Models of Computation 3: External Memory Model

Mayank Goswami

11.07.2014

Last time

There are basically two main techniques used to prove lower bounds in the cell probe model:

Last time

There are basically two main techniques used to prove lower bounds in the cell probe model:

- Communication complexity method

Last time

There are basically two main techniques used to prove lower bounds in the cell probe model:

- Communication complexity method
  1. Greedy communication complexity.
  2. Probability amplification method—Round elimination lemma.

Last time

There are basically two main techniques used to prove lower bounds in the cell probe model:

- Communication complexity method
  1. Greedy communication complexity.
  2. Probability amplification method—Round elimination lemma.
- Chronogram technique.

Last time

There are basically two main techniques used to prove lower bounds in the cell probe model:

- Communication complexity method
  1. Greedy communication complexity.
  2. Probability amplification method—Round elimination lemma.
- Chronogram technique.

Recap: Chronogram Technique

- We want to prove a $\Omega\left(\frac{\log n}{\log \log n}\right)$ lower bound on the update/query time of *any* dynamic problem.

Recap: Chronogram Technique

- We want to prove a $\Omega\left(\frac{\log n}{\log \log n}\right)$ lower bound on the update/query time of *any* dynamic problem.
- We generate a sequence of n random updates, followed by a random query.

Recap: Chronogram Technique

- We want to prove a $\Omega\left(\frac{\log n}{\log \log n}\right)$ lower bound on the update/query time of *any* dynamic problem.
- We generate a sequence of n random updates, followed by a random query.
- Looking back in time, we partition the updates into *epochs* that are exponentially growing.

Recap: Chronogram Technique

- We want to prove a $\Omega\left(\frac{\log n}{\log \log n}\right)$ lower bound on the update/query time of *any* dynamic problem.
- We generate a sequence of n random updates, followed by a random query.
- Looking back in time, we partition the updates into *epochs* that are exponentially growing.
- For a certain r, epoch i contains the r^i updates immediately before epoch $i-1$.

Recap: Chronogram Technique

- We want to prove a $\Omega\left(\frac{\log n}{\log \log n}\right)$ lower bound on the update/query time of *any* dynamic problem.
- We generate a sequence of n random updates, followed by a random query.
- Looking back in time, we partition the updates into *epochs* that are exponentially growing.
- For a certain r, epoch i contains the r^i updates immediately before epoch $i-1$.
- **For all i, the query needs to read at least one cell from epoch i with constant probability.**
- This gives a lower bound of $\Omega(\log_r n)$.

Chronogram Technique

- Clearly, information about epoch i cannot be reflected in earlier epochs, as those occurred back in time.

Chronogram Technique

- Clearly, information about epoch i cannot be reflected in earlier epochs, as those occurred back in time.
- The latest $i-1$ epochs contain $O(r^{i-1})$ updates. Let update time be t_u.

Chronogram Technique

- Clearly, information about epoch i cannot be reflected in earlier epochs, as those occurred back in time.
- The latest $i-1$ epochs contain $O(r^{i-1})$ updates. Let update time be t_u.
- This means that at most $O(r^{i-1}t_u w)$ bits are written. Let $r = Ct_u w/\log U$ for a sufficiently large C.

Chronogram Technique

- Clearly, information about epoch i cannot be reflected in earlier epochs, as those occurred back in time.
- The latest $i-1$ epochs contain $O(r^{i-1})$ updates. Let update time be t_u.
- This means that at most $O(r^{i-1}t_u w)$ bits are written. Let $r = Ct_u w/\log U$ for a sufficiently large C.
- $O(r^{i-1}t_u w) \leq \frac{r^i}{10}\log U$.

Chronogram Technique

- Clearly, information about epoch i cannot be reflected in earlier epochs, as those occurred back in time.
- The latest $i-1$ epochs contain $O(r^{i-1})$ updates. Let update time be t_u.
- This means that at most $O(r^{i-1}t_u w)$ bits are written. Let $r = Ct_u w/\log U$ for a sufficiently large C.
- $O(r^{i-1}t_u w) \leq \frac{r^i}{10}\log U$.
- Updates in epoch i contain $r^i \log U$ bits of entropy, so all the information outside epoch i only contains a constant fraction of this.

Chronogram Technique

- Clearly, information about epoch i cannot be reflected in earlier epochs, as those occurred back in time.
- The latest $i-1$ epochs contain $O(r^{i-1})$ updates. Let update time be t_u.
- This means that at most $O(r^{i-1}t_u w)$ bits are written. Let $r = Ct_u w/\log U$ for a sufficiently large C.
- $O(r^{i-1}t_u w) \leq \frac{r^i}{10}\log U$.
- Updates in epoch i contain $r^i \log U$ bits of entropy, so all the information outside epoch i only contains a constant fraction of this.
- If a random query is forced to learn information about a random update from epoch i, it is forced to read a cell from epoch i with constant probability, because the information is not available outside this epoch.

Chronogram Technique

- Clearly, information about epoch i cannot be reflected in earlier epochs, as those occurred back in time.
- The latest $i-1$ epochs contain $O(r^{i-1})$ updates. Let update time be t_u.
- This means that at most $O(r^{i-1}t_u w)$ bits are written. Let $r = Ct_u w/\log U$ for a sufficiently large C.
- $O(r^{i-1}t_u w) \leq \frac{r^i}{10}\log U$.
- Updates in epoch i contain $r^i \log U$ bits of entropy, so all the information outside epoch i only contains a constant fraction of this.
- If a random query is forced to learn information about a random update from epoch i, it is forced to read a cell from epoch i with constant probability, because the information is not available outside this epoch.
- Thus the query must make $\Omega(1)$ probes in expectation into every epoch.

