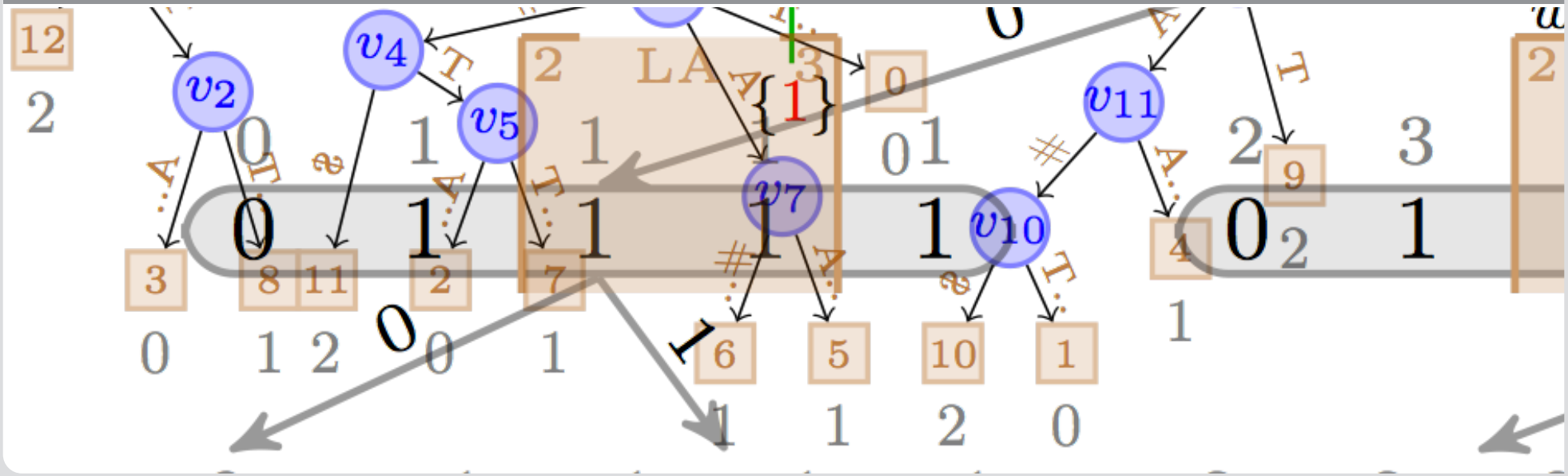
Karlsruhe Institute of Technology

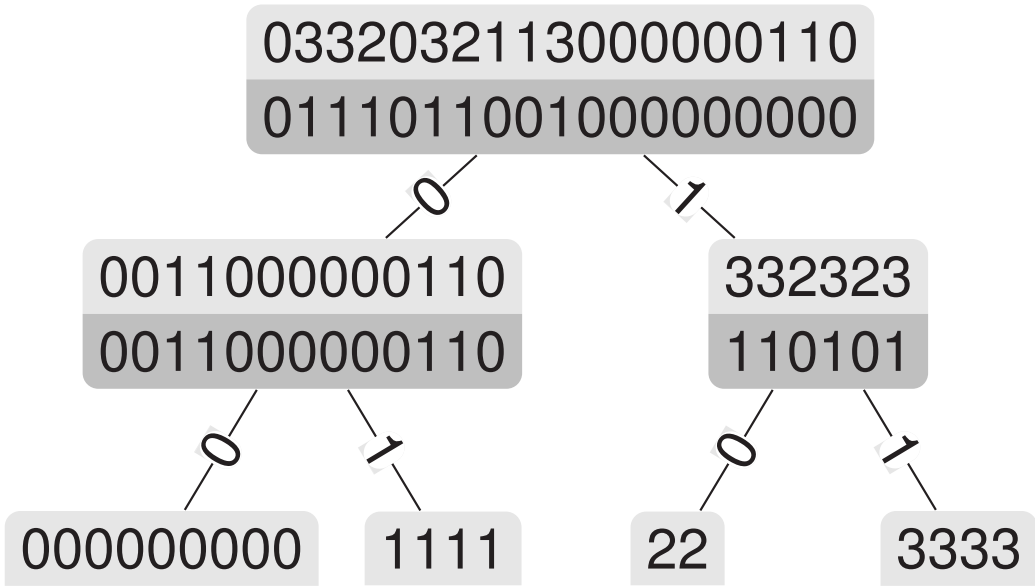
Advanced Data Structures

Simon Gog – *gog@kit.edu*

Institute of Theoretical Informatics - Algorithmics



Wavelet Trees (WTs)



Overview

- The Wavelet Tree (Grossi, Gupta, Vitter, SODA 2003) is a versatile sequence representation and can efficiently answer a number of queries:
  - access(i)
  - rank(i, c)
  - select(i, c)
  - range quantile queries
  - range next value
  - range intersection
  - $2d$ range counting and reporting

Literature

T. Gagie, G. Navarro, S.J. Puglisi: New algorithms on wavelet trees and applications to information retrieval. (TCS 2011)
G. Navarro: Wavelet trees for all. (CPM 2012)

Basic operations: $access(i, A)$

```
00  access(i, A)
01      return access(v_root, i)

00  access(v, i)
01      if is_leaf(v)
02          return symbol(v)
03      if B_v[i] = 0 then
04          return access(v_ℓ, rank(i, 0, B_v))
05      else
06          return access(v_r, rank(i, 1, B_v))
```

