Freya Varez

9/25/2018

Cpts 223

PA3 Report

**Problem statement:**

The goal of this project is to test the efficiency of a std∷vector and std∷list in carrying out a variation of the Josephus problem. In this problem there are N people passing around a potato, each round, the potato is passed M times in the same direction (where M and N are prompted from the user) the last person with the potato loses, and is dismissed from the game (deleted from the std∷vector or list respectively).

**Algorithm design**:

In the source code specific to the classes, the following implementations were used:

* + ListMyJosephus(int N, int M): This implementation builds a list/vector of size N (specified by the user) using the built in push\_to\_back() function.
  + void init(int N, int M): \*Same implementation as constructor above.
  + void clear()

Use built-in functions within std∷vector and std∷list

* + int currentSize()
  + bool isEmpty()

void eliminateNext(): Moves directly to the address of the loser using pointer arithmetic, taking into account loser = (M + previous loser) % currentSize();

With this implementation, loser is updated each round (each time eliminateNext() is called). Runtime efficiency is achieved by accessing the address of the Person to be deleted instead of iterating through the list, leading to a constant runtime.

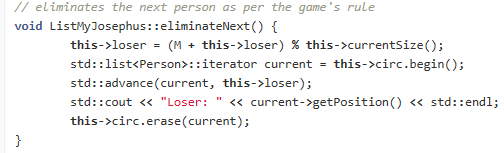
The ‘circular’ list is simulated with the use of mod (%) based on the current size of the list during the round.

* + void printAll(): Simple design, uses a for-each loop to iterate through the list/vector, printing each Person→position. Linear runtime based on the size of the list/vector.
  + void testList(): Implemented with a basic for loop running N - 1 times (where N is the size of the list, aka the population of players). Therefore the time complexity is .

**Experimental setup:**

Two experiments were done with 18 samples. With 9 having fixed N and varying M in powers of 2 (see *figure 1*); and 9 additional samples with fixed N = 512, and varying M in powers of 2 (see *figure 2*).

Note that the elimination time is not included, as this time is simply the constant time to perform a mod (%) calculation and directly access a handful addresses (See *figure 4* below), as such the elimination time is much too small to be recorded using microseconds and remains constant with an increase of the size of lis/vector.



*Figure 4: eliminateNext() algorithm*

The program was compiled and expire experimentally using a Unix system through the EECS servers and using a g++ compiler

**Results:**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| M = 3 | 4 | 8 | 16 | 32 | 64 | 128 | 256 | 512 | 1024 |
| List | 16 | 9 | 22 | 17 | 31 | 92 | 515 | 2040 | 5244 |
| Vector | 10 | 5 | 16 | 12 | 21 | 60 | 274 | 869 | 1490 |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |

*Figure 1: Fixed N = 3, varying M*

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| N = 512 | 4 | 8 | 16 | 32 | 64 | 128 | 256 | 512 | 1024 |
| List | 1070 | 1021 | 1010 | 1010 | 2029 | 2083 | 1873 | 2005 | 2161 |
| Vector | 404 | 405 | 390 | 385 | 726 | 943 | 728 | 816 | 886 |

*Figure 2: Fixed N = 512, varying M*

**Experimental Results & Discussion:**

The implementations for list and vector were quite surprising. In both cases, the std∷list had a greater overall time complexity than the std∷vector, this is perhaps because the vector is a random access iterator, thus its implementation of advance() runs in constant time, whereas std∷list runs advance() in linear time.

**Discussion:**

Independent of the input, vector ran with a more efficient time-complexity as seen in *figure 2* and *figure 3*. These results follow theoretical expectations to a point, however *figure 3* had an interesting time complexity, seemingly linear, but with a ‘bump’ despite multiple tests with the same inputs.