Optimal Storage on Tapes

Sagnik Chaudhuri

Given $n$programs stored on a computer tape and length of each program $i$is $L_i$where $1 <= i <= n$, find the order in which the programs should be stored in the tape for which the Mean Retrieval Time (MRT given as $\frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{i} L_j$) is minimized.

The greedy algorithm finds the MRT as following:

```
Algorithm MRT_SINGLE_TAPE(L)
 // Description: Find storage order of n programs to such that mean
retrieval time is minimum
 //Input: L is array of program length sorted in ascending order
 // Output: Minimum Mean Retrieval Time

Tj  <– 0

for i <– 1 to n do

for j <- 1 to i do
 Tj <- Tj + L[j]
    end
end

MRT <- sum(T) / n
```

Complexity analysis:

Primitive operation in the above algorithm is the addition of program length, which is enclosed within two loops. The running time of algorithm is given by,

$$T(n) = 0(1)$$

=Σ Σ l

$=\Sigma i=1+2+3+\ldots+n$

$=n(n+1)/2$

$= n^2/2 + n/2$

$T(n) = 0(n^2)$

This algorithm runs in $O(n^2)$time.

Example:

```
Input : n = 3
        L[] = { 5, 3, 10 }
Output : Order should be { 3, 5, 10 } with MRT = 29/3
```

Prerequisites: Magnetic Tapes Data Storage

Recommended: Please try your approach on *{IDE}* first, before moving on to the solution.

Let us first break down the problem and understand what needs to be done.

A magnetic tape provides only sequential access of data. In an audio tape/cassette, unlike a CD, a fifth song from the tape can't be just directly played. The length of the first four songs must be traversed to play the fifth song. So in order to access certain data, head of the tape should be positioned accordingly.

Now suppose there are 4 songs in a tape of audio lengths 5, 7, 3 and 2 mins respectively. In order to play the fourth song, we need to traverse an audio length of 5 + 7 + 3 = 15 mins and then position the tape head.

Retrieval time of the data is the time taken to retrieve/access that data in its entirety. Hence retrieval time of the fourth song is 15 + 2 = 17 mins.

tape head points to the front of the tape every time, a new term can be defined called the Mean Retrieval Time (MRT).

Let's suppose that the retrieval time of program $i$is T_i. Therefore, $T_i = \sum_{j=1}^{i} L_j$

MRT is the average of all such T_i. Therefore $MRT = \frac{1}{n}\sum_{i=1}^{n} T_i$, or $MRT = \frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{i} L_j$
The sequential access of data in a tape has some limitations. Order must be defined in which the data/programs in a tape are stored so that least MRT can be obtained. Hence the order of storing becomes very important to reduce the data retrieval/access time.

Thus, the task gets reduced – to define the correct order and hence minimize the MRT, i.e. to minimize the term $\sum_{i=1}^{n}\sum_{j=1}^{i} L_i$
For e.g. Suppose there are 3 programs of lengths 2, 5 and 4 respectively. So there are total 3! = 6 possible orders of storage.

	Order	Total Retrieval Time	Mean Retrieval Time
1	1 2 3	2 + (2 + 5) + (2 + 5 + 4) = 20	20/3
2	1 3 2	2 + (2 + 4) + (2 + 4 + 5) = 19	19/3
3	2 1 3	5 + (5 + 2) + (5 + 2 + 4) = 23	23/3
4	2 3 1	5 + (5 + 4) + (5 + 4 + 2) = 25	25/3
5	3 1 2	4 + (4 + 2) + (4 + 2 + 5) = 21	21/3
6	3 2 1	4 + (4 + 5) + (4 + 5 + 2) = 24	24/3

It's clear that by following the second order in storing the programs, the mean retrieval time is least.

In above example, the first program's length is added 'n' times, the second 'n-1' times… and so on till the last program is added only once. So, careful analysis suggests that in order to minimize the MRT, programs having greater lengths should be put towards the end so that the summation is reduced. Or, the lengths of the programs should be sorted immediate choice of putting the program having the least time first, in order to build up the ultimate optimized solution to the problem piece by piece.

