Closest Pair Report

abax, bemn, diem, emba, mkkr, sebni

October 27, 2020

Results

Our implementation produces the expected results on all input–output file pairs.

The following table shows the closest pairs in the input files `wc-instance-*.txt`.

file	point 1	point 2	δ
2	0	1	1
6	4	0	1
14	11	9	1
30	11	9	1
62	24	11	1
126	122	91	1
254	29	16	1
1022	1005	337	1
4094	3953	2802	1
16382	11670	1043	1
65534	35326	14974	1

Implementation details

We sort once, at the beginning of the algorithm by both x, and y. This is then inserted into a data stucture (called `SortedPoints`) that can keep track of where the corresponding elements are in both lists. After that, on every recursive step, subsets are taken carefully to avoid sorting again. This is done in a linear manner.

The comparison on the merging step is bound by looking at neighbors at most 7 indexes apart in the y sorted set. It's worth mentioning that K&T proofs the correctness for a bound of at most 15, but we found on other references 7 as a better bound[1].

[1]

We add an extra check when looking 7 ahead, since we also know that we are looking for values that are closer than δ.

While this doesn't change the asymptotic analysis (it's still bounded by a constant), it may give us some speed-ups.

Here is the corresponding part of our code:

```
against = index+1
while against < len(strip)
```

```
  and against-index < BOUND
  and strip[against].y - strip[index].y < delta:
    currentPair = Pair(strip[against], strip[index])
    if best is None:
        best = currentPair
    else:
        best = min(best, currentPair)
    against += 1
```

Our running time is $O(n \log n)$ for n points. Since we have $\log(n)$ levels of recursion, each of which in total add up to n the amount of work they do (since getting sorted subsets is done linearly)

References

[1] UCSB slides on closest pair problem.
https://sites.cs.ucsb.edu/šuri/cs235/ClosestPair.pdf