CS345 Programming Assignment 1

Ayush Agarwal, 13180

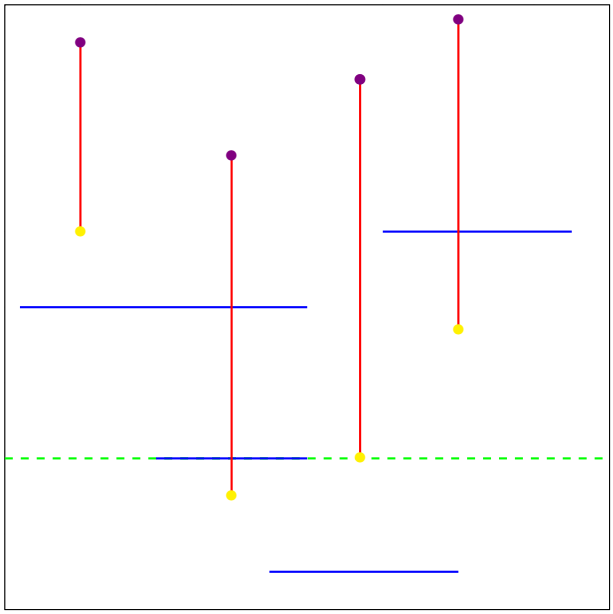
M.Arunothia, 13378

1 Red-Blue line segments

1.1 Algorithmic Description

For the given $2 * n$ line segments (n vertical red and n horizontal blue) (inside the square defined by (0,0) and (1,1)) our implementation does the following to find the number of intersection.

- We use a line sweep along the Y-direction.



- The green dashed line in the above figure shows our sweep line. This basically means that the horizontal blue lines are processed in increasing y co-ordinates.

- All yellow points (bottom point of red lines) that lies below or on the sweep line indicate those red lines that can potentially cause an intersection with the blue line. So, the x-coordinates of these yellow points are inserted into the BST maintained.
- All violet points (top point of red lines) that lies above the sweep line indicate those red lines that can no longer cause an intersection with the blue line. So, the x-coordinates of these violet points are deleted from the BST maintained.
- Now the number of intersections of this blue line (say is defined by $(x_1, y), (x_2, y)$) are found by finding the number of nodes in the BST having x in the range $[x_1, x_2]$.
- To optimize the above implementation, the BST was augmented by subtree size along with usual node data.
- Two implementations of the range count was done, with first one being submitted -
  - By finding count of $x > x_2$ and $x >= x_1$ and by taking their difference for the required range.
  - by LCA method

1.2 Implementation Description

- **Language of Implementation** - C++
- **Number of lines of Code** - 190 lines (including time and random generator functions)

- **Configuration of System used for Experiment**

```
$ lscpu
Architecture:          x86_64
CPU op-mode(s):        32-bit, 64-bit
Byte Order:            Little Endian
CPU(s):                4
On-line CPU(s) list:   0-3
Thread(s) per core:    1
Core(s) per socket:    4
Socket(s):             1
NUMA node(s):          1
Vendor ID:             GenuineIntel
CPU family:            6
Model:                 23
Stepping:              10
CPU MHz:               1998.000
BogoMIPS:              5302.48
Virtualization:        VT-x
L1d cache:             32K
L1i cache:             32K
L2 cache:              2048K
NUMA node0 CPU(s):     0-3
```

- **Number of repetitions** made for a given n is $10^7/n$.
- **Range of** n is $\{1, 10, 10^2, 10^3, 10^4, 10^5, 10^6, 10^7\}$

Result Table					
n	Time(seconds)	No. of Intersections	$n^2/9$	$n * \log n$	$(10^7 * Time/n * \log n)$
1	0	0.1111	0.11	0	
10	$4.00 * 10^{-6}$	11.1052	11.11	33.2	
10^2	$7.00 * 10^{-5}$	1110.92	1111.11	664	
10^3	0.0011	111082	111111.11	9960	3000
10^4	0.015	$1.11 * 10^7$	$1.11 * 10^7$	132800	40000
10^5	0.25	$1.11 * 10^9$	$1.11 * 10^7$	$1.66 * 10^6$	
10^6	4.8	$1.11 * 10^{11}$	$1.11 * 10^7$	$1.992 * 10^7$	
10^7	62	$1.11 * 10^{13}$	$1.11 * 10^7$	$2.324 * 10^8$	

1.3 Plot

1.4 Inferences

- From Result Table, we can see that execution time of the algorithm is $O(n * logn)$
- From Result Table, we can see that on an average, the number of intersections of the red-blue line segments when generated uniformly randomly in $(0, 1)$ is $n^2/2$

1.4.1 Bonus Question Proof

From our uniform generation we have that $(0 <= x_1 <= 1)$, $(0 <= x_2 <= 1)$, $(0 <= y_1 <= 1)$, $(0 <= y_2 <= 1)$, $(0 <= x <= 1)$ and $(0 <= y <= 1)$ are random variables with probability distribution function as 1 in $[0, 1]$ and 0 otherwise. Therefore, the probability of x lying between $[x_1, x_2]$ will be (taking $x_1 <= x_2$) -

$$\int_0^1 \int_{x_1}^1 \int_x^1 1 * dx_2 \text{ * dx * } dx_1 = 1/6$$

Similarly for $x_1 >= x_2$ it will be $1/6$. Implying overall probability of x lying between x_1 and x_2 is $1/3$.
Blue and Red line segments will intersect when

$$x \text{ lies between } x_1 \text{ and } x_2$$
$$\text{and}$$
$$y \text{ lies between } y_1 \text{ and } y_2$$

Implies probability of one blue and one red line segment intersection is -

$$= 1/3 * 1/3 = 1/9$$

As x and y are independent random variables here.

There are n^2 pairs of $(red, blue)$ line segments. Implying on an average when n is pretty large, the number of intersections should be

$$= n^2 * 1/9$$
$$= n^2/9$$

- We tried random generation also based on the constraints -
  - $0 <= x_2 <= 1$, $0 <= x_1 <= x_2$ and $0 <= x <= 1$
  - $0 <= y_2 <= 1$, $0 <= y_1 <= x_2$ and $0 <= y <= 1$

  In this case we got average number of intersections $= n^2/16$, which can again be proved like above and is also very intuitive.

```
algorithms $./a.out
1
Averaged number of intersections for n = 1 is 0.062845
Brute force answer for n = 1 is 0.062845
algorithms $./a.out
10
Averaged number of intersections for n = 10 is 6.255370
Brute force answer for n = 10 is 6.255370
algorithms $./a.out
100
Averaged number of intersections for n = 100 is 624.210300
Brute force answer for n = 100 is 624.210300
algorithms $./a.out
1000
Averaged number of intersections for n = 1000 is 62456.021000
Brute force answer for n = 1000 is 62456.021000
algorithms $
```

- If you notice the output shown above, the answer is matching exactly with the brute force answer, indicating the code should be right. But, this code does NOT work properly with the corner case of overlapping verticals and it got a WA on judge. This should be because, when double/float values are randomly generated it is very unlikely that they end up being same!!
- Though we are NOT balancing the BST after any insertion, this implementation shows that running time is indeed in $O(n * log(n))$, showing the power of a randomized input.