Below is the implementation:

C++

```cpp
// CPP Program to find the order
// of programs for which MRT is
// minimized
#include <bits/stdc++.h>

using namespace std;

// This functions outputs the required
// order and Minimum Retrieval Time
void findOrderMRT(int L[], int n)
{
    // Here length of i'th program is L[i]
    sort(L, L + n);

    // Lengths of programs sorted according to increasing
    // lengths. This is the order in which the programs
    // have to be stored on tape for minimum MRT
    cout << "Optimal order in which programs are to be"
            "stored is: ";
    for (int i = 0; i < n; i++)
        cout << L[i] << " ";
    cout << endl;

    // MRT - Minimum Retrieval Time
    double MRT = 0;
    for (int i = 0; i < n; i++) {
        int sum = 0;
        for (int j = 0; j <= i; j++)
            sum += L[j];
        MRT += sum;
    }
    MRT /= n;
    cout << "Minimum Retrieval Time of this"
           " order is " << MRT;
}

// Driver Code to test above function
int main()
{
    int L[] = { 2, 5, 4 };
    int n = sizeof(L) / sizeof(L[0]);
    findOrderMRT(L, n);
    return 0;
}
```

```java
// Java Program to find the order
// of programs for which MRT is
// minimized
import java.io.*;
import java .util.*;

class GFG
{

// This functions outputs
// the required order and
// Minimum Retrieval Time
static void findOrderMRT(int []L,
                         int n)
{
    // Here length of
    // i'th program is L[i]
    Arrays.sort(L);

    // Lengths of programs sorted
    // according to increasing lengths.
    // This is the order in which
    // the programs have to be stored
    // on tape for minimum MRT
    System.out.print("Optimal order in which " +
              "programs are to be stored is: ");
    for (int i = 0; i < n; i++)
        System.out.print(L[i] + " ");
        System.out.println();

    // MRT - Minimum Retrieval Time
    double MRT = 0;
    for (int i = 0; i < n; i++)
    {
        int sum = 0;
        for (int j = 0; j <= i; j++)
            sum += L[j];
        MRT += sum;
    }
    MRT /= n;
    System.out.print( "Minimum Retrieval Time" +
                    " of this order is " + MRT);
}

// Driver Code
public static void main (String[] args)
{
    int []L = { 2, 5, 4 };
    int n = L.length;
    findOrderMRT(L, n);
}
}
```

Python3

```
# Python program to find the order of programs for which MRT is minimized

# This function outputs the required order and Minimum Retrieval Time
def findOrderMRT(L, n):

        # Here length of i'th program is L[i]
    L.sort()

    # Lengths of programs sorted according to increasing lengths.
    # This is the order in which the programs have to be stored on tape for minimum MR
    print("Optimal order in which programs are to be stored is:", end=" ")
    for i in range(0, n):
        print(L[i], end=" ")
    print()

    # MRT - Minimum Retrieval Time
    MRT = 0
    for i in range(0, n):
        sum = 0
        for j in range(0, i+1):
            sum += L[j]
        MRT += sum

    MRT /= n
    print("Minimum Retrieval Time of this order is", "{0:.5f}".format(MRT))


L = [2, 5, 4]
n = len(L)
findOrderMRT(L, n)

# This code is contributed by lokesh (lokeshmvs21).
```

C#

```
// C# Program to find the
// order of programs for
// which MRT is minimized
using System;

class GFG
{

// This functions outputs
// the required order and
// Minimum Retrieval Time
static void findOrderMRT(int []L,
                         int n)
```

```
    // i'th program is L[i]
    Array.Sort(L);

    // Lengths of programs sorted
    // according to increasing lengths.
    // This is the order in which
    // the programs have to be stored
    // on tape for minimum MRT
    Console.Write("Optimal order in " +
                  "which programs are" +
                  " to be stored is: ");
    for (int i = 0; i < n; i++)
        Console.Write(L[i] + " ");
        Console.WriteLine();

    // MRT - Minimum Retrieval Time
    double MRT = 0;
    for (int i = 0; i < n; i++)
    {
        int sum = 0;
        for (int j = 0; j <= i; j++)
            sum += L[j];
        MRT += sum;
    }
    MRT /= n;
    Console.WriteLine("Minimum Retrieval " +
                  "Time of this order is " +
                                       MRT);
}

// Driver Code
public static void Main ()
{
    int []L = { 2, 5, 4 };
    int n = L.Length;
    findOrderMRT(L, n);
}
}

// This code is contributed
// by anuj_67.
```

