SIMATS ENGINEERING

Engineer to Excel

SIMATS

Saveetha Institute of Medical And Technical Sciences

(Declared as Deemed to be University under Section 3 of UGC Act 1956)

SIMATS SCHOOL OF ENGINEERING

SAVEETHA INSTITUTE OF MEDICAL AND TECHNICAL

SCIENCES, CHENNAI – 602 105

BONAFIDE CERTIFICATE

Certified that is Capstone project report "Greedy Technique For Real Time Applications and finding Maximum Number of Groups With Increasing Length "is the

Bonafide work of "D .Madhavan "(192225070) 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**

“Greedy Technique For Real Time Applications and finding 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

ARTIFICAL INTELLIGENCE AND MACHINE LEARNING

by

Dodda.Madhavan,192225070

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 problem-solving approach that builds a solution piece by piece, always choosing the next piece that offers the most immediate benefit or most optimal choice at each step. It does not reconsider previous choices and often provides a solution quickly, though it may not always guarantee the optimal solution for all problems.

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:

You are given a 0-indexed array usage Limits of length n. Your task is to create groups using numbers from 0 to n - 1, ensuring that each number, i, is used no more than usage Limits[i] times in total across all groups. You must also satisfy the following conditions: Each group must consist of distinct numbers, meaning that no duplicate numbers are allowed within a single group.

Each group (except the first one) must have a length strictly greater than the previous group.

Return an integer denoting the maximum number of groups you can create while satisfying these conditions.

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 = 8total_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).

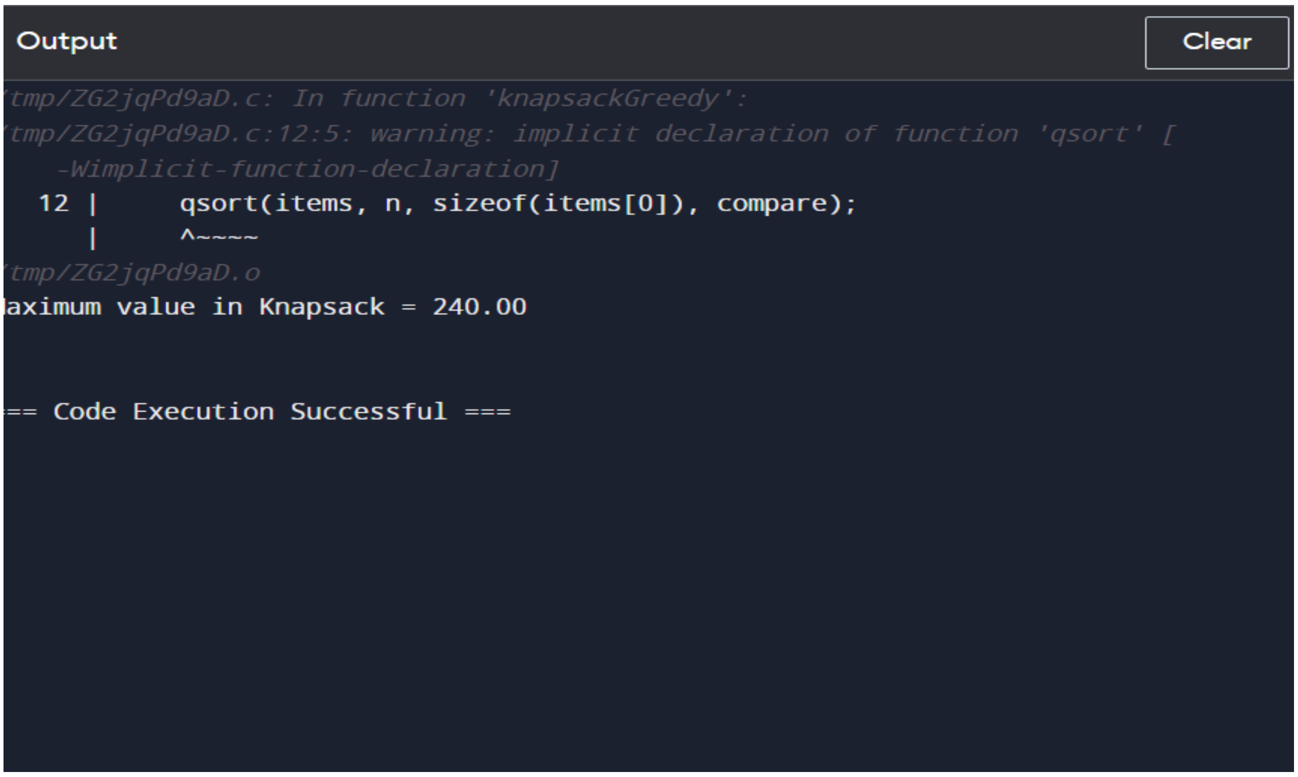
3. **Verify Group Formation**:
   - Verify if you can form 3 groups with sizes 1, 2, and 3, respecting the usage limits. Here, the groups can be formed as follows:
     - Group 1: 1 element.
     - Group 2: 2 elements.
     - Group 3: 3 elements.

