total weight of 10. The 1 unit of weight left cannot be used to put Bs inside, so the process ends and the maximum value is 32, derived from 2As and 2Cs.

3.1.2. Method 0

By summarizing the process in solving example 1, I have come up with method 0. Note that it is called method “0” because it reflects some the same methodology as the first idea but leads to wrong results.

1. By default, initialize: $x_i = 0 \ \forall i \in [1, n]$.

2. Let $aveVal_i$ be the average value per unit of weight for item i, calculate $aveVal_i = \frac{v_i}{w_i}$.

3. Sort $aveVal$s in descending order.

4. Suppose item j has the largest $aveVal$, continue taking it until either there is no more this type of items left, or the capacity limit is reached. $x_j = \min\left(m_j, \left\lfloor\frac{W_{left}}{w_j}\right\rfloor\right)$, where W_{left} is the weight capacity left after the previous step.

5. Change W_{left}’s value to $W_{left} - x_j w_j$. This indicates the change of weight capacity left after taking item j.

6. Repeat process 4 and 5 until either every item is chosen, or the capacity limit is reached.

7. Calculate the total value: $\sum_{i=1}^{n} v_i x_i$.

This method seems pretty useful. However, I want to examine another example.

3.2. Example 2

Three items are given, labeled A, B, and C respectively. There are 3 As, each with a value of 9 and a weight of 5. There are 5 Bs, each with a value of 1 and a weight of 1. There is 1 C, with a value of 22 and weight 11. The maximum weight capacity is 15. Find the maximum total value available.

3.2.1. Solution Using Method 0

First, calculate all $aveVal$s.

$$aveVal_A = \frac{9}{5} = 1.8$$

$$aveVal_B = \frac{1}{1} = 1$$

$$aveVal_C = \frac{22}{11} = 2$$

$aveVal_c > aveVal_A > aveVal_B$, so we first choose C, then choose B.

$$W_{left} = 15, x_c = \min\left(m_c, \left\lfloor\frac{W_{left}}{w_c}\right\rfloor\right) = \min\left(1, \left\lfloor\frac{15}{11}\right\rfloor\right) = 1$$

$$W_{left} = 4, x_B = \min\left(m_B, \left\lfloor\frac{W_{left}}{w_B}\right\rfloor\right) = \min\left(5, \left\lfloor\frac{4}{1}\right\rfloor\right) = 4$$

$$W_{left} = 0$$

By default, $x_A = 0$. So the maximum total value is $\sum_{i=1}^{n} v_i x_i = 22 \cdot 1 + 1 \cdot 4 = 26$.

3.2.2. Reflection and Evaluation

The weight capacity, 15, can hold exactly 3 As, producing a total value of $9 \cdot 3 = 27$. $27 > 26$. The previous method clearly doesn’t work.

I reexamined my method. After I place a C, the knapsack gets only 4 units of weight left, and I can only put Bs in it. B’s $aveVal$ is only 1 while A’s $aveVal$ is 1.8, so it is much less worthy of using Bs than As.

I used an idea of “greedy algorithm” (Greedy algorithm - Wikipedia, 2021) that “makes the locally optimal choice at each stage” (Greedy algorithm - Wikipedia, 2021) but ignored the actual optimal choice.

Then I tried some other methods.

3.3. Method 1

3.3.1. Methodology

The most straightforward way is enumeration, i.e., trying every possible way of putting items in the knapsack. $x_i \in [0, m_i] \ \forall i \in [1, n]$. The number to include for item i has $m_i + 1$ choices, so we try all of the possible x_i values for each i, and calculates the maximum value.

3.3.2. Reflection and Evaluation

However, this method takes way too much time. There are $\prod_{i=1}^{n}(m_i + 1)$ possible ways to put those items in the knapsack, so it roughly takes $\prod_{i=1}^{n} m_i$ number of steps.

Notice that when $w_i x_i > W$, it exceeds the weight capacity. The maximum number of item i to include in a knapsack is actually min $(m_i, \left\lfloor \frac{W}{w_i} \right\rfloor)$.

