MOD007357 Algorithm Analysis & Data Structures

Coursework Assignment

Trimester 2, 2022-23

The Multiset Partition Problem equiv. 1500 words: code, equations, graphs, findings

In the multiset partition problem we ask the question: can we divide a multiset S into two partitions (S_1 and S_2) that have equal sums? Note that it is not required that $|S_1| = |S_2|$; in the extreme case, there may be one member of S in S_1 and all other members of S in S_2. Recall also that *multiset* implies that S (and consequently S_1 and S_2) can contain duplicates, unlike regular sets. For example, if $S = \{2,2,1,10,5,15,4,1\}$ we can produce $S_1 = \{5,15\}$ and $S_2 = \{2,2,1,1,10,4\}$, which both sum to 20 (i.e., the difference between the sums of S_1 and S_2 is 0).

This childishly simple problem is computationally expensive if we seek to find the best answer. For a set of cardinality n there are 2^n possible subsets, so a brute force approach (i.e., trying all possibilities) would lead to exponential time complexity $O(2^n)$. It is traditionally framed as an NP-complete decision problem (i.e., producing a yes/no result, depending upon whether an equal-sum partition is or is not possible). We can also frame it as an optimisation problem that aims to minimise the absolute difference between the two sums. For instance, if $S = \{2,6,15,10\}$ then the optimal partition is $S_1 = \{2,15\}$ and $S_2 = \{6,10\}$, which have sums of 17 and 16 respectively, giving an absolute difference of 1. It is the optimisation variant of the partition problem that we shall consider here.

We will compare several approaches to see how well they partition S to minimise the difference between sums over a range of cardinalities. Our multiset S can be implemented as an array in any programming language of your choice (choose wisely: the use of OO may unnecessarily complicate matters). The array should be populated with randomly generated numbers $\in \mathbb{N}$ in the interval $[10 \times |S|..100 \times |S|]$. We will test the following 8 cardinalities of S: 8, 16, 32, 64, 128, 256, 512, and 1024. As our objective is to divide S into two partitions with equal sum, we will define the *ideal* sum to be equal to the sum of all values in S divided by 2 (although this may not be achievable in practice for a specific multiset S) and measure how close we can get to this ideal.

The algorithms to be compared are:

A. add the first half of the values in S to S_1 and the second half of the values in S to S_2 (i.e., split the array in half).
B. add all values stored in even array indices to S_1 and all values stored in odd array indices to S_2. Note: 0 is even.
C. add all even values in S to S_1 and all odd values in S to S_2.
D. add the first value in S to S_1 and the second value in S to S_2 and then iterate through the remaining values, adding them to whichever partition currently has the smallest sum.
E. sort S into ascending order, then add values stored at even array indices to S_1 and values stored at odd array indices to S_2.
F. sort S into ascending order, then add the first value in S to S_1 and the second values in S to S_2 and then iterate through the remaining values, adding them to whichever partition currently has the smallest sum.
G. sort S into descending order, then add the first value in S to S_1 and the second values in S to S_2 and then iterate through the remaining values, adding them to whichever partition currently has the smallest sum.
H. iterate through all possible two-way partitions to find the one that has the lowest deviation from ideal (note: this may run indefinitely for large $|S|$, so run only as far as practically possible).
I. do some research and find a provably **better** approach (lower difference between sums) than **A..G**.

For each algorithm A..I, complete the following tasks:

1. provide a written example walkthrough (similar to those given above above) for a small $|S|$, such as 8, to make sure you fully understand how each method will work in practice (up to 50 words for each method, on average).
2. provide a function implementing each algorithm (note that, if implemented concisely, **A..G** should be ~10 lines of code each, possibly even less; **H** and **I** may be longer).
3. conduct some testing to make sure your function operates correctly by sending it some arrays containing known values and comparing the output to a paper-based walkthrough.

