Name: Eliza Dhungel

Date: 12/17/2018

**Greedy algorithms and Genome Rearrangements:**

Both SimpleReversalSort and BreakPointReversalSort are implemented in c++ and compiled through hopper in the university computer. So, the program can be compiled in hopper or any other system in command line in the directory where these files are available using

* g++ -std=c++11 SimpleReversalSort.cpp and
* g++ -std=c++11 BreakPointRevesalSort.cpp

and runned using

* ./a.out

The simple reversal sort adds zero at the beginning of permutation and keeps reversing the strands and increasing the size of the sorted prefix until all the integers are sorted. The input must be non-negative non-zero number, permutation of 1…. N. Character input signal end of input.

An example of simple reversal sort is as follows;

Input:

3

5

4

9

8

6

1

7

2

10

P

Output

016894537210

012735498610

012375498610

012345798610

012345689710

012345679810

The Break Point reversal sort adds zero and N+1 element at either end of input vector and sorts input by finding the decreasing strand with least inter, i and reversing the sequence between i and i-1 integer. All singletons are treated as decreasing strand. When decreasing strand is not present, the algorithm flips an increasing strand and carry on decreasing breakpoints by reversing sequence between i-1 and i in decreasing strands until all breakpoints are resolved. Some examples of breakpoint reversal sort are as follows.

Case 1: Only decreasing strand

Input

3

2

1

6

5

4

9

8

7

p

Output:

0 1 2 3 6 5 4 9 8 7 10

0 1 2 3 4 5 6 9 8 7 10

0 1 2 3 4 5 6 7 8 9 10

Case 2: decreasing strand with singleton in beginning

3

6

5

4

9

8

7

2

1

p

Output:

0 1 2 7 8 9 4 5 6 3 10

0 1 2 3 6 5 4 9 8 7 10

0 1 2 3 4 5 6 9 8 7 10

0 1 2 3 4 5 6 7 8 9 10

Case 3: All singleton input

3

6

9

2

5

8

1

4

7

p

Output:

0 1 8 5 2 9 6 3 4 7 10

0 1 2 5 8 9 6 3 4 7 10

0 1 2 7 4 3 6 9 8 5 10

0 1 2 3 4 7 6 9 8 5 10

0 1 2 3 4 5 8 9 6 7 10

0 1 2 3 4 5 9 8 6 7 10

0 1 2 3 4 5 9 10 7 6 8

0 1 2 3 4 5 6 7 10 9 8

0 1 2 3 4 5 6 7 8 9 10

Case 4: All increasing strip input

4

5

6

1

2

3

7

8

9

p

Output:

0 4 5 6 3 2 1 7 8 9 10

0 1 2 3 6 5 4 7 8 9 10

0 1 2 3 4 5 6 7 8 9 10

Case 5: Increasing strip with singleton in middle

4

5

6

7

8

9

3

1

2

10

11

12

p

Output:

0 4 5 6 7 8 9 10 2 1 3 11 12 13

0 1 2 10 9 8 7 6 5 4 3 11 12 13

0 1 2 3 4 5 6 7 8 9 10 11 12 13

Case 6: Increasing, Decreasing strip and singleton in input

6

5

4

7

8

9

1

2

3

10

15

14

11

12

13

P

Output:

0 6 5 10 3 2 1 9 8 7 4 15 14 11 12 13 16

0 1 2 3 10 5 6 9 8 7 4 15 14 11 12 13 16

0 1 2 3 4 7 8 9 6 5 10 15 14 11 12 13 16

0 1 2 3 4 5 6 9 8 7 10 15 14 11 12 13 16

0 1 2 3 4 5 6 7 8 9 10 15 14 11 12 13 16

0 1 2 3 4 5 6 7 8 9 10 15 16 13 12 11 14

0 1 2 3 4 5 6 7 8 9 10 11 12 13 16 15 14

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

There were other various cases evaluated. Systemic evaluation of runtime and memory consumption was not done. But the algorithm should run in O(n2) time.