3.4. Method 2

3.4.1. Methodology

Let $P = \underset{1 \le i \le n}{\operatorname{lcm}} (w_i)$, where lcm refers to "least common multiple" (Least common multiple - Wikipedia, 2021) of all w_i values. The purpose is to ensure that the method works in weight capacity from 0 to P, as multiple schemes can lead to the same weight. Hence, we can use the enumeration method to settle the optimal choice for a weight capacity under P. For weight capacities greater than P, we use the greedy idea in method 0. That is to say,

If $W > P$, then:

1. Use method 0 to calculate x_is until $W_{left} = P$.

2. Use method 1 to calculate the optimal scheme for a weight capacity of P. Notice that the maximum number of item i to include in a knapsack is actually min $(m_i, \left\lfloor \frac{W}{w_i} \right\rfloor)$.

3. Combine x_i values in step 1 and 2.

4. Calculate the total value: $\sum_{i=1}^{n} v_i x_i$.

If $W \le P$, then:

Use method 1 to calculate the maximum value.

3.4.2. *Reflection and Evaluation*

The steps required in the enumeration part is $\prod_{i=1}^{n}\left(\min\left(m_i, \left\lfloor\frac{P}{w_i}\right\rfloor\right) + 1\right)$, approximately $\prod_{i=1}^{n}\min\left(m_i, \left\lfloor\frac{P}{w_i}\right\rfloor\right)$. It significantly reduces the steps when $W \gg P$.

However, enumeration still seems undesirable when P is large.

Let $f(w)$ be the maximum value obtained from completely filling a knapsack of weight capacity w. Then, $f(w)$ may lead to other f values: if we add another item in the knapsack, then the maximum value may change. So, we consider every item that can be put in the knapsack. If one item i is selected, then the value changes by v_i and weight changes by w_i. When there is still enough capacity to select an extra item i ($w + w_i \leq W$) and there are still items left ($x_i + 1 \leq m_i$), we have:

$$f(w + w_i) = f(w) + v_i$$

Add a max function to indicate that $f(w + w_i)$ is the optimal value from selecting one of the n different items. Also, included $f(w + w_i)$ itself in the max function, to represent the case that no new items are selected.

$$f(w + w_i) = \max(f(w + w_i), f(w) + v_i)$$

To write it in a more readable form:

$$f(w) = \max_{1 \leq i \leq n}(f(w), f(w - w_i) + v_i)$$

Given that $w - w_i \geq 0$ and $x_i + 1 \leq m_i$.

However, it is hard to keep track of the items selected in each process, which is, we don't know x_i and cannot ensure that we are not taking more items than the total items given. So, I decide to add x_is to the variable of the function: change $f(w)$ to be $f(w, x_1, x_2, \dots, x_n)$ that indicates the maximum value obtained from **completely filling** the knapsack of weight capacity w and x_i number of item is are used.

3.5. Method 3

3.5.1. Methodology

Let $f(w, x_1, x_2, \dots, x_n)$ be the maximum value obtained from **completely filling** the knapsack of weight capacity w and x_i number of item is are used. $f(w - w_1, x_1 - 1, x_2, \dots, x_n)$ means not picking an item 1 compared with $f(w, x_1, x_2, \dots, x_n)$: $f(w, x_1, x_2, \dots, x_n) = f(w - w_1, x_1 - 1, x_2, \dots, x_n) + v_1$. Repeat this process for all n items, we get the maximum value for taking an arbitrary item is: $\max\limits_{1 \le i \le n}(\, f(w - w_i, x_1, x_2, \dots, x_i - 1, \dots, x_n) + v_1)$. Also, if no more items are selected, the function's value does not change. Adding the consideration of "no change", we have:

$$f(w, x_1, x_2, \dots, x_n) = \max\Big(f(w, x_1, x_2, \dots, x_n), \max_{1 \le i \le n}(\, f(w - w_i, x_1, x_2, \dots, x_i - 1, \dots, x_n) + v_1)\Big)\,(1)$$

For every i such that $w \ge w_i$ and $x_i + 1 \le m_i$.

Or we can say:

Given $f(w, x_1, x_2, \dots, x_n)$, we calculate

$$f(w + w_i, x_1, \dots, x_i + 1, \dots, x_n) = \max(f(w + w_i, x_1, \dots, x_i + 1, \dots, x_n), f(w, x_1, \dots, x_n) + v_i)\ \forall i \in [1, \mathrm{n}](2)$$

For every i such that $w + w_i \le \mathrm{W}$ and $x_i + 1 \le m_i$.

Equation (1) shows the theory, while equation (2) tells the action.

