Analysis of Algorithms: Project 1

October 2023

1 Team Members (Team 7)

1.1 Harshit Mundada (UFID: 7022-1700)

Designed algorithms for all the strategies, came up with counter and supporting examples, wrote code for strategy 4, wrote the report, contributed in comparative study by generating the test cases of sizes 1000 to 5000, reading them and measuring average percentages, tested the code for correctness on university's server.

1.2 Rohan Sharma (UFID: 4233-3441)

Designed proof of optimality for Strategy 4, wrote the report, wrote code for strategy 3, came up with counter and supporting examples, made the graph by taking the performance measures for all the four strategies, testing the code for randomly generated test cases.

1.3 Ujjwal Goel (UFID: 8247-9467)

Wrote code for strategies 1 and 2, wrote makefile for executing code, tested code for correctness with test.sh file, refined report structure, measured performance of the strategies, proofread the report.

2 Introduction

We have written the program in Python. In the code, we have used the heapq library to implement MIN-HEAP. The heapq library uses binary heaps

internally. The complexity of initializing (heapifying) a heap from an array is $\theta(n)$. The heappush operation to add the element to heap and the heappop operation to remove the element from the heap takes $\theta(\log n)$ time. The operations compare two items stored as a tuple using the first element that is different, and in case of a tie it moves to the next one.

3 Greedy Strategies

3.1 Strategy 1

For each extra device, place it in the bag, that currently has the minimum number of working devices. Pick the bag i with minimum numWorking$_i$

Algorithm 1 Strategy1

Input: $n \leftarrow$ number of bags
$k \leftarrow$ number of working devices
$nw_i \leftarrow$ number of working devices in bag i
$t_i \leftarrow$ total number of devices in bag i

Output: S - set of indices of bags to which devices were added

1: Let H be a MIN-HEAP designed to store bags in such a way that the element at the top, represented as $[nw_0, 0, t_0]$ possesses the smallest value of nw when compared to all other elements. In case of a tie, use the index of the items
2: Initially $S \leftarrow \emptyset$
3: **while** $k > 0$ **do**
4: $b \leftarrow$ pop a bag from H
5: $S \leftarrow S \cup \{index \text{ of } b\}$
6: $b[nw] \leftarrow b[nw] + 1$
7: $b[t] \leftarrow b[t] + 1$
8: Add b back to the heap H
9: $k \leftarrow k - 1$
10: **end while**
11: **return** S

3.1.1 Analysis

1. The initializing of MIN-HEAP in step 1 takes θ(n) time.

2. The while loop runs for k times and involves pop and push operations, which take $\theta(\log n)$ time. So the overall complexity of while loop is $\theta(k \log n)$.

3. Hence, the overall complexity of the algorithm is $\theta(n + k \log n)$

3.1.2 Supporting example

1. Consider 2 bags (nw_i, t_i) : b1:(1,3), b2:(8,56) and $k = 2$

2. Strategy1 will choose the bag b1 since it has the least number of working devices and will add the new device to it.

3. Now the second device will again be added to b1 since it has the least number of working devices.

4. Finally, b1:(3,5), b2:(8,56)

5. We can see that the average percentage of working devices in the bags increases from 47% to 74%. Hence, this strategy works optimally for the above given example.

3.1.3 Counter example

1. Consider 3 bags (nw_i, t_i) $b1 = (2, 2)$, $b2 = (6, 8)$, $b3 = (9, 10)$ and $k = 2$

2. Strategy1 will choose the bag $b1$ since it has the least number of working devices and adds it to bag $b1$.

3. Now the second device will again be added to $b1$ since it has the least number of working devices.

4. Finally, $b1 = (4, 4)$, $b2 = (6, 8)$ and $b3 = (9, 10)$

5. We can see that the average percentage remains same (88.33%) even after adding 2 devices to bag $b1$ whereas it could be increased if placed in the other bags.

