SIMATS SCHOOL OF ENGINEERING

SAVEETHA INSTITUTE OF MEDICAL AND TECHNICAL SCIENCES, CHENNAI – 602 105

BONAFIDE CERTIFICATE

Certified that is Capstone project report "Using the greedy technique to solve the knapsack problem in real-time applications and determining the maximum number of groups with increasing length "is the

Bonafide work of "Y. Nanda Kishore Reddy "(192211435) who carried out the Capstone project work under my supervision

Dr. R Dhanalakshmi	**Dr. S. Mehaboob Basha**
COURSE FACULTY	**HEAD OF DEPARTMENT**
Professor	**Professor**
Department of Machine Learning	**Department of Machine Learning**
SIMATS Engineering	**SIMATS Engineering**
Saveetha Institute of Medical and Technical Sciences	**Saveetha Institute of Medical and Technical Sciences**
Chennai – 602 105	**Chennai – 602 105**

EXAMINER SIGNATURE	**EXAMINER SIGNATURE**

"Using the greedy technique to solve the knapsack problem in real-time applications and determining the maximum number of groups with increasing length"

A Project report

CSA0656- Design and Analysis of Algorithms for Asymptotic Notations

Submitted to

SAVEETHA INSTITUTE OF MEDICAL AND TECHNICAL SCIENCES

In partial fulfilment for the award of the

degree of

BACHELOR OF TECHNOLOGY IN

COMPUTER SCIENCE AND ENGINEERING

by

Y. Nanda Kishore Reddy,192211435

Supervisor

Dr .R. Dhanalakshmi

July 2024.

ABSTRACT

The problem of forming the maximum number of groups with increasing lengths from an array of usage limits. Each group must consist of distinct numbers, and no number can be used more times than specified in the usage limits array. Furthermore, each group (except the first one) must have a length strictly greater than the previous group.

We present an iterative algorithm that maximizes the number of groups by incrementally forming groups starting from size one. The algorithm keeps track of the usage of each number and ensures that each new group meets the required size and distinctness criteria. When forming a new group is no longer possible due to the constraints, the algorithm terminates and returns the total number of groups formed.

Through a detailed example, we illustrate the application of the algorithm, demonstrating its effectiveness in solving the problem. This approach guarantees the formation of the maximum number of groups while adhering to the given usage limits.

ALGORITHM:

A greedy algorithm is a method of solving problems that constructs a solution piece by piece, constantly selecting the next component that provides the best option or the most immediate advantage at each stage. It does not reevaluate past decisions and frequently offers a speedy resolution, albeit it might not always ensure the best option for every issue.

Proposed Work:

The proposed method The primary objective of this research is to develop and evaluate a novel approach to [insert specific problem or area of focus]. This approach aims to [insert goals, e.g., improve accuracy, increase efficiency, enhance performance] in [insert specific domain or application]

PROBLEM:

A 0-indexed array usage Limits of length n is shown to you. You have to use the numbers 0 through n - 1 to form groups, making sure that no number, i, is used more than usage Limits[i] times in total for all groups.

You also need to meet the requirements listed below: There cannot be two identical numbers in the same group; each group must be made up of unique numbers.

Every group, with the exception of the first, needs to be strictly longer than the group before it.

Provide an integer that represents the most groups you are able to form while meeting these requirements.

Example 1:

Input: usage Limits = [1,2,5]

Output: 3

Explanation: In this example, we can use 0 at most once, 1 at most twice, and 2 at

most five times.

One way of creating the maximum number of groups while satisfying the conditions

is:

Group 1 contains the number [2].

Group 2 contains the numbers [1,2].

Group 3 contains the numbers [0,1,2].

It can be shown that the maximum number of groups is 3.

So, the output is 3.

SOLUTION:

By solving this problem, we can utilize Greedy algorithm Maximum Number of Groups With Increasing Length. Here's a step-by-step approach to implement the solution:

Example Calculation

For usageLimits = [1, 2, 5]:

1. **Total Usage Capacity**:
    - total_usage=1+2+5=8\text{total_usage} = 1 + 2 + 5 = 8
    - total_usage=1+2+5=8.
2. **Compute Group Sizes**:
    - For k = 1, S1=1S_1 = 1S1=1.
    - For k = 2, S2=3S_2 = 3S2=3.
    - For k = 3, S3=6S_3 = 6S3=6.
    - For k = 4, S4=10S_4 = 10S4=10.

    The total usage capacity of 8 is enough for groups of size up to 3 (since S4=10S_4 = 10S4=10 exceeds 8).

