Models of Computation 2: Cell probe model

Mayank Goswami

7.07.2014

The membership problem

Given n keys from the universe $M = \{1, \cdots, m\}$, store them into a table $\mathcal{T}$ of size n, such that for any given key K, one can efficiently determine whether K is in $\mathcal{T}$ or not.

The membership problem

Given n keys from the universe $M = \{1, \cdots, m\}$, store them into a table $\mathcal{T}$ of size n, such that for any given key K, one can efficiently determine whether K is in $\mathcal{T}$ or not. Let the search time be denoted as $f(n, m)$.

The membership problem

Given n keys from the universe $M = \{1, \cdots, m\}$, store them into a table $\mathcal{T}$ of size n, such that for any given key K, one can efficiently determine whether K is in $\mathcal{T}$ or not. Let the search time be denoted as $f(n, m)$.

- Simple case—$n = 2, m = 3$. $f(2, 3) =$?

The membership problem

Given n keys from the universe $M = \{1, \cdots, m\}$, store them into a table $\mathcal{T}$ of size n, such that for any given key K, one can efficiently determine whether K is in $\mathcal{T}$ or not. Let the search time be denoted as $f(n, m)$.

- Simple case—$n = 2, m = 3$. $f(2, 3) =?$
- $f(2, 4) =?$

The membership problem

Given n keys from the universe $M = \{1, \cdots, m\}$, store them into a table $\mathcal{T}$ of size n, such that for any given key K, one can efficiently determine whether K is in $\mathcal{T}$ or not. Let the search time be denoted as $f(n, m)$.

- Simple case—$n = 2, m = 3$. $f(2, 3) =?$
- $f(2, 4) =?$

Theorem

For every n there exists an $N(n)$ such that $f(n, m) = \lceil \log(n + 1) \rceil$ for all $m \geq N(n)$.

The membership problem

Given n keys from the universe $M = \{1, \cdots, m\}$, store them into a table $\mathcal{T}$ of size n, such that for any given key K, one can efficiently determine whether K is in $\mathcal{T}$ or not. Let the search time be denoted as $f(n, m)$.

- Simple case—$n = 2, m = 3$. $f(2, 3) =?$
- $f(2, 4) =?$

Theorem

For every n there exists an $N(n)$ such that $f(n, m) = \lceil \log(n + 1) \rceil$ for all $m \geq N(n)$.

Proof...

Generalizations

- Theorem holds under general conditions of pointers and repeated keys.

Generalizations

- Theorem holds under general conditions of pointers and repeated keys.
- Assume a set P of p special symbols (pointers), and an array $\mathcal{T}$ containing q cells.

Generalizations

- Theorem holds under general conditions of pointers and repeated keys.
- Assume a set P of p special symbols (pointers), and an array $\mathcal{T}$ containing q cells.
- Denote the search complexity as $f(n, m, p, q)$.

Generalizations

- Theorem holds under general conditions of pointers and repeated keys.
- Assume a set P of p special symbols (pointers), and an array $\mathcal{T}$ containing q cells.
- Denote the search complexity as $f(n, m, p, q)$.

Theorem

For any n, p, q there exists an $N(n, p, q)$ such that $f(n, m, p, q) = \lceil \log(n+1) \rceil$ for all $m \geq N(n, p, q)$.

What if one more cell that can store anything is allowed?

Theorem

There exists a number $N'(n)$ such that if $m \geq N'(n)$, then by adding one extra cell in a sorted table, the search can always be accomplished in two probes. The content in the extra cell is allowed to be any integer between 1 and m.

Proof on board.

Cell probe lower bound problems

Problems are broadly classified into two types:

- Static:

Cell probe lower bound problems

Problems are broadly classified into two types:

- Static:Three finite sets—D (data), Q (query) and A (answers) are given, along with a function $f : D \times Q \to A$.

Cell probe lower bound problems

Problems are broadly classified into two types:

- Static:Three finite sets—D (data), Q (query) and A (answers) are given, along with a function $f : D \times Q \to A$.
- Devise a scheme to encode elements of D into data structures in the memory of a RAM.

Cell probe lower bound problems

Problems are broadly classified into two types:

- Static:Three finite sets—D (data), Q (query) and A (answers) are given, along with a function $f : D \times Q \to A$.
- Devise a scheme to encode elements of D into data structures in the memory of a RAM.
- When $x \in D$ has been encoded, given $y \in Q$, efficiently (least probes) determine $f(x, y)$.

