COSC201 Assignment 1: Forming a pool

"Short" exemplar solution

Introduction

As a collection of circular puddles grow and merge from randomly scattered centers, how long does it take before they form one large pool? How many individual contacts between two puddles create new pools as opposed to extending an existing one? This report will address these problems, but also (and perhaps more importantly) consider the efficiency of various possible ways of doing so.

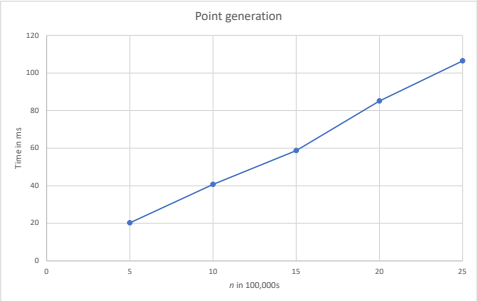
Experimental operations and their time requirements

To conduct the experiment we need to: generate n points, generate the merge-ordering list for these points, and then carry out union operations using a union-find implementation until only one group remains. We will consider the theoretical and observed time complexities of each of these phases in turn.

Point generation

The cost, in theory, of generating n points is $\Theta(n)$. The points are generated in the constructor of `Puddles.java` which assigns an array of n `Point2D` objects whose construction requires constant time per object.

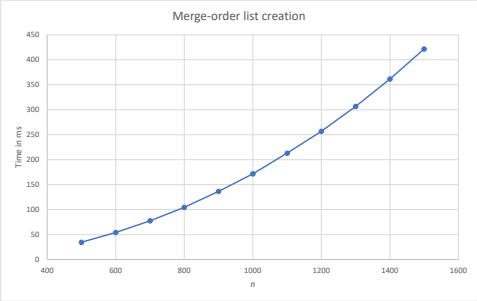
Observationally, the time required to construct n points was insignificant until n was approximately 500,000. To collect observational data, 100 trials were performed and the mean time computed for n from $500,000$ to $2,500,000$ in steps of $500,000$. These (shown below) show a clear linear trend.



Merge-order list

The theoretical complexity of generating the merge-order list is $\Theta(n^2 \log n)$. It is a list of all the pairs of point indices, sorted by distance between them. As such, it constains $n(n-1)/2$ elements (generated in nested for-loop). Sorting these elements adds an extra factor of $\log n$ (technically, $\log n^2$ but this is $2 \log n$) to the cost.

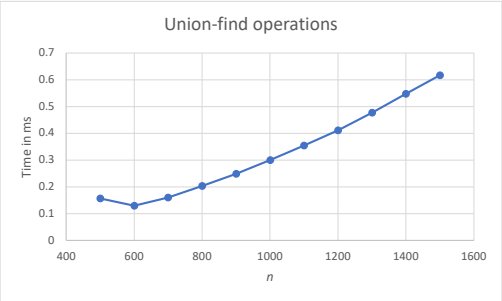
Observational data shows a clear non-linear trend. It was collected for n from 500 to 1500 in steps of 100 (again reporting the mean time in milliseconds of 100 trials).



Union operations

Union operations are carried out only to the last essential merge. There are $n - 1$ essential merges but we do not know how many superfluous merges there will be. Let t be the total number of merges carried out. Then, based on the underlying estimates for time complexity of the various union-find implementations our theoretical time estimates would be: $O(tn)$ for UF1 and UF2, and $O(t \log n)$ for UF3 and UF4. In fact, for UF1 we can certainly expect $O(n^2)$ since a superfluous union takes constant time.

Observational data for UF1 does not show a clear non-linearity, but it's not inconsistent with the observations either. Note that the times are very small, so the data should be viewed with some scepticism. In particular, the reduction from 500 to 600 is almost certainly due to random changes. Observational data was collected for n from 500 to 1500 in steps of 100 (again reporting the mean time in milliseconds of 100 trials).



Comparing the union-find implementations

The intended method of comparing the union-find implementations was to consider which value of n produced a wall-clock time of roughly one second for just the union-find phase of the main experiment. However, the memory and time requirements of the merge-order generation created a bottleneck that made this impossible. Therefore, I considered a fixed value of n, close to that bottleneck, and compared the union-find times at that common value. The value chosen was $n = 5000$ which led to a computation time of roughly 7 seconds for a full run (using any of the union-find implementations).

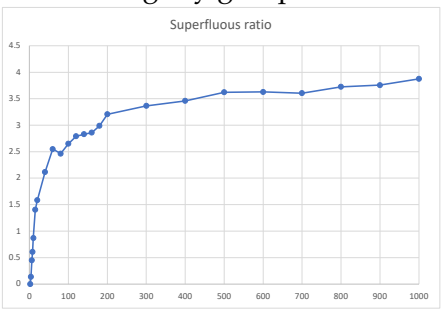
For each implementation, 100 trials were carried out with the union-find phase being timed. The mean, minimum and maximum times (in microseconds to three significant figures) are shown in the table below. Each trial had each of the four union find implementations operating on the same instance of `Puddles`. It took a little less than an hour to generate this data.

Implementation	Mean	Min	Max
UF1	6470	6160	12700
UF2	6280	2410	23100
UF3	1420	680	4850
UF4	499000	387000	542000

The most striking part of this data is the poor performance of UF4 - presumably due to the recursive implementation of the path-compression. Other than that, it does appear that UF3 provides an improvement over UF1 and UF2, which are roughly the same as one another.

Counting superfluous merges

The main purpose of this experiment was to study how the ratio between the number of superfluous merges and the number of essential merges changes as the number of puddles increases. Our observations suggest that this ratio increases at a rate that appears to be logarithmic in the number of puddles. The relevant data are shown below, with the usual 100 trials per size (reporting the mean). Because of the behaviour of the growth, we chose n that were more tightly grouped for lesser values.



The ratios for individual runs of the experiment can vary quite widely, which accounts for the lack of smoothness in the graph above. For example, at $n = 100$, I observed values of the ratio ranging from 1.05 to 7.01.

Improving merge-order computations

In the sorting phase of computing the merge-ordering list every time a pair of points is involved in a comparison the distance between the points is computed. This suggests that we might be able to improve the performance by precomputing the distances. How might we do this?

The most obvious approach would be to store a two-dimensional array of doubles called `d` where `d[i][j]` is the distance between puddle i and puddle j. This can be computed in time $\Theta(n^2)$ using a simple nested for loop. Then, in the comparator, where we currently have:

```
double dd = getPoint(o1[0]).distance2(getPoint(o1[1])) -
     getPoint(o2[0]).distance2(getPoint(o2[1]));
```

we could simply use

```
double dd = d[o1[0]][o1[1]] - d[o2[0]][o2[1]];
```

instead.

Conclusion

Except for the apparently poor performance of UF4, our observations of the time required for various parts of this experiment seem broadly in agreement with theory. As to the actual results of the experiment, it seems that the ratio of superfluous to essential merges grows slowly (apparently logarithmically) as the number of puddles increases.