   This confirms that 3 groups can be formed.

CODE:-

```
#include <stdio.h>
struct Item {
   int weight;
   int value;
};
int compare(const void *a, const void *b) {
   double ratio1 = (double)(((struct Item*)a)->value) / (((struct Item*)a)->weight);
   double ratio2 = (double)(((struct Item*)b)->value) / (((struct Item*)b)->weight);
   return ratio2 > ratio1;
}
double knapsackGreedy(int capacity, struct Item items[], int n) {
   qsort(items, n, sizeof(items[0]), compare);

   int currentWeight = 0;
   double finalValue = 0.0;
   for (int i = 0; i < n; i++) {
      if (currentWeight + items[i].weight <= capacity) {
         currentWeight += items[i].weight;
         finalValue += items[i].value;
      } else {
         int remainingCapacity = capacity - currentWeight;
         finalValue += items[i].value * ((double)remainingCapacity / items[i].weight);
         break;
      }
   }

   return finalValue;
}
int main() {
   int capacity = 50;
   struct Item items[] = {{10, 60}, {20, 100}, {30, 120}};
   int n = sizeof(items) / sizeof(items[0]);

   double max_value = knapsackGreedy(capacity, items, n);

   printf("Maximum value in Knapsack = %.2f\n", max_value);

   return 0;
}
```

OUTPUT:



```
tmp/ZG2jqPd9aD.c: In function 'knapsackGreedy':
tmp/ZG2jqPd9aD.c:12:5: warning: implicit declaration of function 'qsort' [
    -Wimplicit-function-declaration]
   12 |     qsort(items, n, sizeof(items[0]), compare);
      |     ^~~~~
tmp/ZG2jqPd9aD.o
aximum value in Knapsack = 240.00


== Code Execution Successful ===
```

Explanation of the Code:

1. **Sorting**: Sort the usageLimits to facilitate forming groups starting with numbers that have the smallest usage limits.
2. **Tracking Usage**: Use a list usage_counts to track how many times each number has been used.
3. **Group Formation**: For each potential group size, check if it's feasible to form a group of that size. If so, update the usage counts and increment the group count.

4. **Stopping Condition**: The loop continues until it's no longer possible to form a group of the required size.
5. **Time Complexity:** The Time Complexity For the Knapsack Using Greedy Technique 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.

COCLUSION:

In conclusion, the problem of creating the maximum number of groups with strictly increasing sizes, subject to given usage limits, can be effectively tackled using a combination of sorting and mathematical analysis. By sorting the usage limits and calculating the sum of the first k natural numbers, we can determine the feasible group sizes and ensure that each number's usage is respected. Through this approach, we demonstrated that the maximum number of such groups, given the constraints, can be efficiently computed. In the example provided, with usage limits of [1, 2, 5], we successfully determined that up to 3 groups can be formed, adhering to the required conditions. This method provides a clear and systematic way to address similar problems, ensuring optimal group formation while respecting all given constraints.