Some notes on data structures used in search algorithms

Freitagsseminar

Tobias Denkinger

tobias.denkinger@tu-dresden.de

Institute of Theoretical Computer Science
Faculty of Computer Science
Technische Universität Dresden

2018-11-09

Outline

1. Pushdowns
2. Log-domain
3. Double-ended priority queues

Pushdowns

- **Usage:** e.g. in pushdown automata; represent backlinks in graph search

[1] R. E. Tarjan (Apr. 1985). "Amortized Computational Complexity". *SIAM Journal on Algebraic Discrete Methods* 6.2, pp. 306–318. DOI: 10.1137/0606031

Pushdowns

- **Usage:** e.g. in pushdown automata; represent backlinks in graph search
- **Complexities:**

	growable array
pop, peek	$\mathcal{O}(1)$
push	$\mathcal{O}(1)$ (amortised[1])

[1] R. E. Tarjan (Apr. 1985). "Amortized Computational Complexity". *SIAM Journal on Algebraic Discrete Methods* 6.2, pp. 306–318. DOI: 10.1137/0606031

Pushdowns

- **Usage:** e.g. in pushdown automata; represent backlinks in graph search
- **Complexities:**

	growable array
pop, peek	$\mathcal{O}(1)$
push	$\mathcal{O}(1)$ (amortised[1])
examples	vector (C++), Vec (Rust)

[1] R. E. Tarjan (Apr. 1985). "Amortized Computational Complexity". *SIAM Journal on Algebraic Discrete Methods* 6.2, pp. 306–318. DOI: 10.1137/0606031

Pushdowns

- **Usage:** e.g. in pushdown automata; represent backlinks in graph search
- **Complexities:**

	growable array
pop, peek	$\mathcal{O}(1)$
push	$\mathcal{O}(1)$ (amortised[1])
clone	$\mathcal{O}(n)$
examples	vector (C++),
	Vec (Rust)

[1] R. E. Tarjan (Apr. 1985). “Amortized Computational Complexity”. *SIAM Journal on Algebraic Discrete Methods* 6.2, pp. 306–318. DOI: 10.1137/0606031

Pushdowns

- **Usage:** e.g. in pushdown automata; represent backlinks in graph search
- **Complexities:**

	growable array	*(lazy) linked list*
pop, peek	$\mathcal{O}(1)$	$\mathcal{O}(1)$
push	$\mathcal{O}(1)$ (amortised[1])	$\mathcal{O}(1)$
clone	$\mathcal{O}(n)$	$\mathcal{O}(1)$
examples	vector (C++), Vec (Rust)	

[1] R. E. Tarjan (Apr. 1985). “Amortized Computational Complexity”. *SIAM Journal on Algebraic Discrete Methods* 6.2, pp. 306–318. DOI: 10.1137/0606031

Pushdowns

- **Usage:** e.g. in pushdown automata; represent backlinks in graph search
- **Complexities:**

	growable array	*(lazy) linked list*
pop, peek	$\mathcal{O}(1)$	$\mathcal{O}(1)$
push	$\mathcal{O}(1)$ (amortised[1])	$\mathcal{O}(1)$
clone	$\mathcal{O}(n)$	$\mathcal{O}(1)$
examples	vector (C++), Vec (Rust)	List (Haskell),

[1] R. E. Tarjan (Apr. 1985). "Amortized Computational Complexity". *SIAM Journal on Algebraic Discrete Methods* 6.2, pp. 306–318. DOI: 10.1137/0606031

Pushdowns

- **Usage:** e.g. in pushdown automata; represent backlinks in graph search
- **Complexities:**

	growable array	*(lazy) linked list*
pop, peek	$\mathcal{O}(1)$	$\mathcal{O}(1)$
push	$\mathcal{O}(1)$ (amortised[1])	$\mathcal{O}(1)$
clone	$\mathcal{O}(n)$	$\mathcal{O}(1)$
examples	vector (C++),	List (Haskell),
	Vec (Rust)	Pushdown (Rustomata)

[1] R. E. Tarjan (Apr. 1985). "Amortized Computational Complexity". *SIAM Journal on Algebraic Discrete Methods* 6.2, pp. 306–318. DOI: 10.1137/0606031

