A Functional Approach to External Graph Algorithms

Paper Presentation – 6.5060

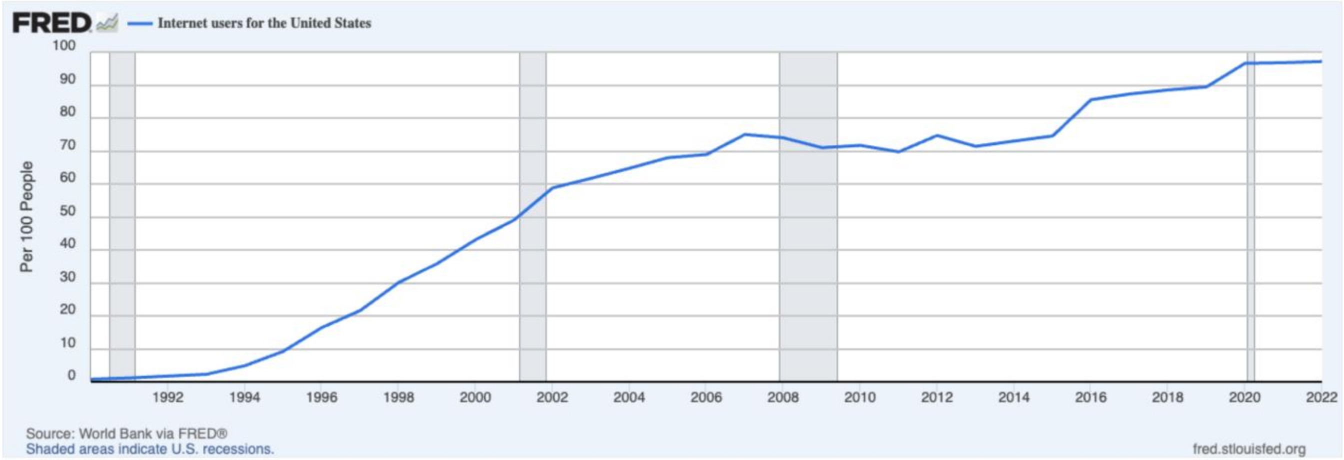
A Functional Approach to External Graph Algorithms

James Abello, Adam Buchsbaum, Jefferey Westbrook

- Researchers at AT&T

Published in the March 2002 edition of Algorithmica

Published as an article in 1998 and a conference paper in 1999



Motivation

- Data often exceeds main memory limits
- External memory is orders of magnitude slower
- Classical algorithms don’t scale well in I/O bound regimes
- Graph algorithms exhibit particularly poor data locality

Computation Model

I/O Model of Complexity (Aggarwal and Vitter)

- N = Number of items
- M = Number of items that can fit into main memory
- B = Number of items per disk block

General Goal:

Replace N times bounds with $\mathrm{N/B}$

Replace $\log_2(\mathrm{n})$ time bounds with $\log_{\mathrm{M/B}}(\mathrm{n})$ time bounds

Primitives

- $\text{sort}(N) = \Theta((N/B)\log_{M/B}(N/B))$
- $\text{scan}(N) = \lceil N/B \rceil$

Previous Approaches

PRAM Simulation

A PRAM algorithm using N processors and N Space to solve a problem of size N in time t can be simulated in external memory using one processor in $O(t \cdot sort(N))$ I/O operations

