Karlsruhe Institute of Technology

Advanced Datastructures Project

Implementation Summary

Tobias Fuchs | 2022-07-25

Outline

Dynamic Bitvectors

- [Navarro, 2016]

Dynamic Bitvectors

- [Navarro, 2016]
- Balanced Search Tree: Red-Black tree
  - Store number of bits and ones in left subtree per inner node
  - Store parent pointer for faster Red-Black tree operations (+ color)

Dynamic Bitvectors

- [Navarro, 2016]
- Balanced Search Tree: Red-Black tree
  - Store number of bits and ones in left subtree per inner node
  - Store parent pointer for faster Red-Black tree operations (+ color)
- Leafs: Simple bit-vectors based on `std::vector<uint64_t>`
  - `std::popcount` for scanning leafs
  - Initial leaf size w blocks
  - Stealing or Merging when $\leq \frac{w}{2}$ blocks
  - Splitting when $\geq 2w$ blocks

Dynamic Bitvectors

- [Navarro, 2016]
- Balanced Search Tree: Red-Black tree
  - Store number of bits and ones in left subtree per inner node
  - Store parent pointer for faster Red-Black tree operations (+ color)
- Leafs: Simple bit-vectors based on `std::vector<uint64_t>`
  - `std::popcount` for scanning leafs
  - Initial leaf size w blocks
  - Stealing or Merging when $\leq \frac{w}{2}$ blocks
  - Splitting when $\geq 2w$ blocks
- Access, set, flip, rank, and select in $O(\log n)$

Dynamic Bitvectors

- [Navarro, 2016]
- Balanced Search Tree: Red-Black tree
  - Store number of bits and ones in left subtree per inner node
  - Store parent pointer for faster Red-Black tree operations (+ color)
- Leafs: Simple bit-vectors based on `std::vector<uint64_t>`
  - `std::popcount` for scanning leafs
  - Initial leaf size w blocks
  - Stealing or Merging when $\leq \frac{w}{2}$ blocks
  - Splitting when $\geq 2w$ blocks
- Access, set, flip, rank, and select in $O(\log n)$
- Insert and delete in $O(w + \log n)$

Dynamic BP Trees

- [Navarro, 2016]

Dynamic BP Trees

- [Navarro, 2016]
- Dynamic bitvector augmented with min-tree for excess

Dynamic BP Trees

- [Navarro, 2016]
- Dynamic bitvector augmented with min-tree for excess
- Excess chunks
  - Store block excess and min excess only once for every c blocks

Dynamic BP Trees

- [Navarro, 2016]
- Dynamic bitvector augmented with min-tree for excess
- Excess chunks
  - Store block excess and min excess only once for every c blocks
- Subtree size, parent, child, insert, and delete
  - Mainly as in lecture
  - Forward-/backward-search to find position

Dynamic BP Trees

- [Navarro, 2016]
- Dynamic bitvector augmented with min-tree for excess
- Excess chunks
  - Store block excess and min excess only once for every c blocks
- Subtree size, parent, child, insert, and delete
  - Mainly as in lecture
  - Forward-/backward-search to find position
- Better i-th child [Navarro and Sadakane, 2014] not implemented

Evaluation
Implementation & Setup

- Implemented in C++17[1]
- Heavy use of template classes for hyper-parameter optimization and code de-duplication
- Avoiding recursion

[1] `https://github.com/mathefuchs/advanced-datastructures-st22`

Evaluation
Implementation & Setup

- Implemented in C++17[1]
- Heavy use of template classes for hyper-parameter optimization and code de-duplication
- Avoiding recursion

- `gcc-11.3.0` with flags `-O3 -mtune=native -march=native`
- Single core; `ARM64` architecture

[1] `https://github.com/mathefuchs/advanced-datastructures-st22`

Evaluation

Implementation & Setup

- Implemented in C++17[1]
- Heavy use of template classes for hyper-parameter optimization and code de-duplication
- Avoiding recursion

- `gcc-11.3.0` with flags `-O3 -mtune=native -march=native`
- Single core; `ARM64` architecture

- GTest for testing
- GitHub Actions pipeline
- Code coverage, especially helpful for the Red-Black tree insertion / deletion
  - 1254 lines of code and 95 % line coverage

