Optimal Incremental Sorting

Rodrigo Paredes - Gonzalo Navarro

{raparede,gnavarro}@dcc.uchile.cl

Center for Web Research, Department of Computer Science,
Universidad de Chile
Contents

- Introduction
- Basic concepts
- IncrementalQuickSelect
- Sketch of IQS complexity
- IQS and Kruskal’s MST
- Experimental results
- Conclusions
Introduction

- Assume we have a fixed unsorted set $A$
- Suppose that someone asks for the first object
- and now, for the second, for the third, ..., for the who-knows-th element
- In this scenario we don’t know when to stop the incremental searching!!
Algorithmic examples

- Reviewing the edge set one by one while computing the MST with Kruskal’s algorithm
- Ranking and returning the most relevant documents from the query outcome chunk by chunk in a Web search engine
Basic concepts

The naive solution

- Sort the set $A$
- Return as many elements as they are required
- But, have we to sort the whole set $A$ to retrieve few elements?
Problem formulation

- Incremental sorting
  "Given set $A$ of $m$ numbers, output the elements of $A$ from smallest to largest, so that the process can be stopped after $k$ elements have been output, for any $k \leq m$ that is unknown to the algorithm"

- This is a sort of online problem as $k$ isn’t known beforehand
Related work 1

- The offline version is known as *Partial Sorting*:
  
  “Given set $A$ of $m$ numbers and an integer $k \leq m$, output the smallest $k$ elements of $A$ in ascending order”

- Asymptotical optimal solution:
  
  - Select $p$, which is the $k$-th element of $A$, with *QUICKSELECT* in time $O(m)$ in average
  - Sorting the elements in the left partition, those ones smaller than $p$, with *QUICKSORT* in time $O(k \log k)$ in average
Related work 2

- Total time $O(m + k \log k) \leq O(m \log m)$
- Partial Quick Sort (PQS) interleaves the \texttt{QUICKSELECT} and \texttt{QUICKSORT} stages saving a bit of work
- Current online solution: Minheapify the array, and minima extractions from the heap (HEx)
Contents

- Introduction
- Basic concepts
- Incremental QuickSelect
- Sketch of IQS complexity
- IQS and Kruskal’s MST
- Experimental results
- Conclusions
**IncrementalQuickSelect**

- **IncrementalQuickSelect** (IQS) is an $O(m + k \log k)$ time on average algorithm to solve the Incremental sorting problem.

- **Applications:**
  - Kruskal’s MST Algorithm
  - Ranking a set
  - Partial sorting from any place of the array
  - Priority Queues (!?)
A wrong way

- Call QUICKSELECT on $A[0, m - 1]$
- Call QUICKSELECT on $A[1, m - 1]$
- ...
- Call QUICKSELECT on $A[k - 1, m - 1]$
- The overall complexity is $O(km)$ ($> O(m + k \log k)$), and $A[0, k - 1]$ is sorted,
- But what’s wrong?
The Core Idea

When we call QuickSelect on $A[1, m - 1]$, we have already computed a decreasing sequence of pivots from the invocation on $A[0, m - 1]$.

Reuse the computed pivots!!
Using the stored pivots

Given the pivots in $S$, the next minimum place is:

- The index on top of $S$

<table>
<thead>
<tr>
<th>place</th>
<th>value</th>
<th>obs</th>
</tr>
</thead>
<tbody>
<tr>
<td>2nd</td>
<td>18</td>
<td>chunk</td>
</tr>
<tr>
<td>3rd</td>
<td>25</td>
<td>pivot</td>
</tr>
<tr>
<td>4th</td>
<td>29</td>
<td>chunk</td>
</tr>
<tr>
<td>5th</td>
<td>33</td>
<td>pivot</td>
</tr>
</tbody>
</table>
The algorithm

**IQS** (Set $A$, Int $idx$, Stack $S$)

If $idx = S$.top() Then $S$.pop(), Return $A[idx]$

$pidx \leftarrow \text{random}[idx, S$.top()−1]$

$pidx' \leftarrow \text{partition}(A, A[pidx], idx, S$.top()−1)\)