Pushdowns – Rustomata

- **Idea:** share prefixes via pointer + copy on write [cf. `rustomata` on github]

Pushdowns – Rustomata

- **Idea:** share prefixes via pointer + copy on write [cf. `rustomata` on github]
- use algebraic data type (a.k.a. tagged union)

Haskell:

```haskell
data [a]
= []
| a : [a]
-- approximately
```

Rust:

```rust
pub enum Pushdown<A> {
  Empty,
  Cons {val: A, below: Rc<Pushdown<A>>},
}
```

Pushdowns – Rustomata

- **Idea:** share prefixes via pointer + copy on write [cf. rustomata on github]
- use algebraic data type (a.k.a. tagged union)

Haskell:

```haskell
data [a]
= []
| a : [a]
-- approximately
```

Rust:

```rust
pub enum Pushdown<A> {
  Empty,
  Cons {val: A, below: Rc<Pushdown<A>>},
}
```

- current element is always unique

```rust
pub fn pop(self) -> Result<(Self, A), Self> {
  match self {
    Pushdown::Empty              => Err(Pushdown::Empty            ),
    Pushdown::Cons{val,below} => Ok((below.deref().clone(), val)),
  }
}
```

Log-domain

- **Usage:** represent probabilities (especially small values)

Log-domain

- **Usage:** represent probabilities (especially small values)
- **Idea:** use bijection $\ln\colon \mathbb{R}_{\geq 0} \to \mathbb{R} \cup \{\infty\}$ [cf. `rust-log-domain` on github]

```
/// Same as `new`, but without bounds check.
fn new_unchecked(value: F) -> Self {
  LogDomain(value.ln())
}
```

Log-domain

- **Usage:** represent probabilities (especially small values)
- **Idea:** use bijection $\ln\colon \mathbb{R}_{\geq 0} \to \mathbb{R} \cup \{\infty\}$ [cf. `rust-log-domain` on github]

```
/// Same as `new`, but without bounds check.
fn new_unchecked(value: F) -> Self {
  LogDomain(value.ln())
}
```

- use *newtype* to avoid runtime overhead (e.g. in Haskell and Rust)

```
pub struct LogDomain<F: Float>(F);
```

Log-domain

- **Usage:** represent probabilities (especially small values)
- **Idea:** use bijection $\ln\colon \mathbb{R}_{\geq 0} \to \mathbb{R} \cup \{\infty\}$ [cf. `rust-log-domain` on github]

```
/// Same as `new`, but without bounds check.
fn new_unchecked(value: F) -> Self {
    LogDomain(value.ln())
}
```

- use *newtype* to avoid runtime overhead (e.g. in Haskell and Rust)

```
pub struct LogDomain<F: Float>(F);
```

- multiplication becomes addition $\Longrightarrow \ln(\cdot) = (+)$

```
fn mul(self, other: Self) -> Self {
    LogDomain(self.ln().add(other.ln()))
}
```

Log-domain

- **Usage:** represent probabilities (especially small values)
- **Idea:** use bijection ln: $\mathbb{R}_{\geq 0} \to \mathbb{R} \cup \{\infty\}$ [cf. `rust-log-domain` on github]

```
/// Same as `new`, but without bounds check.
fn new_unchecked(value: F) -> Self {
  LogDomain(value.ln())
}
```

- use *newtype* to avoid runtime overhead (e.g. in Haskell and Rust)

```
pub struct LogDomain<F: Float>(F);
```

- multiplication becomes addition $\Longrightarrow \ln(\cdot) = (+)$

```
fn mul(self, other: Self) -> Self {
  LogDomain(self.ln().add(other.ln()))
}
```

- What about $\ln(+)$?

Log-domain – What about $\ln(+)$?

- determine z s.t. $\texttt{LogDomain}(z) = \texttt{LogDomain}(x) + \texttt{LogDomain}(y)$

Log-domain – What about $\ln(+)$?

- determine z s.t. $\texttt{LogDomain}(z) = \texttt{LogDomain}(x) + \texttt{LogDomain}(y)$
  i.e. $\exp(z) = \exp(x) + \exp(y)$

Log-domain – What about ln(+)?

