Algorithms for Memory Hierarchies
Lecture 5

Lecturer: Nodari Sitchinava
Scribe: Mateus Grellert

Previosly

In Lecture 3 we have shown that the orthogonal line intersection reporting problem can be solved with I/O complexity of: $\sum_{i=1}^{|H|} O\left(\log_B N + \frac{K_i}{B}\right) I/Os$ per horizontal segment, so

$$TotalComplexity = O\left(N \log_B N + \frac{K}{B}\right)$$

Today

Today we will show that this can be improved to I/O complexity of:

$$O\left(\frac{N}{B} \log_{\frac{M}{B}} \frac{N}{B} + \frac{K}{B}\right) \text{ I/Os}$$

1 Distribution Sweeping Technique

Distribution sweeping is a technique to solve batched offline problems I/O-efficiently. In batched offline problems, we are given a set of all queries upfront and we want to answer each query, but we don't care in which order the answers arrive and don't mind waiting for the answers to each query till the end of the algorithm.

Prior to understanding the Distribution Sweeping Technique, let us consider a simple problem: Batched Stabbing Query Problem (BSQP)

1.1 The Batched Stabbing Query Problem

Let's suppose we have a set of one-dimensional ranges $S = \{s_1, s_2, ..., s_N\}$, with each range in the form (x_1, x_2) and a set of queries $Q = \{q_1, q_2, ..., q_N\}$. Batched Stabbing Query Problem (BSQP) asks to report for each element $q_i \in S$, which ranges $s_j \in S$ contain q_i. Another way to look at the problem is to view the set S as horizontal segments and Q as infinite vertical lines (as depicted in Figure 1). Then the BSQP asks to report for each vertical line q_i, the segments that intersect q_i.

The following notation will be used in this explanation:

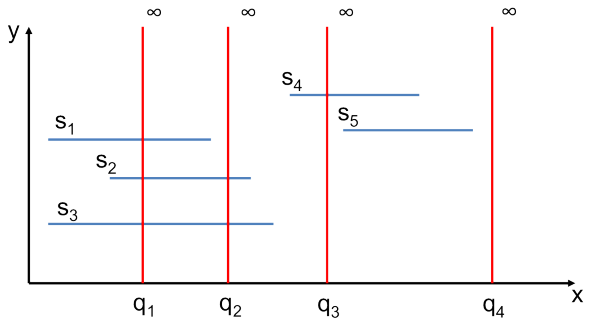

Figure 1: A set of horizontal segments stabbed by a series of vertical queries.

- $x_1(s_i)$: left x coordinate of segment i
- $x_2(s_i)$: right x coordinate of segment i
- $x(q_i)$: x coordinate of query i

Consider the following algorithm:

1. Sort objects by x coordinate (for segments, consider only the left coordinate). Considering our example in Figure 1, this should output:
   $ordList = (s_3, s_1, s_2, q_1, q_2, s_4, q_3, s_5, q_4)$
2. Apply Algorithm 1 to the sorted sequence of segments and queries.

```
Data: ordList: list of queries and segments ordered by the x-coordinate (left endpoint in
      case of segments)
for obj ∈ ordList do
    if obj is a query then
        A' = ∅;
        for each s_i ∈ A do
            if x_2(s_i) ≥ x(obj) then
                report "s_i intersects obj" and copy s_i to A';
            end
        end
        set A ← A';
    else                                                    /* obj is a segment */
        append obj at the end of A;
    end
end
```

Algorithm 1: Pseudocode for batched stabbing query problem.

Algorithm 1 performs a sweep of the sorted sequence of segments and queries and maintains the list of active segments A. A segment becomes active when the sweep reaches the left endpoint of the segment. When the sweep reaches a query, all active segments whose right endpoint's x-coordinate is greater than the query's x-coordinate, contain the query and are reported. All active segments whose right endpoint's x-coordinate is smaller than the query's x-coordinate are "deactivated" and are removed from the active list.

Let's analyze the time complexity of the Algorithm 1. Let N be the total number of segments and queries and let K_i be the number of queries that intersect each segment s_i. Each query q is

touched once (outer for loop). Each segment s_i is touched once (when added to A), reported K_i times and copied to A' K_i times. Thus, the time complexity of the algorithm is

$$\sum_{i=1}^{|Q|} 1 + \sum_{i=1}^{|S|}(1 + 2K_i) = |Q| + |S| + 2|K| \leq 2(N + K) = O(N + K)$$

Combining with the initial sorting step of N objects, we obtain the total time complexity of $O(N \log N + K)$. Note, time complexity depends on the size of the output, thus, we once again have an *output-sensitive* algorithm.