Represent memory and processor state in external memory

- Not practical
- Used to derive existence of external memory algorithms

Previous Approaches

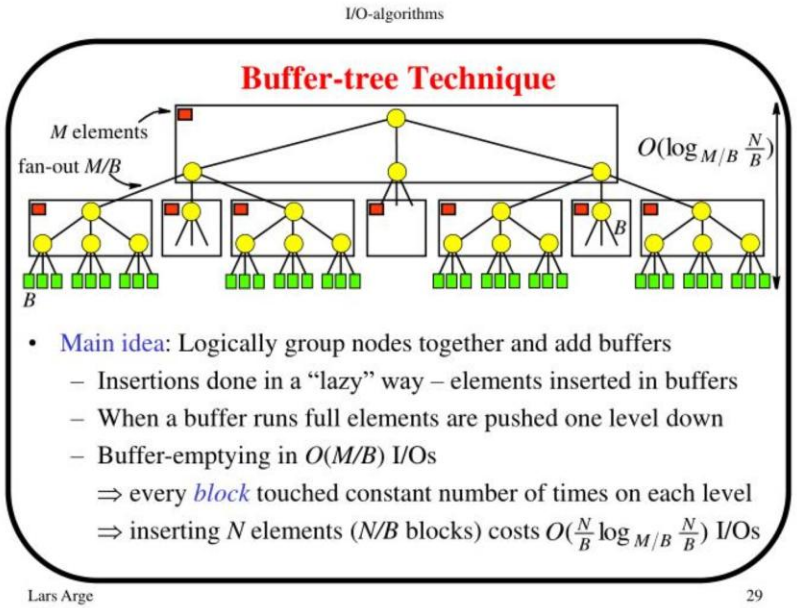
Buffering Data Structures

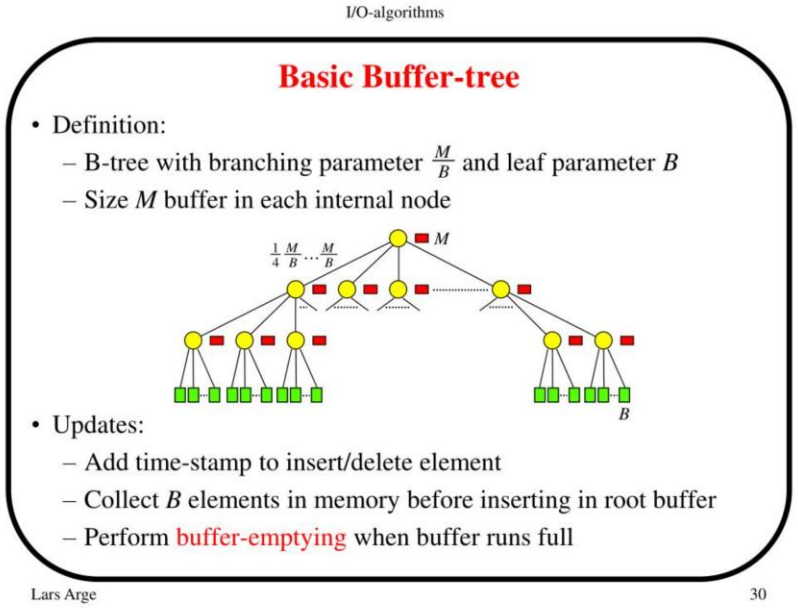
Create external variants of classic data structures to amortize I/O operations

Example: Buffer Trees

[I/O-Algorithms. Lars Arge Spring 2012 February 14 , 2012]

- Graph algorithms often require more complicated interfaces





Functional Approach

- Divide-and-conquer paradigm based on transformations that can be expressed functionally
  - Each step performs computation without modifying the input
  - “No side effects”

Advantages:

- Simplicity
- Checkpointing
- Competitive Performance

Basic Functions

Selection, Relabeling, Contraction, Deletion

Selection

Return the kth biggest element from a list of items

1. Partition I into cM-element subsets, for some $0 < c < 1$.
2. Determine the median of each subset in main memory. Let S be the set of medians of the subsets.
3. $m \leftarrow \texttt{Select}(S, \lceil S/2 \rceil)$.
4. Let I_1, I_2, I_3 be the sets of elements less than, equal to, and greater than m, respectively.
5. If $|I_1| \geq k$, then return $\texttt{Select}(I_1, k)$.
6. Else if $|I_1| + |I_2| \geq k$, then return m.
7. Else return $\texttt{Select}(I_3, k - |I_1| - |I_2|)$.

I/O Complexity: $O(\text{scan}(|I|))$

Relabeling

RL(F, I):

- A rooted forest F expressed as a set of directed edges (p(v), v)
- An edge set I

Output:

- A new edge set where each endpoint is replaced with its parent, if it exists (remove self-loops)

Relabeling

1. Sort F by source vertex, v.
2. Sort I by second component.
3. Process F and I in tandem.
   (a) Let $\{s, h\} \in I$ be the current edge to be relabeled.
   (b) Scan F starting from the current edge until finding $(p(v), v)$ such that $v \geq h$.
   (c) If $v = h$, then add $\{s, p(v)\}$ to I''; otherwise, add $\{s, h\}$ to I''.
4. Repeat steps 2 and 3, relabeling first components of edges in I'' to construct I'.