// $A[0] \leq \ldots \leq A[idx - 1]$

// $\leq A[idx, pidx' - 1] \leq A[pidx']$

// $\leq A[pidx' + 1, S$.top()−1]$

// $\leq A[S$.top(), $m - 1]$

$S$.push($pidx'$)

Return IQS($A$, $idx$, $S$)
Contents

- Introduction
- Basic concepts
- INCREMENTAL QUICKSELECT
- Sketch of IQS complexity
- IQS and Kruskal’s MST
- Experimental results
- Conclusions
Contents

- Introduction
- Basic concepts
- INCREMENTAL QUICK SELECT
- Sketch of IQS complexity
- IQS and Kruskal’s MST
- Experimental results
- Conclusions
Complexity sketch

$h = j = 5$
$h = 4$
$h = 3$
$h = 2$
$h = 1$

QuickSort = $O(k \log k)$

QuickSelect = $O(m)$
The recurrence

\[ T(m, k) = m - 1 + \frac{1}{m} \left( \sum_{p=k}^{m-1} T(p, k) \right) + T(k - 1, k - 1) + \sum_{p=0}^{k-2} \left( T(p, p) + T(m - 1 - p, k - p - 1) \right) \]

\[ T(m, k) < 4m - 8k + (3k + 1)H_k < 4m + 3kH_k \]
Contents

- Introduction
- Basic concepts
- INCREMENTAL QUICKSELECT
- Sketch of IQS complexity
- IQS and Kruskal’s MST
- Experimental results
- Conclusions
Contents

- Introduction
- Basic concepts
- **INCREMENTAL QUICK SELECT**
- Sketch of IQS complexity
- **IQS and Kruskal’s MST**
- Experimental results
- Conclusions
Kruskal’s MST

Kruskal1 (Graph G(V, E))
UnionFind C ← {v ∈ V, {v}}
// the set of all connected components
mst ← ∅ // the growing minimum spanning tree
ascendingSort(E), k ← 0
While |C| > 1 Do
  // select an edge in ascending order
  (e = {u, v}) ← E[k], k ← k + 1
  If C.find(u) ≠ C.find(v) Then
    mst ← mst ∪ {e}, C.union(u, v)
Return mst
**IQS and Kruskal’s MST**

**Kruskal3** (Graph $G(V,E)$)

UnionFind $C \leftarrow \{v \in V, \{v\}\}$
// the set of all connected components

$mst \leftarrow \emptyset$ // the growing minimum spanning tree

Stack $S$, $S$.push($m$), $k \leftarrow 0$ // $m = |E|$

**While** $|C| > 1$ **Do**

// select the lowest edge incrementally

$(e = \{u,v\}) \leftarrow$ IQS($E$, $k$, $S$), $k \leftarrow k + 1$

**If** $C$.find($u$) $\neq$ $C$.find($v$) **Then**

$mst \leftarrow mst \cup \{e\}$, $C$.union($u$, $v$)

**Return** $mst$
Contents

- Introduction
- Basic concepts
- Incremental QuickSelect
- Sketch of IQS complexity
- IQS and Kruskal’s MST
- Experimental results
- Conclusions
Experimental results