Cell probe lower bound problems

Problems are broadly classified into two types:

- Static:Three finite sets—D (data), Q (query) and A (answers) are given, along with a function $f : D \times Q \to A$.
- Devise a scheme to encode elements of D into data structures in the memory of a RAM.
- When $x \in D$ has been encoded, given $y \in Q$, efficiently (least probes) determine $f(x, y)$.
- Question: tradeoff between storage space s and query time t.

Cell probe lower bound problems

Problems are broadly classified into two types:

- Static:Three finite sets—D (data), Q (query) and A (answers) are given, along with a function $f : D \times Q \to A$.
- Devise a scheme to encode elements of D into data structures in the memory of a RAM.
- When $x \in D$ has been encoded, given $y \in Q$, efficiently (least probes) determine $f(x, y)$.
- Question: tradeoff between storage space s and query time t.
- Dynamic: Updates/operations to be performed, that change the cell contents.

Cell probe lower bound problems

Problems are broadly classified into two types:

- Static:Three finite sets—D (data), Q (query) and A (answers) are given, along with a function $f : D \times Q \to A$.
- Devise a scheme to encode elements of D into data structures in the memory of a RAM.
- When $x \in D$ has been encoded, given $y \in Q$, efficiently (least probes) determine $f(x, y)$.
- Question: tradeoff between storage space s and query time t.
- Dynamic: Updates/operations to be performed, that change the cell contents.
- Operation—insert/delete/query, etc.
- Each operations is assigned its own *decision assignment tree.*

Decision Assignment Tree

- Internal node: labelled with the index of a cell and has exactly 2^w children.

Decision Assignment Tree

- Internal node: labelled with the index of a cell and has exactly 2^w children.
- Each of the 2^w outgoing edges e has two values, a read value r_e and a write value w_e assigned to it. Both r_e and w_e are in $\{0,1\}^w$.

Decision Assignment Tree

- Internal node: labelled with the index of a cell and has exactly 2^w children.
- Each of the 2^w outgoing edges e has two values, a read value r_e and a write value w_e assigned to it. Both r_e and w_e are in $\{0,1\}^w$.
- Every value in $\{0,1\}^w$ occurs exactly once as a read value r_e.

Decision Assignment Tree

- Internal node: labelled with the index of a cell and has exactly 2^w children.
- Each of the 2^w outgoing edges e has two values, a read value r_e and a write value w_e assigned to it. Both r_e and w_e are in $\{0,1\}^w$.
- Every value in $\{0,1\}^w$ occurs exactly once as a read value r_e.
- When performing an operation we proceed from the root of the corresponding tree to a leaf.

Decision Assignment Tree

- Internal node: labelled with the index of a cell and has exactly 2^w children.
- Each of the 2^w outgoing edges e has two values, a read value r_e and a write value w_e assigned to it. Both r_e and w_e are in $\{0,1\}^w$.
- Every value in $\{0,1\}^w$ occurs exactly once as a read value r_e.
- When performing an operation we proceed from the root of the corresponding tree to a leaf.
- When we encounter a node labelled i, we read the content $c \in \{0,1\}^w$ of the cell C_i and proceed with the unique edge e with $r_e = c$.

Decision Assignment Tree

- Internal node: labelled with the index of a cell and has exactly 2^w children.
- Each of the 2^w outgoing edges e has two values, a read value r_e and a write value w_e assigned to it. Both r_e and w_e are in $\{0,1\}^w$.
- Every value in $\{0,1\}^w$ occurs exactly once as a read value r_e.
- When performing an operation we proceed from the root of the corresponding tree to a leaf.
- When we encounter a node labelled i, we read the content $c \in \{0,1\}^w$ of the cell C_i and proceed with the unique edge e with $r_e = c$.
- At the same time change the content of C_i to w_e.

Decision Assignment Tree

- Internal node: labelled with the index of a cell and has exactly 2^w children.
- Each of the 2^w outgoing edges e has two values, a read value r_e and a write value w_e assigned to it. Both r_e and w_e are in $\{0,1\}^w$.
- Every value in $\{0,1\}^w$ occurs exactly once as a read value r_e.
- When performing an operation we proceed from the root of the corresponding tree to a leaf.
- When we encounter a node labelled i, we read the content $c \in \{0,1\}^w$ of the cell C_i and proceed with the unique edge e with $r_e = c$.
- At the same time change the content of C_i to w_e.
- **The time complexity of a solution is given by the depth of the deepest tree in the solution.**

Lower bound techniques

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

Lower bound techniques

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

- Communication complexity method

Lower bound techniques

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.

Lower bound techniques

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.

Lower bound techniques

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.

Communication complexity technique

- Static problem $f : D \times Q \to A$; want to show there is *not* a solution with word size w, number of cell s, and query time t.