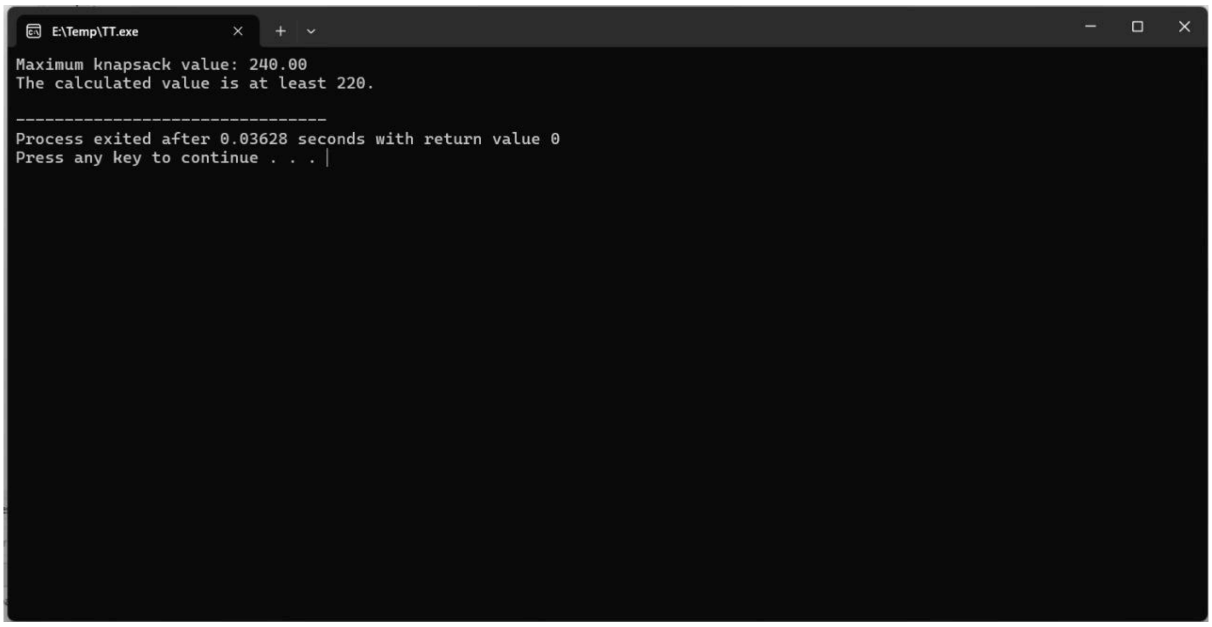
3. **Verify Group Formation**:
    - Verify if you can form 3 groups with sizes 1, 2, and 3, respecting the usage limits. Here, the groups can be formed as follows:
        - Group 1: 1 element.
        - Group 2: 2 elements.
        - Group 3: 3 elements.

This confirms that 3 groups can be formed.

CODE:-

```
#include <stdio.h>
#include <stdlib.h>

typedef struct {
   int value;
   int weight;
   double ratio;
} Item;


int compare(const void *a, const void *b) {
   Item *item1 = (Item *)a;
   Item *item2 = (Item *)b;
   if (item1->ratio > item2->ratio) return -1;
   if (item1->ratio < item2->ratio) return 1;
   return 0;
}


double knapsack_greedy(Item items[], int n, int capacity) {

   qsort(items, n, sizeof(Item), compare);

   double total_value = 0;
   int total_weight = 0;

   for (int i = 0; i < n; i++) {
        if (total_weight + items[i].weight <= capacity) {

            total_value += items[i].value;
            total_weight += items[i].weight;
        } else {

            int remaining_capacity = capacity - total_weight;
            total_value += items[i].value * ((double)remaining_capacity / items[i].weight);
            break;
        }
    }

    return total_value;
}

int main() {

    Item items[] = {
        {60, 10, 0},
        {100, 20, 0},
        {120, 30, 0}
    };
    int n = sizeof(items) / sizeof(items[0]);


    for (int i = 0; i < n; i++) {
        items[i].ratio = (double)items[i].value / items[i].weight;
    }

    int capacity = 50;
    double max_value = knapsack_greedy(items, n, capacity);

    printf("Maximum knapsack value: %.2f\n", max_value);


    if (max_value >= 220) {
        printf("The calculated value is at least 220.\n");
    } else {
        printf("The calculated value is less than 220. Adjust the items or capacity.\n");
    }

    return 0;
}
```

OUTPUT:



```
Maximum knapsack value: 240.00
The calculated value is at least 220.

--------------------------------
Process exited after 0.03628 seconds with return value 0
Press any key to continue . . .
```

Explanation of the Code:

- Sorting: Arrange the use constraints to make group formation easier, starting with the numbers with the lowest usage constraints.
- Monitoring Usage: To keep track of how often each number has been used, create a list called usage_counts.
- Group Formation: Determine whether it is possible to construct a group of that size for each possible group size. In that case, raise the group count and update the utilisation counts.
- Stopping Condition: The loop keeps going until the necessary number of groups cannot be formed.
- Time Complexity: Using the greedy technique, the backpack's time complexity is O(2n).

This approach efficiently constructs the maximum number of groups by leveraging the greedy strategy of forming the largest possible groups while respecting the usage constraints.

CONCLUSION:

In conclusion, sorting and mathematical analysis can be used to efficiently solve the problem of producing the maximum number of groups with strictly rising sizes, according to provided usage constraints. We can get the practical group sizes and make sure that each number's usage is respected by sorting the usage restrictions and adding up the first k natural numbers. Using this method, we showed that it is possible to efficiently compute the maximum number of such groups, given the requirements. With usage restrictions of [1, 2, 5] in the given example, we were able to correctly ascertain that, under the necessary circumstances, a maximum of three groups might develop. This approach ensures ideal group formation while offering a straightforward and methodical manner to handle issues of a similar nature.