[1] `https://github.com/mathefuchs/advanced-datastructures-st22`

Evaluation

Dynamic Bitvectors

Leaf	Time (s)	Space (bits)	Space Overhead	Best Runtime	Best Space	Weighted
2						
4						
8						
16						
32						
64						
128						
256						
512						
1024						

Evaluation

Dynamic Bitvectors

Leaf	Time (s)	Space (bits)	Space Overhead	Best Runtime	Best Space	Weighted
2	31 962.4					
4	29 468.2					
8	24 966.8					
16	25 638.2					
32	24 865.0					
64	25 380.2					
128	28 582.8					
256	36 311.2					
512	55 094.4					
1024	93 804.8					

Evaluation

Dynamic Bitvectors

Leaf	Time (s)	Space (bits)	Space Overhead	Best Runtime	Best Space	Weighted
2	31 962.4	77 035 616	7.704			
4	29 468.2	42 482 336	4.248			
8	24 966.8	26 236 256	2.624			
16	25 638.2	18 120 160	1.812			
32	24 865.0	14 061 248	1.406			
64	25 380.2	12 030 496	1.203			
128	28 582.8	11 013 888	1.101			
256	36 311.2	10 507 200	1.051			
512	55 094.4	10 253 152	1.025			
1024	93 804.8	10 126 272	1.013			

Evaluation

Dynamic Bitvectors

Leaf	Time (s)	Space (bits)	Space Overhead	Best Runtime	Best Space	Weighted
2	31 962.4	77 035 616	7.704	1.285	7.608	
4	29 468.2	42 482 336	4.248	1.185	4.195	
8	24 966.8	26 236 256	2.624	1.004	2.591	
16	25 638.2	18 120 160	1.812	1.031	1.789	
32	24 865.0	14 061 248	1.406	1.000	1.389	
64	25 380.2	12 030 496	1.203	1.021	1.188	
128	28 582.8	11 013 888	1.101	1.150	1.088	
256	36 311.2	10 507 200	1.051	1.460	1.038	
512	55 094.4	10 253 152	1.025	2.216	1.013	
1024	93 804.8	10 126 272	1.013	3.773	1.000	

Evaluation

Dynamic Bitvectors

Leaf	Time (s)	Space (bits)	Space Overhead	Best Runtime	Best Space	Weighted
2	31 962.4	77 035 616	7.704	1.285	7.608	4.763
4	29 468.2	42 482 336	4.248	1.185	4.195	2.841
8	24 966.8	26 236 256	2.624	1.004	2.591	1.877
16	25 638.2	18 120 160	1.812	1.031	1.789	1.448
32	24 865.0	14 061 248	1.406	1.000	1.389	1.214
64	25 380.2	12 030 496	1.203	1.021	1.188	1.113
128	28 582.8	11 013 888	1.101	1.150	1.088	1.115
256	36 311.2	10 507 200	1.051	1.460	1.038	1.228
512	55 094.4	10 253 152	1.025	2.216	1.013	1.554
1024	93 804.8	10 126 272	1.013	3.773	1.000	2.248

Evaluation

Dynamic Bitvectors

Leaf	Time (s)	Space (bits)	Space Overhead	Best Runtime	Best Space	Weighted
2	31 962.4	77 035 616	7.704	1.285	7.608	4.763
4	29 468.2	42 482 336	4.248	1.185	4.195	2.841
8	24 966.8	26 236 256	2.624	1.004	2.591	1.877
16	25 638.2	18 120 160	1.812	1.031	1.789	1.448
32	24 865.0	14 061 248	1.406	1.000	1.389	1.214
64	25 380.2	12 030 496	1.203	1.021	1.188	1.113
128	28 582.8	11 013 888	1.101	1.150	1.088	1.115
256	36 311.2	10 507 200	1.051	1.460	1.038	1.228
512	55 094.4	10 253 152	1.025	2.216	1.013	1.554
1024	93 804.8	10 126 272	1.013	3.773	1.000	2.248

Evaluation

Dynamic BP Trees

Leaf	Excess	Time (s)	Space (bits)	Space Overhead	Best Runtime	Best Space	Weighted
2	1						
4	1						
4	2						
8	1						
8	2						
8	4						
16	1						
16	2						
...	...	...	...	...	...	...	...
128	16						