Communication complexity technique

- Static problem $f : D \times Q \to A$; want to show there is *not* a solution with word size w, number of cell s, and query time t.
- Two party communication complexity—Alice and Bob. Alice is given $y \in Q$, while Bob has $x \in D$.

Communication complexity technique

- Static problem $f : D \times Q \to A$; want to show there is *not* a solution with word size w, number of cell s, and query time t.
- Two party communication complexity—Alice and Bob. Alice is given $y \in Q$, while Bob has $x \in D$.
- The object of the game is to let Alice determine the value of $f(x, y)$ through communication with Bob.

Communication complexity technique

- Static problem $f : D \times Q \to A$; want to show there is *not* a solution with word size w, number of cell s, and query time t.
- Two party communication complexity—Alice and Bob. Alice is given $y \in Q$, while Bob has $x \in D$.
- The object of the game is to let Alice determine the value of $f(x, y)$ through communication with Bob.
- Alice sends a message from $\{1, \cdots, s\}$, Bob from $\{0, \cdots, 2^w - 1\}$. Strictly alternating messages.

Communication complexity technique

- Static problem $f : D \times Q \to A$; want to show there is *not* a solution with word size w, number of cell s, and query time t.
- Two party communication complexity—Alice and Bob. Alice is given $y \in Q$, while Bob has $x \in D$.
- The object of the game is to let Alice determine the value of $f(x, y)$ through communication with Bob.
- Alice sends a message from $\{1, \cdots, s\}$, Bob from $\{0, \cdots, 2^w - 1\}$. Strictly alternating messages.
- Alice sends the first message.

Communication complexity technique

- Static problem $f : D \times Q \to A$; want to show there is *not* a solution with word size w, number of cell s, and query time t.
- Two party communication complexity—Alice and Bob. Alice is given $y \in Q$, while Bob has $x \in D$.
- The object of the game is to let Alice determine the value of $f(x, y)$ through communication with Bob.
- Alice sends a message from $\{1, \cdots, s\}$, Bob from $\{0, \cdots, 2^w - 1\}$. Strictly alternating messages.
- Alice sends the first message.
- The complexity is the worst case number of rounds required in an optimal protocol before Alice gives the correct answer.

Communication complexity

Lemma

If there is a cell probe solution with space bound s and time complexity t for the static data structure problem, then the complexity of the communication game is at most t.

If we can show that the communication complexity is more than t, we have the desired lower bound.

Lemma

If there is a cell probe solution with space bound s and time complexity t for the static data structure problem, then the complexity of the communication game is at most t.

If we can show that the communication complexity is more than t, we have the desired lower bound.

Dynamic Polynomial Evaluation: Given a finite field F, let $f(a_0, \cdots, a_n, x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n$.

Lemma

If there is a cell probe solution with space bound s and time complexity t for the static data structure problem, then the complexity of the communication game is at most t.

If we can show that the communication complexity is more than t, we have the desired lower bound.

Dynamic Polynomial Evaluation: Given a finite field F, let $f(a_0, \cdots, a_n, x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n$.
Maintain $(y_0, \cdots, y_{n+1}) \in F^{n+2}$ under $CHANGE(i, b)$ setting $y_i = b$ and $EVALUATE$ returning $f(y_0, \cdots, y_{n+1})$.

Lemma

If there is a cell probe solution with space bound s and time complexity t for the static data structure problem, then the complexity of the communication game is at most t.

If we can show that the communication complexity is more than t, we have the desired lower bound.

Dynamic Polynomial Evaluation: Given a finite field F, let $f(a_0, \cdots, a_n, x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n$.
Maintain $(y_0, \cdots, y_{n+1}) \in F^{n+2}$ under $CHANGE(i, b)$ setting $y_i = b$ and $EVALUATE$ returning $f(y_0, \cdots, y_{n+1})$.
We study the problem in cell probe model with $w = O(\log n + \log |F|)$. Clearly operation time $t = O(n)$ is possible. We show $t = \Omega(n)$.

Lemma

If there is a cell probe solution with space bound s and time complexity t for the static data structure problem, then the complexity of the communication game is at most t.

If we can show that the communication complexity is more than t, we have the desired lower bound.

Dynamic Polynomial Evaluation: Given a finite field F, let $f(a_0, \cdots, a_n, x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n$.
Maintain $(y_0, \cdots, y_{n+1}) \in F^{n+2}$ under $CHANGE(i, b)$ setting $y_i = b$ and $EVALUATE$ returning $f(y_0, \cdots, y_{n+1})$.
We study the problem in cell probe model with $w = O(\log n + \log |F|)$. Clearly operation time $t = O(n)$ is possible. We show $t = \Omega(n)$.

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.

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.

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.

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 $i-1$.

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.