To determine the I/O complexity of Algorithm 1, we observe that copying and reporting is simply scanning of contiguous data. Thus, the I/O-complexity for this algorithm (excluding the initial sorting step) is $scan(O(N + K)) = O(\frac{N}{B} + \frac{K}{B})$. Combining with the initial sorting step of the input, the total I/O-complexity equals:

$$TotalI/O = O\left(\frac{N}{B} \log_{\frac{M}{B}} \frac{N}{B} + \frac{K}{B}\right)$$

1.2 Distribution Sweeping Framework

The BSQP works efficiently when only the horizontal segments have a limited length and the queries are considered of infinite length. However, what if that's not the case and the queries also have a specific range? In order to solve this I/O-efficiently, we use a technique called *Distribution Sweeping Framework*. For simplicity we assume that the coordinates of all objects are distinct.

The framework can be described as follows. Partition the space into $\Theta(M/B)$ vertical slabs, such that each slab contains an equal number of object endpoints with distinct x-coordinate. Sweep the objects in increasing y-coordinate, processing (e.g. reporting intersection among) the portions of horizontal objects that fully span at least one slab and distributing the copies of objects into slabs that contain the endpoints of the objects. Recursively process each slab until the slab contains at most M objects. When a slab contains only M objects (a base case of the recursion), we load the objects of the slab into memory and use internal memory algorithm to solve the problem.

Let us use distribution framework on a specific problem of *orthogonal line segment intersection reporting*: given a set of horizontal and vertical segments, report all pairs of segments that intersect.

We first divide the space into $\Theta(M/B)$ vertical slabs. Consider a single slab σ_j and only the vertical segments within σ_j and only the portions of the horizontal segments that span σ_j. In the example of Figure 2 in the third slab we only consider q_3, q_4 and portion of s_3 fully spanning the slab. We want to report intersections between these segments. (Note, that interesection between s_4 and q_4 will be reported during recursive call.) Observe, that when we only consider these objects within the third slab, the problem of reporting intersections between these objects is exactly BSQP. Thus, to report intersections at this recursive level, we sweep the objects in increasing y-coordinate and maintain $\Theta(M/B)$ active lists (one for each slab). But now, the vertical segments act as segments and horizontal segments act as queries. When the sweep encounters a bottom endpoint of a vertical segment, it adds it to the active list of the slab containing that vertical segment. We also add the vertical segment to the list of objects of the slab that contains it (for the recursive call). When the sweep encounters the horizontal segment, we scan each active list of the slabs which are spanned by the horizontal segment and report intersections as in Algorithm 1. We also add the copies of the horizontal segment into the lists of objects of the slabs which contain the endpoints (for the recursive call). After the sweep completes, we recursively call the procedure on each slab. At the base of the recursion, each slab list contains at most M objects, so we load them into memory and solve them in internal memory.

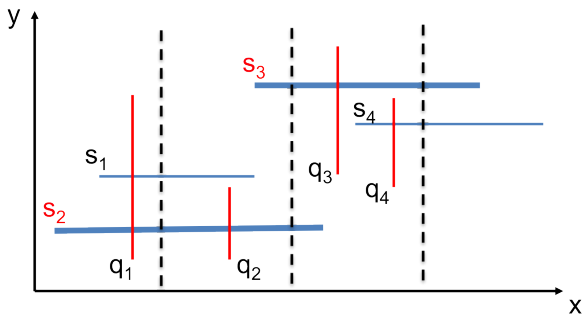

Figure 2: Similar to the BQSP, but with slabs dividing the space.

Note, that to solve a single Batched Stabbing Query Problem we need to keep only 3 blocks in internal memory: one block for set S of sorted elements, one block for list A and one block for list A'. Thus, we can solve $\Theta(M/B)$ BSQP problems simultaneously.

Note, that in each recursive call the lists of objects in each slab must be sorted in increasing y-coordinates. If we sort the objects by the y-coordinate at the beginning, the lists for each slab are also created in sorted order by increasing y-coordinate.

