(Graph) Streaming Algorithms

December 10, 2022

Streaming Model

- The input consists of m objects/items/tokens $e_1, e_2, \ldots, e_m$ that are seen one by one by the algorithm.

Streaming Model

- The input consists of m objects/items/tokens $e_1, e_2, \ldots, e_m$ that are seen one by one by the algorithm.
- The algorithm has "limited" memory say for B tokens where $B < m$ (often $B << m$) and hence cannot store all the input

Streaming Model

- The input consists of m objects/items/tokens $e_1, e_2, \ldots, e_m$ that are seen one by one by the algorithm.
- The algorithm has "limited" memory say for B tokens where $B < m$ (often $B << m$) and hence cannot store all the input
- Want to compute interesting functions over input

Streaming Model

- The input consists of m objects/items/tokens $e_1, e_2, \ldots, e_m$ that are seen one by one by the algorithm.
- The algorithm has "limited" memory say for B tokens where $B < m$ (often $B << m$) and hence cannot store all the input
- Want to compute interesting functions over input

Some examples:

- Each token is a number from $[n]$
- High-speed network switch: tokens are packets with source, destination IP addresses and message contents.
- Each token is an edge in graph (graph streams)
- Each token is a point in some feature space
- Each token is a row/column of a matrix

Streaming model: motivation/connections

- Very large but slow storage (tape, slow disk) that is suited for sequential access and fast main memory. Read data in one (or more) passes from slow medium.

Streaming model: motivation/connections

- Very large but slow storage (tape, slow disk) that is suited for sequential access and fast main memory. Read data in one (or more) passes from slow medium.

- Network switches, sensors etc where huge amount of data is flying by and cannot be stored (due to cost or privacy/legal reasons) but one wants only high-level statistics.

Streaming model: motivation/connections

- Very large but slow storage (tape, slow disk) that is suited for sequential access and fast main memory. Read data in one (or more) passes from slow medium.

- Network switches, sensors etc where huge amount of data is flying by and cannot be stored (due to cost or privacy/legal reasons) but one wants only high-level statistics.

- Distributed computing where data stored in multiple machines, cannot send all data to central location. Streaming algorithms can simulate a class of algorithms that exchange small amount of data. Leads to sketching.

Problem 1: Detecting Majority element

Input: Given an array A of m integers.
Question: Does there exist an element in A that occurs more than $m/2$ times ?

Problem 1: Detecting Majority element

Input: Given an array A of m integers.
Question: Does there exist an element in A that occurs more than $m/2$ times ?

Problem 2: Frequent Element

Input: Given an array A of m integers, and an integer $k \geq 1$
Question: Does there exist an element that occurs at least m/k times ?

Misra Gries Algorithm

Algorithm 1 The Misra–Gries frequency estimation algorithm

Initialize:

1: $A \leftarrow$ (empty associative array)

Process (token j)**:**

2: **if** $j \in \text{keys}(A)$ **then**
3: $\quad A[j] \leftarrow A[j] + 1$
4: **else if** $|\text{keys}(A)| < k - 1$ **then**
5: $\quad A[j] \leftarrow 1$
6: **else**
7: $\quad$ **foreach** $\ell \in \text{keys}(A)$ **do**
8: $\quad\quad A[\ell] \leftarrow A[\ell] - 1$
9: $\quad\quad$ **if** $A[\ell] = 0$ **then** remove ℓ from A

Output (query a)**:**

10: **if** $a \in \text{keys}(A)$ **then** report $\hat{f}_a = A[a]$ **else** report $\hat{f}_a = 0$

Misra Gries Algorithm

Algorithm 1 The Misra–Gries frequency estimation algorithm

Initialize:

1: $A \leftarrow$ (empty associative array)

Process (token j)**:**

2: **if** $j \in \text{keys}(A)$ **then**
3: $\quad A[j] \leftarrow A[j] + 1$
4: **else if** $|\text{keys}(A)| < k - 1$ **then**
5: $\quad A[j] \leftarrow 1$
6: **else**
7: $\quad$ **foreach** $\ell \in \text{keys}(A)$ **do**
8: $\quad\quad A[\ell] \leftarrow A[\ell] - 1$
9: $\quad\quad$ **if** $A[\ell] = 0$ **then** remove ℓ from A

Output (query a)**:**

10: **if** $a \in \text{keys}(A)$ **then** report $\hat{f}_a = A[a]$ **else** report $\hat{f}_a = 0$

A second pass allow us to check if any of the non-empty counters are in fact the answer. Each key in A needs $O(\log n)$ bits and each counter takes $O(\log m)$ bits. In total $O(k(\log m + \log n))$ bits

Graph Streaming

- $G = (V, E)$ is an undirected graph
- $n = |V|$ and $m = |E|$
- Edges $e_1, e_2, \ldots, e_m$ seen as a stream, n known

Graph Streaming

- $G = (V, E)$ is an undirected graph
- $n = |V|$ and $m = |E|$
- Edges $e_1, e_2, \ldots, e_m$ seen as a stream, n known
- Massive graphs include social networks, web graph, call graphs, etc

Graph Streaming

- $G = (V, E)$ is an undirected graph
- $n = |V|$ and $m = |E|$
- Edges $e_1, e_2, \ldots, e_m$ seen as a stream, n known
- Massive graphs include social networks, web graph, call graphs, etc

Question:

- What graph problems can be solve with small space?

