Hiranmaya Gundu |Advanced Algorithms | UE15CS311

Data Table

Effect of changing factors

# Results of Experiment

For 50:50 ratio of inserts to deletes, all time in ms.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |
| **Insert Growth Factor** | **Max(Insert)** | **Min(Insert)** | **Avg(Insert)** | **Max(pop)** | **Min(pop)** | **Avg(Pop)** | **Avg(Ops)** |
| 2 | Pop(0.25) - 0.004398 | 0.000008 | 0.0001192 | 0.00428 | 0.0000088 | 0.0001053 | 0.0001122 |
| Pop(0.5) - 0.004598 | 0.000034 | 0.0003509 | 0.00181 | 0.0000063 | 0.00011433 | 0.0002326 |
| Pop(0.75) - 0.005697 | 0.000025 | 0.0008754 | 0.001137 | 0.0000069 | 0.00013916 | 0.0005075 |
| 3 | Pop(0.25) - 0.003996 | 0.0000067 | 0.0001295 | 0.00728 | 0.0000045 | 0.0001353 | 0.00018325 |
| Pop(0.5) - 0.004098 | 0.0000046 | 0.0002312 | 0.00981 | 0.0000079 | 0.00031425 | 0.0002727 |
| Pop(0.75) - 0.004597 | 0.0000087 | 0.0002377 | 0.01137 | 0.0000392 | 0.0006234 | 0.0004305 |
| 1.5 | Pop(0.25) - 0.051902 | 0.000105 | 0.000278002 | 0.00521 | 0.0000085 | 0.0001003 | 0.00014401 |
| Pop(0.5) –  0.069682 | 0.000455 | 0.000298021 | 0.00881 | 0.0000054 | 0.00021015 | 0.0002542 |
| Pop(0.75) - 0.043932 | 0.000325 | 0.000341920 | 0.00787 | 0.0000069 | 0.0004784 | 0.0004101 |
| 1.25 | Pop(0.25) - 0.001895 | 0.0000031 | 0.00000456 | 0.00567 | 0.0000034 | 0.0000973 | 0.0000509 |
| Pop(0.5) –  0.0039682 | 0.0000028 | 0.00000480 | 0.00511 | 0.0000091 | 0.00012985 | 0.0000673 |
| Pop(0.75) - 0.00543932 | 0.0000050 | 0.00000679 | 0.00682 | 0.0000082 | 0.0001108 | 0.0000587 |
| 1.75 | Pop(0.25) - 0.005597 | 0.0000084 | 0.0001505 | 0.00767 | 0.0000025 | 0.0001571 | 0.0001538 |
| Pop(0.5) –  0.0067612 | 0.0000171 | 0.0002387 | 0.00911 | 0.0000063 | 0.0001981 | 0.0002184 |
| Pop(0.75) - 0.004137 | 0.0000112 | 0.0003339 | 0.01382 | 0.0000097 | 0.0003008 | 0.0003173 |

For 3;2 ratio of inserts to deletes, all time in ms.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |
| **Insert Growth Factor** | **Max(Insert)** | **Min(Insert)** | **Avg(Insert)** | **Max(pop)** | **Min(pop)** | **Avg(Pop)** | **Avg(Ops)** |
| 2 | Pop(0.25) - 0.020636 | 0.0000084 | 0.00015413 | 0.002406 | 0.0000088 | 0.0001322 | 0.0001437 |
| Pop(0.5) - 0.004598 | 0.0000092 | 0.00017886 | 0.003312 | 0.0000097 | 0.0001960 | 0.0001857 |
| Pop(0.75) - 0.005697 | 0.000087 | 0.00022791 | 0.003109 | 0.0000032 | 0.0001306 | 0.0002806 |
| 3 | Pop(0.25) - 0.010343 | 0.0000087 | 0.00019309 | 0.002423 | 0.0000088 | 0.0001353 | 0.00016997 |
| Pop(0.5) –  0.01398 | 0.0000096 | 0.00024012 | 0.003493 | 0.0000079 | 0.00021425 | 0.0002297 |
| Pop(0.75) –  0.02527 | 0.0000097 | 0.00028940 | 0.003552 | 0.000039 | 0.0002234 | 0.0002630 |
| 1.5 | Pop(0.25) –  0.03303 | 0.0000031 | 0.00019841 | 0.002209 | 0.0000085 | 0.0001203 | 0.0001671 |
| Pop(0.5) –  0.03998 | 0.0000082 | 0.000223298 | 0.003870 | 0.0000084 | 0.00021015 | 0.0002238 |
| Pop(0.75) - 0.041702 | 0.0000091 | 0.000293510 | 0.003787 | 0.0000060 | 0.0002784 | 0.00028842 |
| 1.25 | Pop(0.25) - 0.058998 | 0.0000084 | 0.00050787 | 0.002300 | 0.0000034 | 0.0001973 | 0.0003836 |
| Pop(0.5) –  0.060614 | 0.0000078 | 0.00058912 | 0.003511 | 0.0000088 | 0.00019985 | 0.0004334 |
| Pop(0.75) - 0.064253 | 0.0000090 | 0.00056512 | 0.002682 | 0.0000054 | 0.0002108 | 0.0004233 |
| 1.75 | Pop(0.25) - 0.027735 | 0.0000084 | 0.00019247 | 0.002446 | 0.0000088 | 0.0001571 | 0.0001784 |
| Pop(0.5) –  0.033142 | 0.0000091 | 0.000218712 | 0.002911 | 0.0000081 | 0.0001981 | 0.0002104 |
| Pop(0.75) - 0.039021 | 0.0000072 | 0.000267534 | 0.003382 | 0.0000087 | 0.0002508 | 0.0002608 |

