External Memory Algorithms for Geometric Problems

Piotr Indyk

Slides partially by Lars Arge and Jeff Vitter
Compared to Previous Lectures

- Another way to tackle large data sets
- Exact solutions (no more embeddings)
External Memory Model

• Parameters:
  - $N$: Elements in structure
  - $B$: Elements per block
  - $M$: Elements in main memory
Today

- Sorting: $O(\frac{N}{B} \log_{M} N)$ time
- 1D data structure for searching in external memory (B-tree): $O(\log_{B} N)$ time
- 2D problem: finding all intersections among a set of horizontal and vertical segments: $O(\frac{N}{B} \log_{\frac{M}{B}} N)$ time
Sorting

• M/B-way merge sort:
  – Split $N$ elements into $K=M/B$ sequences
  – Sort recursively
  – Merge in $O(N/B)$ time:

• Recurrence: $T(N) = K \cdot T(N/K) + O(N/B)$
• $T(N) = O(N/B \log_K N)$
Horizontal/Vertical Line Intersection

- Given: a set of $N$ horizontal and vertical line segments
- Goal: find all H/V intersections
- Assumption: all $x$ and $y$ coordinates of endpoints different
Main Memory Algorithm

- Presort the points in y-order
- Sweep the plane top down with a horizontal line
- When reaching a V-segment, store its x value in a tree. When leaving it, delete the x value from the tree
- Invariant: the balanced tree stores the V-segments hit by the sweep line
- When reaching an H-segment, search (in the tree) for its endpoints, and report all values/segments in between
- Total time is $O(N \log N + P)$
External Memory Issues

• Can use B-tree as a search tree:
  \( \mathcal{O}(N \log_B N) \) operations
• Still much worse than the
  \( \mathcal{O}(N/B \times \log_{M/B} N) \) sorting bound
1D Version of the Intersection Problem

• Given: a set of \( N \) 1D horizontal and vertical line segments (i.e., intervals and points on a line)

• Goal: find all point/segment intersections

• Assumption: all \( x \) coordinates of endpoints different
Interlude: External Stack

• Stack:
  – Push
  – Pop

• Can implement a stack in external memory using $O(P/B)$ I/O’s per $P$ operations
  – Always keep $\leq 2B$ top elements in main memory
  – Perform disk access only when it is “earned”:
    • Read when necessary
    • Write when starting a new block

...
Back to 1D Intersection Problem

• Will use fast stack and sorting implementations

• Sort all points and intervals in x-order (of the left endpoint)

• Iterate over consecutive (end)points $p$
  – If $p$ is a left endpoint of $I$, add $I$ to the stack $S$
  – If $p$ is a point, pop all intervals $I$ from stack $S$ and push them on stack $S'$, while:
    • Eliminating all “dead” intervals
    • Reporting all “alive” intervals
  – Push the intervals back from $S'$ to $S$
Analysis

• Sorting: $O(N/B \times \log_{M/B} N)$ I/O’s
• Each interval is pushed/popped when:
  – An intersection is reported, or
  – Is eliminated as “dead”
• Total stack operations: $O(N+P)$
• Total stack I/O’s: $O( (N+P)/B )$
Back to the 2D Case
Algorithm

- Divide the x-range into $M/B$ slabs, so that each slab contains the same number of V-segments
- Each slab has a stack storing V-segments
- Sort all segments in the $y$-order
- For each segment $I$:
  - If $I$ is a V-segment, add $I$ to the stack in the proper slab
  - If $I$ is an H-segment, then for all slabs $S$ which intersect $I$:
    - If $I$ spans $S$, proceed as in the 1D case
    - Otherwise, store the intersection of $S$ and $I$ for later
- For each slab, recurse on the segments stored in that slab
The recursion

- For each slab *separately* we apply the same algorithm
- On the bottom level we have only one V-segment, which is easy to handle
- Recursion depth: $\log_{M/B} N$
Analysis

• Initial presorting: $O(N/B \times \log_{M/B} N)$ I/O’s
• First level of recursion:
  – At most $O(N+P)$ pop/push operations
  – At most $2N$ of H-segments stored
  – Total: $O((N+P)/B)$ I/O’s
• Further recursion levels:
  – The total number of H-segment pieces (over all slabs) is at most twice the number of the input H-segments; it does not double at each level
  – By the above argument we pay $O(N/B)$ I/O’s per level
• Total: $O(P/B+N/B \times \log_{M/B} N)$ I/O’s
References

• See http://www.brics.dk/MassiveData02, especially:
  – First lecture by Lars Arge (for B-trees etc)
  – Second lecture by Jeff Vitter (for distribution sweep)