In mathematical words, method 3 works as follows:

1. Initialize: $f(w, x_1, x_2, \dots, x_n) = 0\ \ \forall w \le W, x_i \le m_i$.

2. Calculate $f(w_i, 0, 0, \dots, 1, \dots, 0) = v_i$ (the $(i + 1)$th variable is 1).

3. Continue using equation (2) until every f value is known.

4. The answer to the original question is $\max\big(f(w,x_1,x_2,\dots,x_n)\big)\ \forall x_i \le m_i, w \le W$. This means the maximum value of filling the knapsack is the maximum value of all possible ways of filling it.

3.5.2. Reflection and Evaluation

The answer is not $f(W,x_1,x_2,\dots,x_n)$, where the first variable is W instead of w, because using W means the knapsack has to be filled completely. In cases like W is odd and all items has an even weight, the knapsack will never be filled completely, then $f(W,x_1,x_2,\dots,x_n) = 0$, and the answer is also 0.

This method can also calculate the result from only using some items and the weight capacity is not W. If the numbers of items provided changes from m_i to m_i' and weight capacity changes from W to W', the maximum possible value is $\max\big(f(w,x_1,x_2,\dots,x_n)\big)\ \ \forall x_i \le m_i'$ and $w \le W'$.

Note that the "max" function in equation (1) and (2) is unnecessary. Actually, from the definition of f, every f value corresponds to a unique way of selecting the items and filling the knapsack, so its value is determined and not subject to change. For example, $f(10,2,3,4)$ means **exactly** using 2 As, 3Bs, and 4Cs to fill a knapsack of weight capacity 10. Therefore, the essential idea of this method is still enumeration, taking roughly $\prod_{i=1}^{n} m_i$ number of steps. This method's memory cost is $W \cdot \prod_{i=1}^{n} m_i$. Despite the magnificent and impractical number of steps and memory cost, the idea of using functions to express values obtained from schemes is worth expanding.

Incorporating all n item numbers might be unnecessary. I decide to use $f(w,i)$, which means the maximum value from using the first to the ith item and exactly filling a knapsack of weight capacity w.

3.6. Method 4

3.6.1. Methodology

Let $f(w,i)$ be the maximum value from using the first to the $i$rh item and completely filling a knapsack of weight capacity w. We can put 0,1, 2, … to m_i number of item i in the knapsack. If x_i

number of item i is used, then it brings $v_i x_i$ values and costs $w_i x_i$ units of weight. Then we get the equation: $f(w,i) = f(w - w_i x_i, i-1) + v_i x_i$. Combining all $m_i + 1$ options, we have:

$$f(w,i) = \max_{0 \le x_i \le m_i} (f(w - w_i x_i, i-1) + v_i x_i) \tag{3}$$

For $w \ge w_i x_i$ and $i \ge 1$.

In mathematical words, method 4 works as follows:

1. Initialize: $f(w,i) = 0 \;\; \forall w \le W, x_i \le m_i$.

2. $f(w_1 x_1, 1) = v_1 x_1 \;\; \forall x_1 \le m_1$.

3. $f(w,2) = \max_{0 \le x_2 \le m_2} (f(w - w_2 x_2, 1) + v_2 x_2) \; \forall w \ge w_2 x_2$.

4. Using equation (3), repeat process 3 $\forall i \in [3,n], w \ge w_i x_i$.

5. The answer to the original question is $\max\big(f(w,n)\big) \;\; \forall w \le W$.

3.6.2. Reflection and Evaluation

I think method 4 works because the way that function values are assigned goes only in one direction, primarily in ascending order of i and secondarily in ascending order of w. The value of a scheme with larger weight capacity is only determined by other schemes with less weight capacity, and it doesn't work in reverse.

The main reason for method 4 being much more efficient than method 3 is that it only finds the possible results in a much smaller region that is probable to produce the answer. This primarily works when the sum of weight of previous items already exceeds the weight capacity, no more items are taken. For each i, there are $m_i + 1$ choices for x_i, and for the n different items there are $\sum_{i=1}^{n}(m_i + 1)$ steps. Besides, there are $W + 1$ different w values. The approximate number of steps required is therefore $\sum_{i=1}^{n}(m_i + 1)\,(W + 1) \approx W \cdot \sum_{i=1}^{n} m_i$. This is significantly less than the number of steps in method 1 and 2.

Method 4 only records the greatest value from using the previous i items and a given weight capacity. This significantly saves the memory cost compared with method 3, as it only needs a 2-dimensional matrix and Wn units of memory.