3.2 Strategy 2

For each extra device, place it in the bag, that currently has the minimum ratio number of working devices and total devices. Pick the bag i with minimum $\text{numWorking}_i/\text{total}_i$

Algorithm 2 Strategy2

Input: $n \leftarrow$ number of bags
$k \leftarrow$ number of working devices
$nw_i \leftarrow$ number of working devices in bag i
$t_i \leftarrow$ total number of devices in bag i

Output: S - set of indices of bags to which devices were added

1: Let H be a MIN-HEAP designed to store bags in such a way that the element at the top, represented as $[nw_0, 0, t_0]$ possesses the smallest value for the expression nw/t when compared to all other elements. In case of a tie, use the index of the items
2: Initially $S \leftarrow \emptyset$
3: **while** $k > 0$ **do**
4: $b \leftarrow$ pop a bag from H
5: $S \leftarrow S \cup \{index \text{ of } b\}$
6: $b[nw] \leftarrow b[nw] + 1$
7: $b[t] \leftarrow b[t] + 1$
8: Add b back to the heap H
9: $k \leftarrow k - 1$
10: **end while**
11: **return** S

3.2.1 Analysis

1. The initializing of MIN-HEAP in step 1 takes $\theta(n)$ time.

2. The while loop runs for k times and involves pop and push operations, which take $\theta(\log n)$ time. So the overall complexity of while loop is $\theta(k \log n)$.

3. Hence, the overall complexity of the algorithm is $\theta(n + k \log n)$

3.2.2 Supporting example

1. Consider 2 bags (nw_i, t_i) : b1:(4,6), b2:(6,8) and $k = 3$
2. Strategy2 will choose the bag b1 since it has the least value of the ratio nw_i/t_i (0.67) and will add the new device to it.
3. Now the second device will again be added to b1 since it has the least value of the ratio nw_i/t_i (0.71).
4. Lastly, after adding the second device, both the bags will have the same ratio of working devices to total devices. Hence, the third device will be added to the bag with the least index, i.e., bag b1.
5. Finally, b1:(7,9), b2:(6,8)
6. We can see that the average percentage of working devices in the bags increases from 70.8% to 76.4%. Hence, this strategy works optimally for the above given example.

3.2.3 Counter example

1. Consider 2 bags (nw_i, t_i) : b1:(4,15), b2:(4,100) and k=2
2. Strategy1 will choose the bag b2 since it has the least value of the ratio nw_i/t_i and will add the new device to it.
3. Now the second device will again be added to b2 since it has the least value of the ratio nw_i/t_i.
4. Finally, b1:(4,15), b2:(6,102)
5. We can see that the average percentage of working devices increases from 31% to 32% after following this strategy, but if we added the two bags in bag b1 instead of b2, then the average percentage of working devices would have been increased to 39%. Hence, this algorithm does not always give optimal results.

3.3 Strategy 3

For each extra device, place it in the bag, that currently has the minimum number of working devices. Pick the bag i with minimum total$_i$

Algorithm 3 Strategy3

Input: $n \leftarrow$ number of bags
$k \leftarrow$ number of working devices
$nw_i \leftarrow$ number of working devices in bag i
$t_i \leftarrow$ total number of devices in bag i

Output: S - set of indices of bags to which devices were added,

1: Let H be a MIN-HEAP designed to store bags in such a way that the element at the top $[nw_0, 0, t_0]$ has the smallest value of t when compared to all other elements. In case of a tie, use the index of the items.
2: Initially $S \leftarrow \emptyset$
3: **while** $k > 0$ **do**
4: $b \leftarrow$ pop a bag from H
5: $S \leftarrow S \cup \{index \text{ of } b\}$
6: $b[nw] \leftarrow b[nw] + 1$
7: $b[t] \leftarrow b[t] + 1$
8: Add b back to the heap H
9: $k \leftarrow k - 1$
10: **end while**
11: **return** S

3.3.1 Analysis

1. The initializing of MIN-HEAP in step 1 takes θ(n) time.

2. The while loop runs for k times and involves pop and push operations, which take $\theta(\log n)$ time. So the overall complexity of while loop is $\theta(k \log n)$.

3. Hence, the overall complexity of the algorithm is $\theta(n + k \log n)$

3.3.2 Supporting example

1. Consider 2 bags (nw_i, t_i) : b1:(1,3), b2:(8,56) and $k = 2$

2. Strategy3 will choose the bag b1 since it has the least number of total devices and will add the new device to it.

3. Now the second device will again be added to b1 since it has the least number of total devices.

4. Finally, b1:(3,5), b2:(8,56)

5. We can see that the average percentage of working devices in the bags increases from 47% to 74%. Hence, this strategy works optimally for the above given example.