- determine z s.t. $\texttt{LogDomain}(z) = \texttt{LogDomain}(x) + \texttt{LogDomain}(y)$

  i.e. $\exp(z) = \exp(x) + \exp(y)$

$$z = \ln\left(\exp(x) + \exp(y)\right)$$

Log-domain – What about ln(+)?

- determine z s.t. $\texttt{LogDomain}(z) = \texttt{LogDomain}(x) + \texttt{LogDomain}(y)$
  i.e. $\exp(z) = \exp(x) + \exp(y)$

$$z = \texttt{ln}\left(\exp(x) + \exp(y)\right)$$ (3 transcendentals)

Log-domain – What about ln(+)?

- determine z s.t. $\texttt{LogDomain}(z) = \texttt{LogDomain}(x) + \texttt{LogDomain}(y)$
  i.e. $\exp(z) = \exp(x) + \exp(y)$

$$z = \ln\left(\exp(x) + \exp(y)\right) \qquad \text{(3 transcendentals)}$$
$$= \ln\left(\exp(x) + \exp(x) \cdot \frac{\exp(y)}{\exp(x)}\right)$$

Log-domain – What about $\ln(+)$?

- determine z s.t. $\mathtt{LogDomain}(z) = \mathtt{LogDomain}(x) + \mathtt{LogDomain}(y)$
  i.e. $\exp(z) = \exp(x) + \exp(y)$

$$
\begin{aligned}
z &= \ln\left(\exp(x) + \exp(y)\right) && \text{(3 transcendentals)} \\
&= \ln\left(\exp(x) + \exp(x) \cdot \frac{\exp(y)}{\exp(x)}\right) \\
&= \ln\left(\exp(x) + \exp(x) \cdot \exp(y - x)\right)
\end{aligned}
$$

Log-domain – What about $\ln(+)$?

- determine z s.t. $\texttt{LogDomain}(z) = \texttt{LogDomain}(x) + \texttt{LogDomain}(y)$
  i.e. $\texttt{exp}(z) = \texttt{exp}(x) + \texttt{exp}(y)$

$$
\begin{aligned}
z &= \texttt{ln}\left(\texttt{exp}(x) + \texttt{exp}(y)\right) && \text{(3 transcendentals)} \\
&= \texttt{ln}\left(\texttt{exp}(x) + \texttt{exp}(x) \cdot \frac{\texttt{exp}(y)}{\texttt{exp}(x)}\right) \\
&= \texttt{ln}\left(\texttt{exp}(x) + \texttt{exp}(x) \cdot \texttt{exp}(y - x)\right) \\
&= \texttt{ln}\left(\texttt{exp}(x) \cdot (1 + \texttt{exp}(y - x))\right)
\end{aligned}
$$

Log-domain – What about $\ln(+)$?

- determine z s.t. $\texttt{LogDomain}(z) = \texttt{LogDomain}(x) + \texttt{LogDomain}(y)$
  i.e. $\exp(z) = \exp(x) + \exp(y)$

$$
\begin{aligned}
z &= \ln\left(\exp(x) + \exp(y)\right) && \text{(3 transcendentals)} \\
&= \ln\left(\exp(x) + \exp(x) \cdot \frac{\exp(y)}{\exp(x)}\right) \\
&= \ln\left(\exp(x) + \exp(x) \cdot \exp(y - x)\right) \\
&= \ln\left(\exp(x) \cdot (1 + \exp(y - x))\right) \\
&= x + \ln\left(1 + \exp(y - x)\right)
\end{aligned}
$$

Log-domain – What about $\ln(+)$?

- determine z s.t. $\quad \texttt{LogDomain}(z) = \texttt{LogDomain}(x) + \texttt{LogDomain}(y)$

  i.e. $\quad \exp(z) = \exp(x) + \exp(y)$

$$\begin{aligned} z &= \ln\left(\exp(x) + \exp(y)\right) && \text{(3 transcendentals)} \\ &= \ln\left(\exp(x) + \exp(x) \cdot \frac{\exp(y)}{\exp(x)}\right) \\ &= \ln\left(\exp(x) + \exp(x) \cdot \exp(y - x)\right) \\ &= \ln\left(\exp(x) \cdot (1 + \exp(y - x))\right) \\ &= x + \ln\left(1 + \exp(y - x)\right) \\ &= x + \texttt{ln\_1p}\left(\exp(y - x)\right) \end{aligned}$$

Log-domain – What about $\ln(+)$?