I think about reducing the number of variables again. Reexamining my steps in method 4, I discover that all operations in $f(*, i)$ only requires values in $f(*, i-1)$, where * means "anything". Consider the whole binary function to be a set of layers of a unary function stacking up, and $f(*, i)$ is the ith layer. Layer i only requires value from layer $i-1$, which suggests that lower layers only pass values to the upper layers while the upper layer cannot modify values in the lower layer. Additionally, layer i only requires value from its adjacent layer, $i-1$, and all previous layers do not matter in this case. Therefore, the values are assigned from the lowest layer, layer by layer to the top layer.

To further optimize, I use a function called src (short for source), and a function called $dest$ (short for destination). src represents the function values in a lower layer (let it be ith layer) while $dest$ represents the values in a higher layer (let it be $(i+1)$th layer).

3.7. Method 5

3.7.1. Methodology

Let $src(w)$ be the maximum value from using the first to ith order of item, and completely filling a knapsack of weight capacity w. Let $dest(w)$ be the maximum value from using the first to $(i+1)$th order of item, and completely filling a knapsack of weight capacity w. Then we have:

$$dest(w) = \max_{0 \le x_i \le m_i} (src(w - w_i x_i) + v_i x_i) \tag{4}$$

$$src(w) = dest(w) \ \ \forall w \le W \tag{5}$$

Repeat this process $\forall i \le n-1$.

In mathematical words, method 5 works as follows:

1. Initialize: $src(w) = 0 \ \forall w \leq W$.

2. $src(w_1 x_1) = v_1 x_1 \ \forall x_1 \leq m_1$.

3. $dest(w) = \max\limits_{0 \leq x_2 \leq m_2} (src(w - w_2 x_2) + v_2 x_2) \ \forall w \geq w_2 x_2$.

4. $src(w) = dest(w) \ \forall w \leq W$.

5. Using equation (4) and (5), repeat process 3 and 4 $\forall i \in [3, n-1]$, $w \geq w_i x_i$.

6. The answer to the original question is $\max\big(dest(w)\big) \ \forall w \leq W$.

3.7.2. Reflection and Evaluation

Method 5 uses two functions with only one independent variable throughout. It reduces memory cost to $2W$ compared with method 4.

Notice that method 5 still requires trying out every x_i value in every layer, so the total number of steps does not change. It is still roughly $W \cdot \sum_{i=1}^{n} m_i$ steps.

3.8. Method 6

3.8.1. Methodology

Notice that the order of assignment explained in 3.6.2 is: primarily in ascending order of i and secondarily in ascending order of w. If I only use one function, say, $f(w)$ be the maximum value of completely filling a knapsack of weight capacity w, then should require a value from $f(w')$ where w' is less than w and $f(w')$ means the value from a lower layer. Thus, the order of assignment should be in descending order of w, to ensure that the data we require comes from a previous layer.

$$f(w) = \max_{0 \leq x_i \leq m_i} (f(w - w_i x_i) + v_i x_i) \tag{6}$$

Repeat this process for all $i \leq n$, and calculate $f(w)$ values in descending order of w.

In mathematical words, method 5 works as follows:

1. Initialize: $f(w) = 0 \;\; \forall w \leq W$.

2. $f(w_1 x_1) = v_1 x_1 \;\; \forall x_1 \leq m_1$.

3. Calculate $f(w) = \max\limits_{0 \leq x_2 \leq m_2} (f(w - w_2 x_2) + v_2 x_2) \;\; \forall w \in [w_2 x_2, W]$ in ascending order of w.

4. Using equation (6), repeat process 3 $\forall i \in [3, n], w \geq w_i x_i$.

5. The answer to the original question is $\max\big(f(w)\big) \;\forall w \leq W$.

3.8.2. Reflection and Evaluation

This method is currently the most efficient one because it takes the least steps and the lowest memory cost. Compared with method 5, it only requires one function throughout, and the memory cost is W.

However, the number of steps for this method is still approximately $W \cdot \sum_{i=1}^{n} m_i$.

4. Further Explanation

4.1. General Evaluation

I have developed seven methods in this essay.

1. Method 0 uses an idea similar to "greedy algorithm" (Greedy algorithm - Wikipedia, 2021). It is problematic as it only focuses on "the best" locally and ignores the true best result as shown in example 2.

2. Method 1 uses enumeration. Number of steps: $\prod_{i=1}^{n} m_i$.

3. Method 2 is a minor improvement on method 1 by reducing the enumeration region to $[0, \operatorname{lcm}\limits_{1 \leq i \leq n}(w_i)]$. Number of steps: $\prod_{i=1}^{n} \min\left(m_i, \left\lfloor \frac{P}{w_i} \right\rfloor\right)$.