- Evaluating IQS
- Evaluating Kruskal3
## Evaluating IQS 1

<table>
<thead>
<tr>
<th></th>
<th>Fitting</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>PQS&lt;sub&gt;cpu&lt;/sub&gt;</td>
<td>19.70m + 14.21k log&lt;sub&gt;2&lt;/sub&gt; k</td>
<td>3.90%</td>
</tr>
<tr>
<td>PQS&lt;sub&gt;cmp&lt;/sub&gt;</td>
<td>2.047m + 1.301k log&lt;sub&gt;2&lt;/sub&gt; k</td>
<td>3.55%</td>
</tr>
<tr>
<td>IQS&lt;sub&gt;cpu&lt;/sub&gt;</td>
<td>19.88m + 14.21k log&lt;sub&gt;2&lt;/sub&gt; k</td>
<td>3.89%</td>
</tr>
<tr>
<td>IQS&lt;sub&gt;cmp&lt;/sub&gt;</td>
<td>2.047m + 1.301k log&lt;sub&gt;2&lt;/sub&gt; k</td>
<td>3.55%</td>
</tr>
<tr>
<td>QSS&lt;sub&gt;cpu&lt;/sub&gt;</td>
<td>20.00m + 14.52k log&lt;sub&gt;2&lt;/sub&gt; k</td>
<td>3.89%</td>
</tr>
<tr>
<td>QSS&lt;sub&gt;cmp&lt;/sub&gt;</td>
<td>2.050m + 1.362k log&lt;sub&gt;2&lt;/sub&gt; k</td>
<td>3.61%</td>
</tr>
<tr>
<td>HEx&lt;sub&gt;cpu&lt;/sub&gt;</td>
<td>25.96m + 85.88k log&lt;sub&gt;2&lt;/sub&gt; m</td>
<td>5.05%</td>
</tr>
<tr>
<td>HEx&lt;sub&gt;cmp&lt;/sub&gt;</td>
<td>1.892m + 1.875k log&lt;sub&gt;2&lt;/sub&gt; m</td>
<td>0.65%</td>
</tr>
</tbody>
</table>
- Almost the same time that PQS or QSS
- HEx (the online approach) is 6 times slower than IQS
Evaluating IQS 3

- $k \leq 0.01m$: dominates $m$
- $0.01m < k \leq 0.04m$: both terms take part in the cost
- $0.04m < k$: dominates $k \log k$
### Evaluating Kruskal3 1

<table>
<thead>
<tr>
<th></th>
<th>Fitting</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sorted edges</td>
<td>$0.532n \ln n$</td>
<td>1.47%</td>
</tr>
<tr>
<td>Kruskal1&lt;sub&gt;cpu&lt;/sub&gt;</td>
<td>$12.23m \log_2 m$</td>
<td>6.74%</td>
</tr>
<tr>
<td>Kruskal2&lt;sub&gt;cpu&lt;/sub&gt;</td>
<td>$51.62m + 34.84n \log_2 n \log_2 m$</td>
<td>9.62%</td>
</tr>
<tr>
<td>Kruskal3&lt;sub&gt;cpu&lt;/sub&gt;</td>
<td>$21.67m + 10.01n \log_2^2 n$</td>
<td>9.75%</td>
</tr>
</tbody>
</table>

Note that we expect to sort $\frac{1}{2}n \ln n$ edges

[Jason, Knuth, Łuczak and Pittel, *The birth of the giant component, 1993*]
We have the fastest Kruskal’s MST implementation, for all edge density, and for all $|V|$ :-)

Optimal Incremental Sorting - Rodrigo Paredes - Gonzalo Navarro - Contents - p.28/33
Contents

- Introduction
- Basic concepts
- IncrementalQuickSelect
- Sketch of IQS complexity
- IQS and Kruskal’s MST
- Experimental results
- Conclusions
Conclusions

- IQS is an algorithm to incrementally retrieve the next smallest element from a set.
- IQS has the same complexity than existing solutions, but
- IQS as fast as the optimal offline algorithm (PQS)
- IQS have practical applicability in many problems: MST, ranking, partial sorting form any place of the array, and priority queues
Next work: MinQuickHeaps

- Heapify: IQS(A, 0, S ← |A|): O(m)
- Extraction: IQS(A, k, S): O(\log k) amortized
- Insertion: We need to move some pivots
Can we improve the Prim’s MST algorithm?

- The *DecreaseKey* also can be performed efficiently
- Worst case $O(\log n)$, but in practice it could be $O(1)$ in average
- We can implement Prim’s with minQH, and using the *DecreaseKey* operation, obtain an $O(m + n \log n)$ MST algorithm :-)}
Optimal Incremental Sorting

Rodrigo Paredes - Gonzalo Navarro

{rapared,gnavarro}@dcc.uchile.cl

Center for Web Research, Department of Computer Science,
Universidad de Chile