- determine z s.t. $\quad \mathtt{LogDomain}(z) = \mathtt{LogDomain}(x) + \mathtt{LogDomain}(y)$

  i.e. $\quad \exp(z) = \exp(x) + \exp(y)$

$$
\begin{aligned}
z &= \ln\left(\exp(x) + \exp(y)\right) && \text{(3 transcendentals)} \\
&= \ln\left(\exp(x) + \exp(x) \cdot \frac{\exp(y)}{\exp(x)}\right) \\
&= \ln\left(\exp(x) + \exp(x) \cdot \exp(y - x)\right) \\
&= \ln\left(\exp(x) \cdot (1 + \exp(y - x))\right) \\
&= x + \ln\left(1 + \exp(y - x)\right) \\
&= x + \mathtt{ln\_1p}\left(\exp(y - x)\right) && \text{(2 transcendentals)}
\end{aligned}
$$

Log-domain – What about ln(+)?

- determine z s.t. $\texttt{LogDomain}(z) = \texttt{LogDomain}(x) + \texttt{LogDomain}(y)$
  i.e. $\texttt{exp}(z) = \texttt{exp}(x) + \texttt{exp}(y)$

$$\begin{aligned}
z &= \texttt{ln}\left(\texttt{exp}(x) + \texttt{exp}(y)\right) && \text{(3 transcendentals)} \\
&= \texttt{ln}\left(\texttt{exp}(x) + \texttt{exp}(x) \cdot \frac{\texttt{exp}(y)}{\texttt{exp}(x)}\right) \\
&= \texttt{ln}\left(\texttt{exp}(x) + \texttt{exp}(x) \cdot \texttt{exp}(y - x)\right) \\
&= \texttt{ln}\left(\texttt{exp}(x) \cdot (1 + \texttt{exp}(y - x))\right) \\
&= x + \texttt{ln}\left(1 + \texttt{exp}(y - x)\right) \\
&= x + \texttt{ln_1p}\left(\texttt{exp}(y - x)\right) && \text{(2 transcendentals)}
\end{aligned}$$

- implementation:

```
173 LogDomain(x + (y - x).exp().ln_1p())
```

Double-ended priority queues

- **Usage:** e.g. beam search

Double-ended priority queues

- **Usage:** e.g. beam search
- **Complexities:**

	min-heap
enqueue	$\mathcal{O}(\log n)$
peekMin	$\mathcal{O}(1)$
dequeueMin	$\mathcal{O}(\log n)$

Double-ended priority queues

- **Usage:** e.g. beam search
- **Complexities:**

	min-heap
enqueue	$\mathcal{O}(\log n)$
peekMin	$\mathcal{O}(1)$
dequeueMin	$\mathcal{O}(\log n)$
peekMax	$\mathcal{O}(n)$
dequeueMax	$\mathcal{O}(n)$

Double-ended priority queues

- **Usage:** e.g. beam search
- **Complexities:**

	min-heap	*balanced search tree*
enqueue	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$
peekMin	$\mathcal{O}(1)$	$\mathcal{O}(\log n)$
dequeueMin	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$
peekMax	$\mathcal{O}(n)$	$\mathcal{O}(\log n)$
dequeueMax	$\mathcal{O}(n)$	$\mathcal{O}(\log n)$

Double-ended priority queues

- **Usage:** e.g. beam search
- **Complexities:** [cf. Atkinson, Sack, Santoro, and Strothotte 1986]

	min-heap	*balanced search tree*	*min-max-heap*
enqueue	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$
peekMin	$\mathcal{O}(1)$	$\mathcal{O}(\log n)$	$\mathcal{O}(1)$
dequeueMin	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$
peekMax	$\mathcal{O}(n)$	$\mathcal{O}(\log n)$	$\mathcal{O}(1)$
dequeueMax	$\mathcal{O}(n)$	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$