The last thing that we must determine is how to determine the slab boundaries at each recursive level so that each slab contains equal number of objects. This is done by sorting the end-points by the x-coordinate, picking boundaries deterministically, storing them in an array and looking up the slab boundaries in this array at each recursive level. Since in the base case each slab contains $\Theta(M)$ objects, there will be at most $O(N/M)$ slabs throughout the algorithm, and, therefore, the size of this array is at most $O(N/M)$, i.e. does not affect the asymptotic space complexity.

I/O Complexity: Let us analyze the I/O complexity of each recursive call. First, we need to read the slab boundaries. The most number of slabs is in the base case of the recursion, where each slab contains $\Theta(M)$ objects, thus, there are at most $O(N/M)$ boundaries to read. Even if we spend 1 I/O to read each boundary, we still spend at most $O(N/M) < O(N/B)$ I/Os for this step. Next, we perform the sweep solving BSQP problem. As we have seen in the previous section, it takes $O\left(\frac{N}{B} + \frac{K_k}{B}\right)$ I/Os, where K_k is the number of intersection reported at this recursive level.

Summing up over $\log_{M/B} \frac{N}{M}$ recursive levels, we get the I/O complexity of the recursive calls:

$$\sum_{k=1}^{\log_{M/B} \frac{N}{M}} O\left(\frac{N}{B} + \frac{K_k}{B}\right) = O\left(\frac{N}{B} \log_{M/B} \frac{N}{M} + \sum_{k=1}^{\log_{M/B} \frac{N}{M}} \frac{K_k}{B}\right)$$

At the base case of the recursion, we load $O(N/M)$ slabs each of size $\Theta(M)$ into internal memory (to solve using internal memory algorithm). Let K' be the number of intersections that are reported in all base cases. Then, this step takes $O(N/M \cdot M/B + K'/B) = O(N/B + K'/B)$ I/Os.

Thus, the total I/O complexity is:

$$\begin{aligned} & O\left(\frac{N}{B}\log_{M/B}\frac{N}{M}+\sum_{k=1}^{\log_{M/B}\frac{N}{M}}\frac{K_k}{B}+\frac{N}{B}+\frac{K'}{B}\right) \\ = \; & O\left(\frac{N}{B}\left(1+\log_{M/B}\frac{N}{M}\right)+\left(\frac{K'}{B}+\sum_{k=1}^{\log_{M/B}\frac{N}{M}}\frac{K_k}{B}\right)\right) \\ = \; & O\left(\frac{N}{B}\log_{M/B}\frac{N}{B}+\frac{K}{B}\right), \end{aligned}$$

where K is the total number of intersection reported over all levels, i.e. $K = K' + \sum_{k=1}^{\log_{M/B}\frac{N}{M}} K_k$.

Internal Memory Version: When we have enough space in the internal memory, we can apply the same algorithm, but using only 2 slabs instead of $\frac{M}{B}$ resulting in $O(N \log N)$ time complexity.

1.3 Distribution Sweeping Applications

This framework can be used for batched geometric problems, such as:

1. General (non-orthogonal) line segment intersection reporting
2. Orthogonal range reporting (see Figure 3)

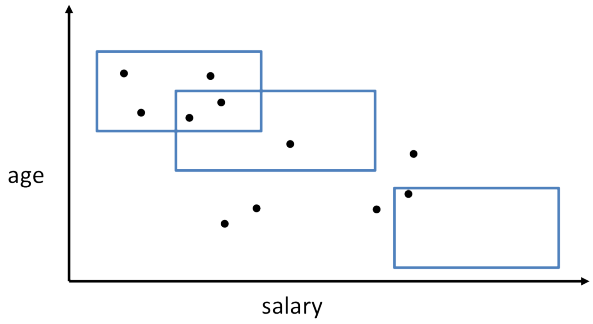

Figure 3: For each rectangle, which points fall inside of it?

3. Rectangle intersections problem – report all pairs of rectangles on the plane that overlap/intersect (by combining solutions to orthogonal line segment intersection reporting and orthogonal range reporting)
4. Two-dimensional Stabbing Queries
5. d-dimensional extensions of these problems (the I/O complexity increases by a factor of $O(\log_{M/B} N/B)$ for each dimension)
6. Two-dimensional point location/ray-shooting – given a set of non-intersecting line segments and points on the plane, report for each point the line segment directly above the point