Evaluation

Dynamic BP Trees

Leaf	Excess	Time (s)	Space (bits)	Space Overhead	Best Runtime	Best Space	Weighted
2	1	3609.0					
4	1	3653.8					
4	2	4658.2					
8	1	4058.6					
8	2	5731.2					
8	4	8886.8					
16	1	5159.0					
16	2	8098.0					
...	...	...	...	...	...	...	...
128	16	327 873.6					

Evaluation

Dynamic BP Trees

Leaf	Excess	Time (s)	Space (bits)	Space Overhead	Best Runtime	Best Space	Weighted
2	1	3609.0	46 403 904	46.404			
4	1	3653.8	24 423 744	24.424			
4	2	4658.2	22 252 864	22.253			
8	1	4058.6	15 809 696	15.810			
8	2	5731.2	13 736 160	13.736			
8	4	8886.8	12 699 424	12.699			
16	1	5159.0	11 823 168	11.823			
16	2	8098.0	9 788 800	9.789			
...	...	...	...	...	...	...	...
128	16	327 873.6	4 711 232	4.711			

Evaluation

Dynamic BP Trees

Leaf	Excess	Time (s)	Space (bits)	Space Overhead	Best Runtime	Best Space	Weighted
2	1	3609.0	46 403 904	46.404	1.000	9.850	
4	1	3653.8	24 423 744	24.424	1.012	5.184	
4	2	4658.2	22 252 864	22.253	1.291	4.723	
8	1	4058.6	15 809 696	15.810	1.125	3.356	
8	2	5731.2	13 736 160	13.736	1.588	2.916	
8	4	8886.8	12 699 424	12.699	2.462	2.696	
16	1	5159.0	11 823 168	11.823	1.429	2.510	
16	2	8098.0	9 788 800	9.789	2.244	2.078	
...	...	...	...	...	...	...	...
128	16	327 873.6	4 711 232	4.711	90.849	1.000	

Evaluation

Dynamic BP Trees

Leaf	Excess	Time (s)	Space (bits)	Space Overhead	Best Runtime	Best Space	Weighted
2	1	3609.0	46 403 904	46.404	1.000	9.850	5.867
4	1	3653.8	24 423 744	24.424	1.012	5.184	3.307
4	2	4658.2	22 252 864	22.253	1.291	4.723	3.179
8	1	4058.6	15 809 696	15.810	1.125	3.356	2.352
8	2	5731.2	13 736 160	13.736	1.588	2.916	2.318
8	4	8886.8	12 699 424	12.699	2.462	2.696	2.591
16	1	5159.0	11 823 168	11.823	1.429	2.510	2.024
16	2	8098.0	9 788 800	9.789	2.244	2.078	2.152
...	...	...	...	...	...	...	...
128	16	327 873.6	4 711 232	4.711	90.849	1.000	41.432

Evaluation

Dynamic BP Trees

Leaf	Excess	Time (s)	Space (bits)	Space Overhead	Best Runtime	Best Space	Weighted
2	1	3609.0	46 403 904	46.404	1.000	9.850	5.867
4	1	3653.8	24 423 744	24.424	1.012	5.184	3.307
4	2	4658.2	22 252 864	22.253	1.291	4.723	3.179
8	1	4058.6	15 809 696	15.810	1.125	3.356	2.352
8	2	5731.2	13 736 160	13.736	1.588	2.916	2.318
8	4	8886.8	12 699 424	12.699	2.462	2.696	2.591
16	1	5159.0	11 823 168	11.823	1.429	2.510	2.024
16	2	8098.0	9 788 800	9.789	2.244	2.078	2.152
...	...	...	...	...	...	...	...
128	16	327 873.6	4 711 232	4.711	90.849	1.000	41.432

Bibliography I

[Navarro, 2016] Navarro, G. (2016).
Compact Data Structures - A Practical Approach.
Cambridge University Press.

[Navarro and Sadakane, 2014] Navarro, G. and Sadakane, K. (2014).
Fully functional static and dynamic succinct trees.
ACM Trans. Algorithms, 10(3):16:1–16:39.