Double-ended priority queues

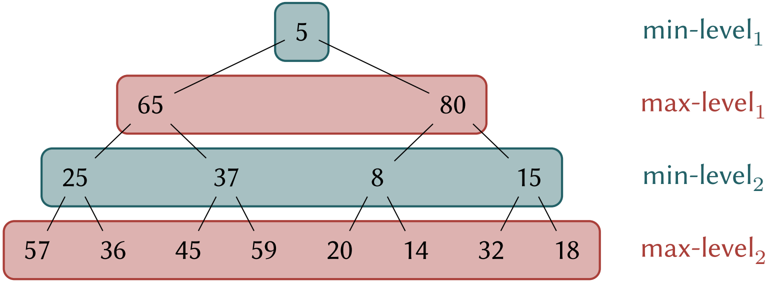
- **Usage:** e.g. beam search
- **Complexities:** [cf. Atkinson, Sack, Santoro, and Strothotte 1986]

	min-heap	*balanced search tree*	*min-max-heap*
enqueue	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$
peekMin	$\mathcal{O}(1)$	$\mathcal{O}(\log n)$	$\mathcal{O}(1)$
dequeueMin	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$
peekMax	$\mathcal{O}(n)$	$\mathcal{O}(\log n)$	$\mathcal{O}(1)$
dequeueMax	$\mathcal{O}(n)$	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$

- **Idea:** shuffle a max-heap into a min-heap

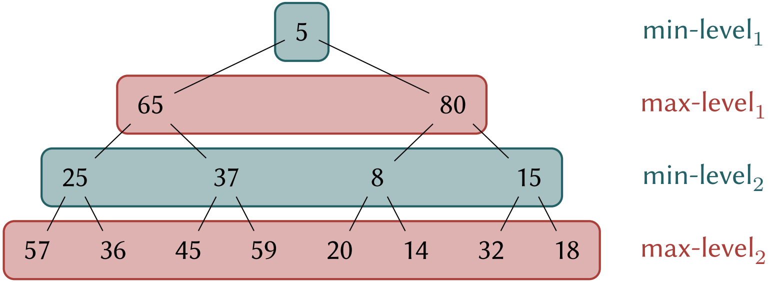
Double-ended priority queues – min-max-heap

- data structure:

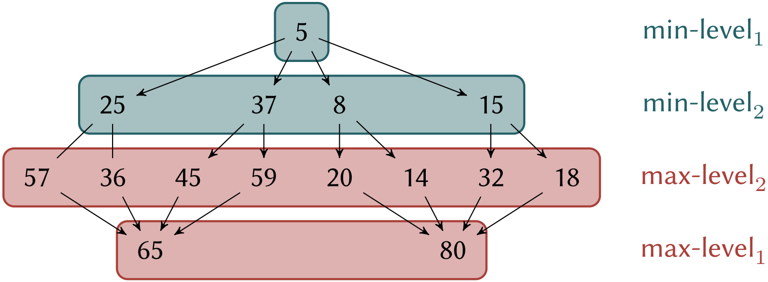


Double-ended priority queues – min-max-heap

- data structure:



- Hasse diagram:



References

M. D. Atkinson, J.-R. Sack, B. Santoro, and T. Strothotte (1986). "Min-max heaps and generalized priority queues". *Communications of the ACM*. DOI: 10.1145/6617.6621.

D. H. Larkin, S. Sen, and R. E. Tarjan (2013). "A Back-to-Basics Empirical Study of Priority Queues". DOI: 10.1137/1.9781611973198.7.

Professur GDP (2018a). *Log-domain in Rust (referenced version).* URL: https://github.com/tud-fop/rust-log-domain/blob/9568c5b7db6992fc11151829f236bc91337b182a/src/lib.rs.

Professur GDP (2018b). *Pushdowns in rustomata (referenced version).* URL: https://github.com/tud-fop/rustomata/blob/d8cb2798688ea1345735b08236441097e0757788/src/util/push_down.rs.

R. E. Tarjan (1985). "Amortized Computational Complexity". *SIAM Journal on Algebraic Discrete Methods*. DOI: 10.1137/0606031.