Javascript

```
// Javascript Program to find the order
// of programs for which MRT is
// minimized

// This functions outputs
// the required order and
// Minimum Retrieval Time
function findOrderMRT(L, n)
```

```
        // i'th program is L[i]
        L.sort();

        // Lengths of programs sorted
        // according to increasing lengths.
        // This is the order in which
        // the programs have to be stored
        // on tape for minimum MRT
       document.write("Optimal order in which " +
                   "programs are to be stored is: ");
        for (let i = 0; i < n; i++)
            document.write(L[i] + " ");
            document.write("<br/>");

        // MRT - Minimum Retrieval Time
        let MRT = 0;
        for (let i = 0; i < n; i++)
        {
            let sum = 0;
            for (let j = 0; j <= i; j++)
                sum += L[j];
            MRT += sum;
        }
        MRT /= n;
       document.write( "Minimum Retrieval Time" +
                         " of this order is " + MRT);
    }

    // driver code

         let L = [ 2, 5, 4 ];
        let n = L.length;
        findOrderMRT(L, n);
```

Output

```
Optimal order in which programs are to bestored is: 2 4 5
Minimum Retrieval Time of this order is 6.33333
```

Time complexity of the above program is the time complexity for sorting, that is $O(nlgn)$ (Since std::sort() operates in $O(nlgn)$) If you use bubble sort instead of std::sort(), it will take $O(n^2)$

You may think that the time complexity for this particular above code should be due to both the loops in 'mrt' calculation, that is,$O(n^2)$, but do remember that intuitively, the for loops used can also be coded in this manner to avoid two loops :

```
for (int i = 0; i < n; i++)
    MRT += (n - i) * L[i];
```

Last Updated : 25 Apr, 2023

5

Similar Reads

1. Sorting with Tapes : Balanced Merge
2. Vertical and Horizontal retrieval (MRT) on Tapes
3. Secretary Problem (A Optimal Stopping Problem)
4. Optimal partition of an array into four parts
5. Optimal sequence for AVL tree insertion (without any rotations)
6. Find optimal weights which can be used to weigh all the weights in the range [1, X]
7. Optimal Strategy for a Game | Set 3
8. Optimal strategy for a Game with modifications
9. Optimal Strategy for a Game | DP-31
10. Finding optimal move in Tic-Tac-Toe using Minimax Algorithm in Game Theory

Related Tutorials

1. Learn Data Structures with Javascript | DSA Tutorial
2. Introduction to Max-Heap – Data Structure and Algorithm Tutorials
3. Introduction to Set – Data Structure and Algorithm Tutorials
4. Introduction to Map – Data Structure and Algorithm Tutorials
5. What is Dijkstra’s Algorithm? | Introduction to Dijkstra's Shortest Path Algorithm

Next

Article Contributed By :

Sagnik Chaudhuri
Sagnik Chaudhuri

Vote for difficulty

Current difficulty : Easy

Easy | Normal | Medium | Hard | Expert

Improved By : vt_m, Sagnik Chaudhuri, chinmoy1997pal, lokeshmvs21, hardikkoriintern, phasing17, laxmishinde5t82

Article Tags : programming-puzzle, Arrays, DSA, Greedy

Practice Tags : Arrays, Greedy

Improve Article | Report Issue

GeeksforGeeks
A-143, 9th Floor, Sovereign Corporate Tower, Sector-136, Noida, Uttar Pradesh - 201305

feedback@geeksforgeeks.org

Company

About Us

Careers

In Media

Contact Us

Terms and Conditions

Privacy Policy

Copyright Policy

Third-Party Copyright Notices

Explore

Job Fair For Students

POTD: Revamped

Python Backend LIVE

Android App Development

DevOps LIVE

DSA in JavaScript

Languages

Python

Java

C++

GoLang

SQL

R Language

Android Tutorial

Data Structures

Array

String

Linked List

Stack

Queue

Tree

Graph

Algorithms

Sorting

Searching

Greedy

Dynamic Programming

Pattern Searching

Recursion

Backtracking

Web Development

HTML

CSS

JavaScript

Bootstrap

ReactJS

AngularJS

NodeJS

Data Science & ML

Data Science With Python

Data Science For Beginner

Machine Learning Tutorial

Maths For Machine Learning

Pandas Tutorial

NumPy Tutorial

NLP Tutorial

Interview Corner

Company Preparation

Preparation for SDE

Company Interview Corner

Experienced Interview

Internship Interview

Competitive Programming

Aptitude

Python

Python Tutorial

Python Programming Examples

Django Tutorial

Python Projects

Python Tkinter

OpenCV Python Tutorial

GfG School

CBSE Notes for Class 8

CBSE Notes for Class 9

CBSE Notes for Class 10

CBSE Notes for Class 11

CBSE Notes for Class 12

English Grammar

UPSC/SSC/BANKING

Write & Earn