3.3.3 Counter example

1. Consider 3 bags (nw_i, t_i) : b1:(2,2), b2:(6,8), b3:(9,10) and k=2

2. Strategy1 will choose the bag b1 since it has the least number of total devices and adds it to bag b1.

3. Now the second device will again be added to b1 since it has the least number of total devices.

4. Finally, b1:(4,4), b2:(6,8), b3:(9,10)

5. We can see that the average percentage remains same (88.33%) even after adding 2 devices to bag b1 whereas it could be increased if placed in the other bags.

3.4 Strategy 4

For each extra device, place it in the bag, whose percentage of number of working devices will increase the most. Pick the bag i with maximum $(\text{numWorking}_i + 1/\text{total}_i + 1) - (\text{numWorking}_i/\text{total}_i)$

Algorithm 4 Strategy4

Input: $n \leftarrow$ number of bags
$k \leftarrow$ number of working devices
$nw_i \leftarrow$ number of working devices in bag i
$t_i \leftarrow$ total number of devices in bag i

Output: S - set of indices of bags to which devices were added

1: Let H be a MIN-HEAP designed to store bags in such a way that the element at the top, represented as $[nw_0, 0, t_0]$ possesses the smallest value for the expression $(nw + 1)/(t + 1) - nw/t$ when compared to all other elements. In case of a tie, use the index of the items.
2: Initially $S \leftarrow \emptyset$
3: **while** $k > 0$ **do**
4: $b \leftarrow$ pop a bag from H
5: $S \leftarrow S \cup \{index \text{ of } b\}$
6: $b[nw] \leftarrow b[nw] + 1$
7: $b[t] \leftarrow b[t] + 1$
8: Add b back to the heap H
9: $k \leftarrow k - 1$
10: **end while**
11: **return** S

3.4.1 Analysis

1. The initializing of MIN-HEAP in step 1 takes $\theta(n)$ time.

2. The while loop runs for k times and involves pop and push operations, which take $\theta(\log n)$ time. So the overall complexity of while loop is $\theta(k \log n)$.

3. Hence the overall complexity of the algorithm is $\theta(n + k \log n)$

3.4.2 Supporting example

1. Consider 2 bags (nw_i, t_i) : b1:(4,6), b2:(6,8) and $k = 2$
2. Strategy2 will choose the bag b1 since its percentage of working devices is increasing the most $((nw_i + 1/t_i + 1) - (nw_i/t_i) = 4.76\%)$ and will add the new device to it.
3. Now the second device will again be added to b1 since its percentage of working devices is increasing the most $((nw_i + 1/t_i + 1) - (nw_i/t_i) = 3.5\%)$.
4. Finally, b1:(6,8), b2:(6,8)
5. We can see that the average percentage of working devices in the bags increases from 70.8% to 75%. Hence, this strategy works optimally for the above given example.

3.4.3 Proof of Optimality

1. Let S' be our strategy-4 algorithm and S be an optimal algorithm for this problem, which has n bags, namely $b_i, b_j \dots b_n$, and k new working devices are to be added to the bags. As shows in the class, it is possible to change the optimal to output to the output of strategy-4 by performing some inversions. Let b_i, b_j be an inversion in S from S', such that
$$\frac{nw_i + 1}{t_i + 1} - \frac{nw_i}{t_i} < \frac{nw_j + 1}{t_j + 1} - \frac{nw_j}{t_j}$$
(expression 1)
2. Now, let's assume that a new device is getting added to b_i, as it precedes b_j due to our inversion.
3. Hence, the final average percentage value of working devices among all the bags will be
$$\frac{\frac{nw_i+1}{t_i+1} + \frac{nw_j}{t_j} + \dots + \frac{nw_n}{t_n}}{n}$$
(expression 2)

4. Next, if we swap b_i and b_j in the order of bags, we'll get the order obtained through our S' algorithm $(b_j, b_i, ...b_n)$. To prove that S' is optimal, we must show that the average percentage does not decrease after this swapping operation.

5. In this order, a new working device will be added to bag b_j, instead of bag b_i. The new final average percentage value of working devices among all the bags will be

$$(\frac{nw_j + 1}{t_j + 1} + \frac{nw_i}{t_i} + ... + \frac{nw_n}{t_n})/n$$

   (expression 3)

6. To prove: expression 3 $\geq$ expression 2

7. As the new working devices are getting added to bags b_i and b_j, so the percentage of working devices in rest of the bags will remain the same before and after the swapping operation.

