REPORT

NAME : LAKSHMI MADHURI YARAVA
ROLL : 2018101116

1. Running of code :

1. Give the number of nodes as input

2. for each node give the number of elements followed by all the elements in that node that are to be pushed

3. give the number of queries for operations enque and deque

4. for each query give the number of elements to be dequed or enqued

5. give the elements to be pushed or popped followed by the index of node at which it is going to be present

6. the element must be present in the queue at that particular time else error is thrown and exited if it is tried to deque

7. all the indices to be given are 1 based

2. Explanationd and Working :

1. when all the elements in the node are given all the elements then pushing starts

2. **Here threading is used ,** where every is pushed into a thread such that **all the elements in that node are pushed** in the thread it self after that all the threads will be detached

3. Then the query starts it asks for multiple queries where in each query we can give multiple elements to enque or queue which happens parellely

4. so once after the programe has taken all the queries it exits and every time the new queue is printed so as to put it in track of.

5. **Here at some places where the threading is used i have used some delays also because when the threading is used because of its speed the vector's modifiction is not reflected in time so that printing gives the wrong vector after the updates in this way delay is useful .**

3. Data Structures Used :

1. for heap : an array of vectors to keep track of it

2. main queue : vector of vectors to make it simpler for popping at certain indices too else the complexity will raise a lot

3. for storing the queries : pair of vectors

4. Complexity :

1. **for enque : O(q+1)** where q is number of total elements ,and they **have to be pushed into the thread** here concurrency is used so that all the elements will be pushed at a time so that practically it is **O(n) and here n threads are used in the programe .**

For pushing operation initially it is theretically O(n+m) where n is number of nodes ie. For pushing into threads and m is maximum number of elements in the nodes or average number of nodes

g++ cds.cpp -lpthread

./a.out

2. **for deque : O(n+1) practically similar to above if pushing into the thread is neglected as the time complexity**

So here for initial creation of queue threading will be useful but not in queries because it is as same complexity as direct method but for concurrency it is used for now .

5. Main theme and langauge :

1. **Threading is used in cpp** because the java is not strcutured with the pointers so its easier to do it in cpp

2. **Here delays are used for obseving the outputs and to make sure that the changes in the vectors and heaps reflect perfectly in the programe**

3. **To know more about use of delays remove the delays and try to run the code then we will observe that most of the times the vector is not updtaed but it is printing the previous vectors which inturn gives segmentation faults sometimes .**

4. Delays are used only in threading but not in the other code which runs sequentially so no problem occurs while printing it updates good

6. Wind up:

1. The main theme is using **THREADING** which are very fast in concurrency

2. Because of the pointers issue in java , Threading is done in cpp

7. Instructions to compile and run the code :

To compile : **g++ --std=c++14 cds.cpp -lpthread**
(-lpthread is an extension must include this and ignore the warnings it doesnot harm the programe)

To run : ./a.out

required gcc version >= c++14

CONCLUSIONS :

Serial no	Number of nodes	Avg no of elements	Delay for each iter (ms)	Time Without threading	Time in threading (milli sec)	Difference
1.	100	300	0.01	8.55453	2.77835	~(5.8)
2.	500	300	0.01	29.2752	9.31419	~(20)
3.	50	3000	0.01	37.5207	1.96919	~(36)
4.	100	500	0.01	12.5765	3.83965	~(9)
5.	300	500	0.05	31.0257	5.73413	~(25.5)
6.	300	4000	0.05	334.605	12.1008	~(320)
7.	30	40	0.05	0.384233	0.931467	~(-0.6)
8.	30	40	0.001	0.384233	0.681467	~(-0.28)
9.	200	12	0.001	0.915582	3.70447	~(-3)
10.	2000	125	0.05	45.7845	26.4583	~(20)

Here we have observed that threading takes more time than the original one in some cases ie. For the smaller number of average elements , this is because :

1. Here very small number of operations are going on in the thread so that it is taking time for the things such as insertion into the thread and detaching from the thread.
2. For larger number of elements threading hgave significantly better results because more operations are done parellelly.

Remarks :

1. In these type of algorithms when only small number of operations to be done parellelly threading is not much useful but it will be useful when the operations are more in each thread .

Paper Link :

https://people.csail.mit.edu/shanir/publications/Baskets%20Queue.pdf