Knapsack problem

1. This knapsack problem includes eleven different items. Each item has an id, benefit, and weight.
2. The representation of the solution looks like this:

*x = {x1, x2,..,..,xn}*

n = 11

1. What we want to maximize in this problem, is the benefit. We always look after the item which has the best benefit.
2. The knapsack problem is restricted to each items weight because the knapsack has a limit of how much weight it can carry.
3. In the knapsack-program, we are using two different algorithms, breadth-first, and depth-first, to find the best path.
4. **Comparison of the time expended by the algorithms.**

|  |  |  |
| --- | --- | --- |
| Round | DFS | BFS |
| 1 | 12 ms | 6 ms |
| 2 | 7 ms | 6 ms |
| 3 | 5 ms | 5 ms |
| 4 | 6 ms | 5 ms |
| 5 | 5 ms | 5 ms |
| 6 | 9 ms | 7 ms |

As we see in the table above in almost all cases, *Breadth\_First* executes faster than *Depth\_First*. And in those cases, it is not faster, both algorithms have the same execution time. In the table above, it is two times out of 6 possible times that *BFS* and *DFS* have the same execution time, which makes *BFS* faster in 33 % of the cases.

1. **Comparison of the space used in memory at a time by the algorithms.**

The algorithms are using almost the same amount of memory, but *Depth\_First* are using a very small amount more. *Breadth\_First* are using 492,06 kb and *Depth\_First* are using 492,09 kb. So *Depth\_First* are only using 0,3 more kb.

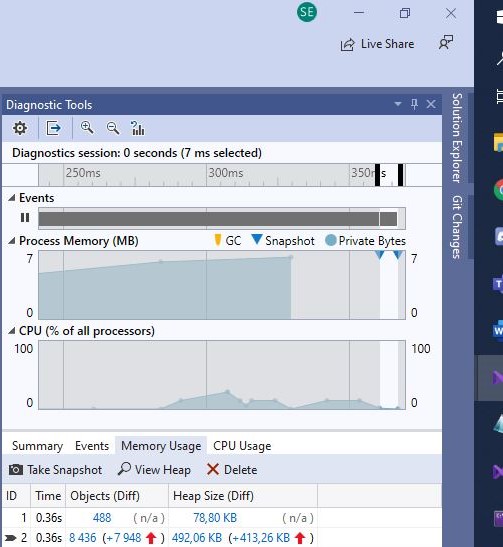
****

Figure 1: Breadth\_First

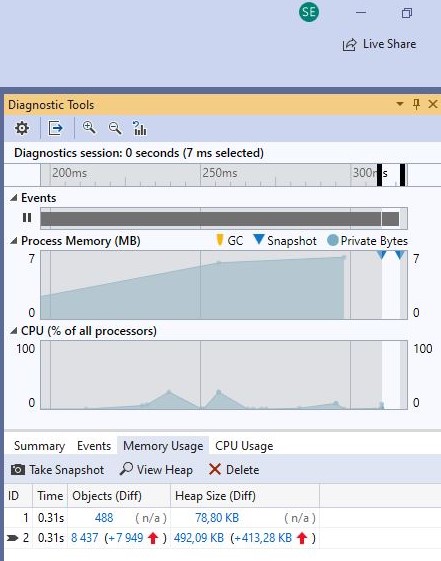
****

Figure : Depth\_First