4. Method 3 uses a function to express the value of a scheme, but its essence is still enumeration. Number of steps: $\prod_{i=1}^{n} m_i$; memory cost: $W \cdot \prod_{i=1}^{n} m_i$.

5. Method 4 improves on method 3 by reducing the number of variables of the function to 2 and significantly reduce the number of steps and memory cost. Number of steps: $W \cdot \sum_{i=1}^{n} m_i$; memory cost: Wn.

6. Method 5 improves the memory cost by using two unary functions, src and $dest$. It has same number of steps as method 4. Memory cost: $2W$.

7. Method 6 further improves the memory cost by using only one unary function. It has same number of steps as method 4. Memory cost: W.

The efficiency of methods is better for newer methods than previous methods.

4.2. Extension: Multi-dimensional Weight Requirement

When there is more than one weight requirement (here the concept of "weight" means a type of requirement, and in real life it could be weight, length, price, etc.), the function adds other dimensions, and takes the similar steps in method 6.

4.2.1. Definition

A knapsack has k different types of weight capacities, being $W_1, W_2, \dots, W_k$ units respectively. Given n items, numbered from 1 up to n. A $k \times n$ matrix $w_{k,n}$ means, $w_{j,i}$ represents the weight of item i under criterion j. Item i has a value of v_i and a maximum amount of m_i, and x_i number of item i is included in the knapsack.

$$\text{Maximize } \sum_{i=1}^{n} v_i x_i$$

$$\text{Subject to } \sum_{i=1}^{n} w_{j,i} x_i \le W_k \text{ and } x_i \le m_i \;\; \forall i \in [1, \mathrm{n}], j \le k.$$

4.2.2. Solution

Let $f(w_1', w_2', \dots, w_k')$ be the maximum value of completely filling a knapsack of weight capacity $\{w_1', w_2', \dots, w_k'\}$ in all k requirements, then we have:

$$f(w_1', w_2', \dots, w_k') = \max_{0 \le x_i \le m_i} \left(f\left(w_1' - w_{i,1} x_i, w_2' - w_{i,2} x_i, \dots, w_n' - w_{i,n} x_i\right) + v_i x_i \right)$$

Repeat this process for all $i \leq n$. The steps are similar in method 6, and the answer to the question is $\max\left(f(w_1', w_2', \dots, w_k')\right)\ \ \forall w_i' \leq W_i$.

5. Conclusion

This essay mainly discusses the knapsack problem. First, I have explained the definition of knapsack problem and narrowed the topic to the "bounded knapsack problem." Then, I have given two examples and seven methods, gradually optimizing each one. Method 6 requires low memory cost and few total steps. I also extend the topic to cases with multi-dimensional weight requirement.

The knapsack problem has real life implications. For example, in allocating our time for a vacation, there are various necessary tasks to accomplish and fun things to do, but all of them have a cost in time, effort, and material. Within the limited time and budget for the vacation (similar to weight capacity of the knapsack), we need to efficiently allocate the resources to yield the best possible outcome.

Still, the number of steps for method 6, $W \cdot \sum_{i=1}^{n} m_i$, might still be further reduced. I would prefer to investigate further in those extended questions in the future when I have a more profound understanding of those fascinating math topics.

6. Bibliography

1. Cui, T., 2021. *pack/V2.pdf at master · tianyicui/pack*. [online] GitHub. Available at: <https://github.com/tianyicui/pack/blob/master/V2.pdf> [Accessed 14 June 2021].

2.En.wikipedia.org. 2021. *Knapsack problem - Wikipedia*. [online] Available at: <https://en.wikipedia.org/wiki/Knapsack_problem> [Accessed 14 June 2021].

3. En.wikipedia.org. 2021. *Mathematical induction - Wikipedia*. [online] Available at: <https://en.wikipedia.org/wiki/Mathematical_induction> [Accessed 14 June 2021].

4. En.wikipedia.org. 2021. *Greedy algorithm - Wikipedia*. [online] Available at: <https://en.wikipedia.org/wiki/Greedy_algorithm> [Accessed 18 October 2021].

5. En.wikipedia.org. 2021. *Least common multiple - Wikipedia*. [online] Available at: <https://en.wikipedia.org/wiki/Least_common_multiple> [Accessed 18 October 2021].