RADBOUD UNIVERSITY

ALGORITHMS AND DATA STRUCTURES

Practical 1: Garbage Collection

Authors:
Rob Föllings
s4144945
Joeri Janssen
s4630610

Radboud University

Algorithm

In this algorithm we calculate if a certain number of bins can be placed on a given number of intersections and its corresponding streets. The only condition is that a bin can't be placed on an intersection when on one of its neighbours a bin is already present.

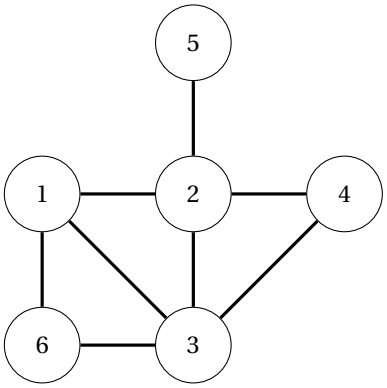
First the initialization starts. The algorithm gets via some input the number of streets, the number of intersections and the number of bins. An adjacency list is made and filled with the given streets in the next lines of input. Every intersection now has a linked list with all of its neighbours. A set is filled with all intersections and an initially empty map is created where all known sets of intersections will be stored. Lastly the solution is set to "impossible".

Now the binsAlgorithm can be run. The algorithm is recursive and with each iteration the first intersection of the remaining possible intersections is investigated. If we have placed a number of bins equal to the total number of bins that needed to be placed the solution is changed to "possible" and the algorithm terminates. If this is not the case we enter the recursive step. In the recursive step we check two situations: the situation where a bin is placed on this intersection and the situation where a bin is not placed on this intersection. If a bin is placed the intersection and its neighbours are removed from the set of possible intersections and if no bin is placed only the intersection itself is removed from the set of possible intersections. Because a lot of intersection steps are calculated multiple times the algorithm saves after every step the set of possible intersections at that step with the maximum number of bins that can be placed until there are no more intersections left. Before each recursion step the algorithm checks if the set is already present in the map of known intersections. If this is the case the value that corresponds to that set is returned. If not the algorithm enters the next recursion step.

Correctness

In the initialization the adjacency list is filled up to the number of streets and the set of possible intersections is filled with the number of intersections. Every time an intersection is checked a bin is either placed on it or not. Because of this every possibility of bin placement in the initial set of possible intersections is investigated. If a certain combination results in a possible solution the algorithm terminates. We are only interested if a possible solution exists, it doesn't matter which particular solution it is. The algorithm also stops investigating intersections if the number of bins left is larger than the number of intersections left. This situation is impossible by default because you can't place more bins on intersections than you have intersections to place bins on. Each set of possible intersections is saved with the maximum number of bins that can be placed on this set until the set is empty. This saves the algorithm from rechecking configurations it has already determined the maximum number of bins for.

Here follows an example run of the algorithm with the following graph:
Streets: 8
Intersections: 6
Bins: 4



1. Initialize → intersections (1, 2, 3, 4, 5, 6) bins 4
2. Algorithm intersections (1, 2, 3, 4, 5, 6) bins 4: Place bin on 1 → Remove 1, 2, 3, 6 → Recursion
3. Algorithm intersections (4, 5) bins 3: bins > intersections → impossible
4. Algorithm intersections (1, 2, 3, 4 ,5, 6) bins 4: Don't place bin on 1 → Remove 1 → Recursion
5. Algorithm intersections (2, 3, 4, 5, 6) bins 4: Place bin on 2 → Remove 2, 3, 4, 5 → Recursion
6. Algorithm intersections (6) bins 3: bins > intersections → impossible
7. Algorithm intersections (2, 3, 4, 5, 6) bins 4: Don't place bin on 2 → Remove 2 → Recursion
8. Algorithm intersections (3, 4, 5, 6) bins 4: Place bin on 3 → Remove 3, 4, 6 → Recursion
9. Algorithm intersections (5) bins 3: bins > intersections → impossible
10. Algorithm intersections (3, 4, 5, 6) bins 4: Don't place bin on 3 → Remove 3 → Recursion
11. Algorithm intersections (4, 5, 6) bins 4: bins > intersections → solution = impossible
12. solution = impossible

Runtime Complexity

The initialization consists of constant time operations and two for loops. One for loop is done number of streets amount of times and consists of only constant time operations. The other for loop is done number of intersections amount of times and consists of inserting values in a set. That operation is logarithmic in its size. The maximum size is equal to the number of intersections. So the initialization takes $\mathcal{O}$(intersections log intersections).

In every run of the algorithm you do a lookup in the known intersections map which is logarithmic in its size. The set of possible intersections is copied twice which takes linear time relative to its size. The linked list of neighbours of the intersection you are investigating is copied once. This takes linear time relative to the number of neighbours which is a maximum of 4. All other operations are done in constant time. The algorithm itself runs $2^{intersections}$ times because there are $2^{intersections}$ possible ways the set of possible intersections can be filled. This gives the algorithm $\mathcal{O}(2^{intersections}$ (log $2^{intersections}$ + 2 intersections + 4)) $\rightarrow \mathcal{O}(2^{intersections}$ (3 intersections + 4)).

This gives a total complexity of $\mathcal{O}$(intersections log intersections + $2^{intersections}$ (3 intersections + 4)). In reality this will be much lower however. Removing multiple neighbours, not continuing if the number of bins is larger than the number of intersections and saving previously investigated sets of intersections will make the algorithm much faster.