I/O Complexity: $O(\mathrm{sort}(|I|) + \mathrm{sort}(|F|))$

Contraction

Merge set of vertices into single super-vertex

Concrete Application: Contract subcomponents into single vertex

1. For each $C_i = \{\{u_1, v_1\}, \ldots\}$:
   (a) $R_i \leftarrow \emptyset$.
   (b) Pick u_1 to be the canonical vertex.
   (c) For each $\{x, y\} \in C_i$, add (u_1, x) and (u_1, y) to relabeling R_i.
2. Apply relabeling $\bigcup_i R_i$ to I, yielding the contracted edge list I'.

I/O Complexity: $O(\mathrm{sort}(|I|) + \mathrm{sort}(\sum_i |C_i|))$

Note: Preserves Connectivity

Applications

Connected Components, Minimum Spanning Forests, Bottleneck Minimum Spanning Forest, Maximal Matching

Connected Components

Input: G = (V,E)

Output: A forest of rooted stars corresponding to connected components of G

1. Let E_1 be any half of the edges of G; let $G_1 = (V, E_1)$.
2. Compute $CC(G_1)$ recursively.
3. Let $G' = G/CC(G_1)$.
4. Compute $CC(G')$ recursively.
5. $CC(G) = CC(G') \cup RL(CC(G'), CC(G_1))$.

I/O Complexity: $\mathrm{O}(\mathrm{sort}(\mathrm{E}) + (\mathrm{E}/\mathrm{V})\mathrm{sort}(\mathrm{V}) \log_2(\mathrm{V}/\mathrm{M}))$

General Formula

1. $G_1 \leftarrow S(G);$
2. $G_2 \leftarrow T_1(G,\ f_{\mathcal{P}}(G_1));$
3. $f_{\mathcal{P}}(G) = T_2(G, G_1, G_2,\ f_{\mathcal{P}}(G_1),\ f_{\mathcal{P}}(G_2)).$

G1 is a subgraph of G

G2 applies some transformation to combine the recursive solution (f_p) and G

The full solution applies a transformation to the recursive solutions and intermediaries

Full Results

Table 1. I/O bounds for our functional external algorithms.

Problem	Deterministic	Randomized	
	I/O bound	I/O Bound	With probability
Connected components	$O(sort(E) + \frac{E}{V} sort(V) \log_2 \frac{V}{M})$	$O(sort(E))$	$1 - e^{\Omega(E)}$
MSFs	$O(sort(E) + \frac{E}{V} sort(V) \log_2 \frac{V}{M})$	$O(sort(E))$	$1 - e^{\Omega(E)}$
BMSFs	$O(sort(E) + \frac{E}{V} sort(V) \log_2 \frac{V}{M})$	$O(sort(E))$	$1 - e^{\Omega(E)}$
Maximal matchings	$O(\frac{E}{V} sort(V) \log_2 \frac{V}{M})$	$O(sort(E))$	$1 - \varepsilon$ for any fixed ε
Maximal independent sets		$O(sort(E))$	$1 - \varepsilon$ for any fixed ε

Randomized Algorithms

- Externalize linear-time MSF algorithm (Karger et al.) to obtain randomized algorithms for CC, MSF, and BMSF
  - Luby's Algorithm => Maximal Independent Sets
  - Yang et al. => Maximal Matching

- Advantages:
  - Improved run times w.h.p
  - Simplicity

Semi-External Memory

Conditions:

- $V \leq M$
- $\mathrm{E} > \mathrm{M}$

Example: AT&T telephone call network

- V = 250 million
- E = 100 billion per year

Keeping vertex data structures in memory can improve performance

Semi-External Memory – CC & MSF

- Use disjoint set union to compute forest of rooted stars with a single scan
  - Kept in memory => No I/O
  - I/O Complexity: $O(\mathrm{scan}(E))$
- For MSF, sort edges first or use dynamic trees
  - Note: Dynamic trees increase internal runtime but further improves external I/O bound

Future Directions

- Incremental and dynamic algorithms for external graph problems
- Easier connectivity testing to improve BMSF algorithm
- Functional contraction of arbitrary edge sets
- Faster BMSF algorithm (than MSF) in external setting

Discussion Questions

Is this framework / model of computation applicable to SSD?

Where do we see the impacts of functional external graph algorithms in research today?

Do functional external graph algorithms have any practical applicability?