Semi-streaming Model: Goals

- Lower bounds show that we require $\Omega(n)$ memory

Semi-streaming Model: Goals

- Lower bounds show that we require $\Omega(n)$ memory
- What can we compute about G in $o(m)$ space?
- Assume we have $\Theta(n \cdot \text{polylog}(n))$ memory. About polylog per vertex of the graph
- Can solve several interesting problems. Essentially reduce dense graphs to sparse graphs.

Main idea

Maintain a sparse subgraph that preserves the property under consideration

Problem 2: Connectivity in graph

I/p: Given an undirected graph $G = (V, E)$
Q: Is the graph connected ?

Algorithm 1. A single-pass semi-streaming algorithm for connectivity.

(*i*) Let $F \leftarrow \emptyset$;

(*ii*) For any edge e in the stream, add e to F if $F \cup \{e\}$ does not have a cycle.

(*iii*) Return G is connected iff F is connected.

Problem 2: Connectivity in graph

I/p: Given an undirected graph $G = (V, E)$
Q: Is the graph connected ?

Algorithm 1. A single-pass semi-streaming algorithm for connectivity.

(*i*) Let $F \leftarrow \emptyset$;

(*ii*) For any edge e in the stream, add e to F if $F \cup \{e\}$ does not have a cycle.

(*iii*) Return G is connected iff F is connected.

- Uses $O(n \log n)$ space as F at every point is a forest and hence contains at most n-1 edges.
- Any single-pass algorithm requires $\omega(n \log n)$ bits of space by a result of Sun and Woodruff ....hence above algorithm is asymptotically optimal.

Problem 3: k-Edge Connectivity

I/p: Given a graph $G = (V, E)$ and an integer $k \geq 1$

Q: is G k-edge-connected, i.e., at least k edges needs to be removed from G to make it disconnected ?

Problem 3: k-Edge Connectivity

I/p: Given a graph $G = (V, E)$ and an integer $k \geq 1$

Q: is G k-edge-connected, i.e., at least k edges needs to be removed from G to make it disconnected ?

Algorithm 2. A single-pass streaming algorithm for k-edge-connectivity.

(*i*) Let $F_1, F_2, \ldots, F_k \leftarrow \emptyset$;

(*ii*) For any edge e in the stream, if there is any F_i such that $F_i \cup \{e\}$ has no cycle, add e to this F_i.

(*iii*) Return G is k-connected iff $F := F_1 \cup \ldots \cup F_k$ is k-connected.

Claim 1: The subgraph $F = \uplus_{i \in [k]} F_i$ is k-connected iff G is k-connected.

Testing Bipartiteness

What is the sparse certificate for being bipartite?

Testing Bipartiteness

What is the sparse certificate for being bipartite?

Proposition

G is bipartite iff there is no odd cycle in G.

Testing Bipartiteness

What is the sparse certificate for being bipartite?

Proposition

G is bipartite iff there is no odd cycle in G.

Initialize : $F \leftarrow \phi$, $X \leftarrow 1$;

Process $\{u, v\}$:

```
1 if X then
2     if F ∪ {{u, v}} does not contain a cycle then
3         F ← F ∪ {{u, v}} ;
4     else if F ∪ {{u, v}} contains an odd cycle then
5         X ← 0 ;
```

Output : X ;

Testing Bipartiteness

What is the sparse certificate for being bipartite?

Proposition

G is bipartite iff there is no odd cycle in G.

Initialize : $F \leftarrow \phi$, $X \leftarrow 1$;

Process $\{u, v\}$:

```
1 if X then
2     if F ∪ {{u, v}} does not contain a cycle then
3         F ← F ∪ {{u, v}} ;
4     else if F ∪ {{u, v}} contains an odd cycle then
5         X ← 0 ;
```

Output : X ;

Claim 1: The algorithm outputs 1 iff G is bipartite.

Testing Bipartiteness

What is the sparse certificate for being bipartite?

Proposition

G is bipartite iff there is no odd cycle in G.

Initialize : $F \leftarrow \phi$, $X \leftarrow 1$;

Process $\{u, v\}$:

```
1 if X then
2     if F ∪ {{u, v}} does not contain a cycle then
3         F ← F ∪ {{u, v}} ;
4     else if F ∪ {{u, v}} contains an odd cycle then
5         X ← 0 ;
```

Output : X ;

Claim 1: The algorithm outputs 1 iff G is bipartite.

Claim 2: Can be implemented in $O(n \log n)$ space.

Problem 4: Find the number of components

Goal: Compute the number of connected components.

Problem 4: Find the number of components

Goal: Compute the number of connected components.

- Maintain a spanning forest F: need only $O(n)$ edges
- When edge $e_i = (u, v)$ arrives. If u and v are in different components add e_i to spanning forest. Otherwise discard e_i.

Problem 4: Find the number of components

Goal: Compute the number of connected components.

- Maintain a spanning forest F: need only $O(n)$ edges
- When edge $e_i = (u, v)$ arrives. If u and v are in different components add e_i to spanning forest. Otherwise discard e_i.
- No. of connected components in F and G are same.

Thank You.