8. Hence, finally, we need to prove that

$$\frac{nw_j + 1}{t_j + 1} + \frac{nw_i}{t_i} \geq \frac{nw_i + 1}{t_i + 1} + \frac{nw_j}{t_j}$$

9. We can rewrite this as

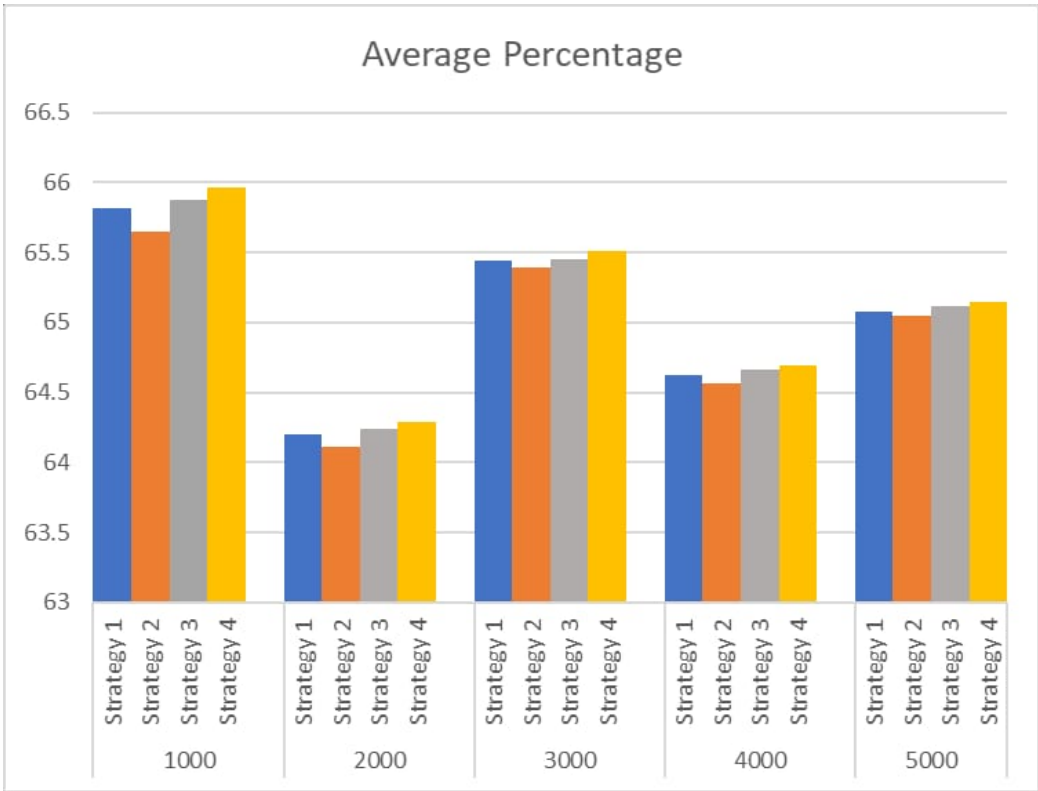
$$\frac{nw_j + 1}{t_j + 1} - \frac{nw_j}{t_j} \geq \frac{nw_i + 1}{t_i + 1} - \frac{nw_i}{t_i}$$

10. From expression 1, this condition is true.

11. Hence, our S' algorithm is optimal.

4 Experimental Comparative Study

Upon creating randomly generated input files of sizes $n = 1000$ to 5000 and for a fixed value of $k = 99000$, we obtained the average percentages for all the strategies using the input files of above-mentioned sizes. Upon plotting them, we got the graph as below:



From the above graph we can see that strategy 4 yields the maximum average percentage and hence is best among all four strategies.

5 Conclusion

The implementation of all the four strategies were quite similar except for the comparator key used for picking the element from the heap. In the experimental study, we found out that the strategy 4 is optimal, closely followed by strategy 3. Strategy 2 yields the least average percentage. We found that strategy 1, strategy 2 and strategy 3 are not optimal using counter

examples. But this is not sufficient to prove that strategy 4 is optimal. We used a proof technique that was taught in class, to arrive at a conclusion that strategy 4 is the optimal strategy. We got to learn a lot about implementation of greedy strategies and their proof of optimality techniques. Finding the counter examples for each strategy was a bit challenging and required a lot of brainstorming, but the entire process gave a great learning experience around logical thinking and mathematical proofs.