Space and Time Trade-Offs

Madiba Hudson-Quansah

Contents

Chapter 1

Introduction

Space and time trade-offs are a common occurrence in computer science. The trade-off is a situation where one must sacrifice one aspect of a system to gain another. In computer science, the trade-off is usually between space and time. Space refers to the amount of memory used by a program, while time refers to the amount of time it takes to run the program. In general, the more space a program uses, the faster it will run, and vice versa. The main idea in computing is to pre-process the problem's input at least somewhat and store additional information obtained to speed up solving the problem afterwards. This is called **input enhancement**.

Another technique that exploits space-for-time trade-offs uses extra space to facilitate faster and more flexible access to the data. This is called **prestructuring** .

Chapter 2

Input Enhancement

2.1 Sorting By Counting

Definition 2.1.1: Comparison Counting Sort

For each element x in the input array A, count the number of elements less than x and store this count in an auxiliary array C. Then, use the counts in C to place each element in its correct position in the output array B.

Algorithm 1 ComparisonCountingSort $(A[0 \ldots n-1])$

▷

▷ Sorts an array by comparison counting
▷ Input: An array $A[0 \ldots n-1]$ of orderable elements
▷ Output: Array $S[0 \ldots n-1]$ of A's elements sorted in nondecreasing order

for $i \leftarrow 0$ to $n-1$ **do**
    $\text{Count}_i \leftarrow 0$
end for
for $i \leftarrow 0$ to $n-2$ **do**
    for $j \leftarrow i+1$ to $n-1$ **do**
        if $A_i > A_j$ **then**
            $\text{Count}_i \leftarrow \text{Count}_i + 1$
        else
            $\text{Count}_j \leftarrow \text{Count}_j + 1$
        end if
    end for
end for
for $i \leftarrow 0$ to $n-1$ **do**
    $S_{\text{Count}_i} \leftarrow A_i$
end for
return S

The time complexity of this algorithm is $O\left(n^2\right)$, where n is the number of elements in the input array

2.1.1 Distribution Counting Sort

The idea of the comparison counting sort performs better when the elements to be sorted belong to a small set of variables and the number of elements to be sorted is large. Given the array:

13	11	12	13	12	12

Whose elements are known to belong to the set $\{11, 12, 13\}$, and should not be overwritten in the process of sorting. The frequency and distribution arrays are:

Array Values	11	12	13
Frequencies	1	3	2
Distribution values	1	4	6

Now the distribution values indicate the final position of the last element of the corresponding value in the sorted array.

Algorithm 2 DistributionCountingSort $(A[0 \ldots n-1], l, u)$

▷

▷ Sorts an array of integers from a limited range by distribution counting
▷ Input: An array $A[0 \ldots n-1]$ of integers between l and u $(l \leqslant u)$
▷ Output: Array $S[0 \ldots n-1]$ of A's elements sorted in nondecreasing order

for $j \leftarrow 0$ to $u - l$ **do**
    $D_j \leftarrow 0$ ▷ Initialize frequencies
end for
for $i \leftarrow 0$ to $n - 1$ **do**
    $D_{A_i - l} \leftarrow D_{A_i - l} + 1$ ▷ Compute frequencies
end for
for $j \leftarrow 1$ to $u - l$ **do**
    $D_j \leftarrow D_{j-1} + D_j$
end for
for $i \leftarrow n - 1$ down to 0 **do**
    $j \leftarrow A_i - l$
    $S_{D_j - 1} \leftarrow A_i$
    $D_j \leftarrow D_j - 1$
end for
return S