For 4;2 ratio of inserts to deletes, all time in ms.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |
| **Insert Growth Factor** | **Max(Insert)** | **Min(Insert)** | **Avg(Insert)** | **Max(pop)** | **Min(pop)** | **Avg(Pop)** | **Avg(Ops)** |
| 2 | Pop(0.25) - 0.029863 | 0.000009 | 0.0002552 | 0.000616 | 0.0001 | 0.00000884 | 0.0001730 |
| Pop(0.5) –  0.024178 | 0.0000092 | 0.0002896 | 0.000285 | 0.0000062 | 0.0001425 | 0.0002405 |
| Pop(0.75) - 0.021697 | 0.000087 | 0.00022311 | 0.008266 | 0.0000052 | 0.0001442 | 0.0001968 |
| 3 | Pop(0.25) –  0.018031 | 0.0000048 | 0.00013835 | 0.000695 | 0.0000082 | 0.00001456 | 0.00019704 |
| Pop(0.5) –  0.01398 | 0.0000066 | 0.00014112 | 0.00657 | 0.0000056 | 0.00011425 | 0.0001321 |
| Pop(0.75) –  0.02137 | 0.0000057 | 0.00018910 | 0.00890 | 0.000089 | 0.0001244 | 0.0001363 |
| 1.5 | Pop(0.25) –  0.01665 | 0.0000049 | 0.0001559 | 0.000759 | 0.0000085 | 0.0000093 | 0.0001070 |
| Pop(0.5) –  0.01928 | 0.0000092 | 0.0002398 | 0.007870 | 0.0000074 | 0.00021015 | 0.0002299 |
| Pop(0.75) –  0.01702 | 0.0000091 | 0.00029503 | 0.008797 | 0.0000090 | 0.0002784 | 0.0002894 |
| 1.25 | Pop(0.25) –  0.48568 | 0.0000080 | 0.0011198 | 0.006300 | 0.0000054 | 0.00001973 | 0.002894 |
| Pop(0.5) –  0.50614 | 0.0000078 | 0.0028212 | 0.006521 | 0.0000068 | 0.00019385 | 0.001881 |
| Pop(0.75) –  0.60253 | 0.0000094 | 0.0026512 | 0.009682 | 0.0000084 | 0.0002998 | 0.001867 |
| 1.75 | Pop(0.25) –  0.03259 | 0.0000092 | 0.0003338 | 0.006446 | 0.0000083 | 0.00001571 | 0.0002277 |
| Pop(0.5) –  0.033142 | 0.0000091 | 0.00021198 | 0.009901 | 0.0000089 | 0.0002211 | 0.0002203 |
| Pop(0.75) - 0.039021 | 0.0000082 | 0.00026342 | 0.008882 | 0.0000080 | 0.0002558 | 0.0002608 |

# Conclusion:

From the data we can see that there is so substantial increase in performance for any growth factor greater than 1.5. However, there is a substantial drop in performance at 1.25 as growth factor.

We also see a small but noticeable increase in pop performance at 0.25 compared to the other factors.

So the best combination is Growth Factor set at 1.5 and Shrink Factor set at 0.25.

# Points to Ponder:

- What happens to the pointers, references and iterators we may have to the element of the dynamic table on resizing?

We must assume every time we insert or pop a value that all previous references and iterators are now invalid.

- Could you come out with a strategy for avoiding this problem?

To avoid the problem we must always use the pointer stored in the structure each time we access the dynamic table. Iterators for the vector should be defined as a structure with the pointer in the table as one field and offset value as the other field. Since we are storing the offset value instead of actually incrementing the pointer, on inserting additional values, we can still start the iterator at the required position. If values are popped out, then the access is invalid, as it should be.

- How to make out whether a given block is garbage if the pointer only points to the beginning of the block?

Since we always have the top value in the structure, easiest way is to check if top is -1.

- The pointers can point to any element in the block?

If there are any pointers pointing directly to any element in the block, they cannot be used after an insert or pop as the array location might have changed. Hence the offset has to be stored each time instead.