When done, complete the following tasks:

4. provide a test harness that runs functions **A..I**; send them randomly generated arrays and collect the deviation from the ideal partition. Run each cardinality repeatedly (e.g., 1000 times) and store all results for plotting.
5. plot all results on a line graph with $|S|$ on the x-axis and mean deviation from the ideal partition on the y-axis. Either plot in a language like MATLAB or Python with matplotlib, or export a text file (e.g., csv) from you program and plot in Excel. Add error bars to each data point to show the dispersion around the mean for each data point.
6. state the exact work and time complexity of each approach, along with some reasoning to explain how you came to this conclusion (e.g., 25 words for each method), and draw some conclusions as to which method is the best both in terms of computing cost and closeness of result to the ideal outcome (up to 250 words).

Mark Scheme

ALGORITHM	TASK 1. walk-though	2. function	3. testing	Marks Available
A	1	2	2	0..5
B	1	2	2	0..5
C	1	2	2	0..5
D	1	2	2	0..5
E	1	2	2	0..5
F	1	2	2	0..5
G	1	2	2	0..5
H	1	2	2	0..5
I	6	10	4	0..20
			MARKS SUB-TOTAL	0..60

TASK	Marks Available
4. test harness	0..10
5. results graph(s); could be >1 if you experiment with different scales	0..10
6. exact formulas, time complexities, and conclusions	0..10
Presentation Checklist	0 or 10
MARKS SUB-TOTAL	0..40

FINAL MARK	0..100

Presentation Checklist (+10% if and only if <u>all</u> are done):

Report Guidelines	Tick Box
I confirm that I have carefully proof-read my work and have used spelling and grammar checkers in my word processor..	☐
I confirm that my work is written in the third person (i.e., doesn't use "I" or any other personal pronouns) in order to maintain scientific clarity and objectivity, and is as concise and factual as possible.	☐
I confirm that all figures, tables and equations have been numbered sequentially, starting from one (Fig. 1, Fig. 2, etc for figures), that there is an explanatory caption stating what the figure/table shows (see examples in this document), and that all figures/tables/equations are referred to in the text.	☐
I confirm that Harvard referencing has been used to clearly identify all third-party sources used in the creation of my submission, which are also listed at the rear of the document for each task.	☐
I confirm that I have remained below the word limit and understand that equations, figures, tables, code snippets and pseudocode are not counted as words.	☐
I will upload a document named to match my SID in pdf or doc/docx format (e.g., `2154321.docx`), along with separate code files in C, C++, C#, Java, MATLAB or Python format. Name your functions `algorithmA.c` to `algorithmI.c` (with a different extension for other languages). No ZIP permitted.	☐

Figure, Table and Equation Guidelines	Tick Box
I confirm that I have labelled the x and y axes of all line graphs, and that the lines appearing in these line graphs are clearly identified in the figure captions provided underneath each figure, and optionally also in a figure legend.	☐
I confirm that all equations have been typeset (e.g., using the Word Equation Editor or LaTeX) rather than being copied/pasted from elsewhere or written in regular text, and that if equations correspond to code/pseudocode fragments, then equation variable names match those used in my code.	☐

Code and Pseudocode Guidelines	Tick Box
I confirm all variable names make sense with respect to their roles in my code, and I have used camelCaps for multi-word variable names and UPPER CASE for constants.	☐
I confirm that my code is well commented, paying particular attention to lines of code that are more difficult to understand on casual inspection.	☐
I confirm that I have avoided single letter variable names wherever possible, except where these relate directly to widely accepted mathematical notation (e.g., S, S_1, and S_2, are acceptable in this work).	☐
I confirm that my code has been properly indented so that nested logic (iteration, selection) is very obvious to the reader.	☐
I confirm that my code has been printed for inclusion in my report in a fixed-width-font such as `Courier` or `Courier New` to preserve indentation and aid general readability.	☐
I confirm that my code is as short as possible (does not perform unnecessary work), and does not include unreachable or non-functioning lines of code, and that different units of code within a task are harmonised in terms of the way they are presented (variable names, function arguments, etc).	☐