Bloom Filter lower bounds

We will get lower bounds on the **approximate membership problem**.

Bloom Filter lower bounds

We will get lower bounds on the **approximate membership problem**.

- Given a set S of n elements from $\{1, \cdots, U\}$, store them in RAM so that given a query $q \in \{1, \cdots, U\}$,

Bloom Filter lower bounds

We will get lower bounds on the **approximate membership problem**.

- Given a set S of n elements from $\{1, \cdots, U\}$, store them in RAM so that given a query $q \in \{1, \cdots, U\}$,
  - If $q \in S$, the data structure must answer yes with probability 1.
  - If $q \notin S$, the data structure must answer no with probability at least $1 - \epsilon$. Thus *false positives* are allowed.

Bloom Filter lower bounds

We will get lower bounds on the **approximate membership problem**.

- Given a set S of n elements from $\{1, \cdots, U\}$, store them in RAM so that given a query $q \in \{1, \cdots, U\}$,
  - If $q \in S$, the data structure must answer yes with probability 1.
  - If $q \notin S$, the data structure must answer no with probability at least $1 - \epsilon$. Thus *false positives* are allowed.
- The parameter ϵ is called the false positive rate.

$n \log(1/\epsilon)$ lower bound.

Bloom Filter lower bounds

We will get lower bounds on the **approximate membership problem**.

- Given a set S of n elements from $\{1, \cdots, U\}$, store them in RAM so that given a query $q \in \{1, \cdots, U\}$,
  - If $q \in S$, the data structure must answer yes with probability 1.
  - If $q \notin S$, the data structure must answer no with probability at least $1 - \epsilon$. Thus *false positives* are allowed.
- The parameter ϵ is called the false positive rate.

$n \log(1/\epsilon)$ lower bound.Bloom filters achieve $n \log(1/\epsilon) \log e$.

Introduction: External Memory Model

- Sorts of extremely large size are becoming more and more common.
- Banks each night typically sort the checks of the current day into increasing order by account number.
- In many cases banks are required to complete this processing before opening for the next business day.
- These days typical file sizes are expected to contain ten million records totalling $10,000$ megabytes and current sorting methods would take most of one day to do the sorting.
- Banks would then have trouble completing this processing before the next business day!!

The Model

The following parameters are used in the external memory model, also called the disk access model (DAM):

- N: the number of records required to sort.
- M: the number of records that can fit into internal memory.
- B: the number of records that can be transferred in a single disk I/O.

Typical values: $N = 10^7$, $M = 3000$, $B = 100$.

Sorting

- **Input:** The internal memory is empty, and the N records reside at the beginning of the disk, that is, $x[i] = nil$ for $1 \leq i \leq M$ and $x[M+i] = R_i$ for $M+1 \leq i < N$.
- **Goal:** The internal memory is empty, and the N records reside at the beginning of the disk in sorted nondecreasing order, that is, $x[i] = nil$ for $i \leq M$ and the records $x[M+1], \cdots x[M+N]$ are ordered in nondecreasing order by their key values.

Permuting

- The problem instance and the goal are same as that of sorting, except that here the key values of the records are required to form a permutation of $\{1, \cdots, N\}$.

Permuting

- The problem instance and the goal are same as that of sorting, except that here the key values of the records are required to form a permutation of $\{1, \cdots, N\}$.
- Before we get lower bounds on these two problems in the external memory model, let us consider a simpler problem first...

Permuting

- The problem instance and the goal are same as that of sorting, except that here the key values of the records are required to form a permutation of $\{1, \cdots, N\}$.
- Before we get lower bounds on these two problems in the external memory model, let us consider a simpler problem first...
- **Searching:** N sorted records stored in external memory at the beginning. Given a record r, determine if it is present in the list, and if it is, retrieve the key value associated to it.

Permuting

- The problem instance and the goal are same as that of sorting, except that here the key values of the records are required to form a permutation of $\{1, \cdots, N\}$.
- Before we get lower bounds on these two problems in the external memory model, let us consider a simpler problem first...
- **Searching:** N sorted records stored in external memory at the beginning. Given a record r, determine if it is present in the list, and if it is, retrieve the key value associated to it.
- **Lower bound for searching:** $\Omega(\log_B N)$. Proof..

Permuting

- The problem instance and the goal are same as that of sorting, except that here the key values of the records are required to form a permutation of $\{1, \cdots, N\}$.
- Before we get lower bounds on these two problems in the external memory model, let us consider a simpler problem first...
- **Searching:** N sorted records stored in external memory at the beginning. Given a record r, determine if it is present in the list, and if it is, retrieve the key value associated to it.
- **Lower bound for searching:** $\Omega(\log_B N)$. Proof..B-trees?
- Now let us derive lower bounds on permuting and sorting...

Algorithms in external memory

- Algorithms that achieve the $\Omega(\frac{N}{B} \log_{M/B} \frac{N}{B})$ lower bound:
  - M/B-way external mergesort.
  - Distribution sort (quicksort in external memory).
  - Buffer tree sort.
- We shall cover these next time.