Basic operations: $rank(i, c, A)$

```
00  rank(i, c, A)
01      return rank(v_root, i, c)

00  rank(v, i, c)
01      if is_leaf(v)
02          return i
03      if c ∈ symbols(v_l) then
04          return rank(v_ℓ, rank(i, 0, B_v), c)
05      else
06          return rank(v_r, rank(i, 1, B_v), c)
```

Basic operations: $select(i, c, A)$

```
00  select(i, c, A)
01      return select(v_root, i, c)

00  select(v, i, c)
01      if is_leaf(v)
02          return i
03      if c ∈ symbols(v_l) then
04          return select(select(v_ℓ, i, c), 0, B_v)
05      else
04          return select(select(v_r, i, c), 1, B_v)
```

Note that we use two different select methods here.

Space

Empirical entropy

Let A be an array of length n over an alphabet Σ of size σ. Let n_c be the number of occurrences of symbol $c \in \Sigma$ in A. Then the *empirical entropy* of A is defined as follows:

$$H_0(A) = \sum_{c \in \Sigma} \frac{n_c}{n} \log \frac{n}{n_c}$$

- $H_0(T) \leq \log \sigma$
- For a bitvector B we get

$$H_0(B) = \frac{n_0}{n} \log \frac{n}{n_0} + \frac{n_1}{n} \log \frac{n}{n_1}$$

Space

- The are constant-time rank/select supported bitvectors which require only $nH_0(B) + o(n)$ bits
- How much space is required if we use such bitvectors for each node?
- Let v_ϵ be the root node which has n_0 zeros and n_1 ones, then the first level uses

$$n_0 \log \frac{n}{n_0} + n_1 \log \frac{n}{n_1} \text{ bits}$$

- Let v_0 (v_1) be the left (right) child and n_{01} (n_{00}) be the number of ones (zeros) in v_0 and n_{10} (n_{11}) the number of ones (zeros) in v_1
- The bitvector of v_0 takes

$$n_{00} \log \frac{n_0}{n_{00}} + n_{01} \log \frac{n_0}{n_{01}} \text{ bits}$$

- Similarly, the for v_1's bitvector we get

$$n_{10} \log \frac{n_1}{n_{10}} + n_{11} \log \frac{n_1}{n_{11}} \text{ bits}$$

Space

- Summing up the space of the three bitvector we get

$$
\begin{aligned}
& (n_{00} + n_{01}) \log \frac{n}{n_0} + (n_{10} + n_{11}) \log \frac{n}{n_1} \\
+ \quad & n_{00} \log \frac{n_0}{n_{00}} + n_{01} \log \frac{n_0}{n_{01}} \\
+ \quad & n_{10} \log \frac{n_1}{n_{10}} + n_{11} \log \frac{n_1}{n_{11}} \\
= \quad & n_{00} \log \frac{n}{n_{00}} + n_{01} \log \frac{n}{n_{01}} + n_{10} \log \frac{n}{n_{10}} + n_{11} \log \frac{n}{n_{11}}
\end{aligned}
$$

- I.e. $nH_0(T)$ bits for $\sigma = 4$. In general we get $nH_0(T) + o(n)$ bits for the bitvectors of a WT for array A.
- A WT based on compressed bitvectors represents A in compressed form.

Space

Questions

- How much space is used for the WT topology?
- Can we improve the space for this part?

Another basic helper operations: $expand(v, i, j)$

Expand v maps a node and an interval to its child nodes and corresponding child intervals.

```
00  expand(v, i, j)
01      if is_leaf(v)
02          return ⊥
03      [i_ℓ, j_ℓ] ← [rank(i, 0, B_v), rank(j, 0, B_v)]
04      [i_r, j_r] ← [i − i_ℓ, j − j_ℓ]
05      return ⟨v_l, v_r, [i_ℓ, j_ℓ], [i_r, j_r]⟩
```

Range Quantile Queries

Karlsruhe Institute of Technology

Definition

A range quantile query $range\_quantile(i, j, k, A)$ returns for an sub-array $A[i, j]$ the k-th smallest element.

```
00  range_quantile(i, j, k, A)
01      return rqq(v_root, i, j, k)
```

Range Quantile Queries

```
00  range_quantile(i, j, k, A)
01      return rqq(v_root, i, j, k)

00  rqq(v, i, j, k)
01      if is_leaf(v)
02          return symbol(v)
03      ⟨v_ℓ, v_r, [i_ℓ, j_ℓ], [i_r, j_r]⟩ ← expand(v, i, j)
04      n_ℓ = j_ℓ − i_ℓ + 1
05      if k ≤ n_ℓ then
06          return rqq(v_ℓ, i_ℓ, j_ℓ, k)
07      else
08          return rqq(v_r, i_r, j_r, k − n_ℓ)
```

Range Next Value Queries

Karlsruhe Institute of Technology

Definition

A range next value query $\mathit{range\_next}(i, j, x, A)$ returns for an sub-array $A[i, j]$ the smallest element $A[r] \geq x$ with $r \in [i, j]$.

Range Next Value Queries

Karlsruhe Institute of Technology

Definition

A range next value query $range\_next(i, j, x, A)$ returns for an sub-array $A[i, j]$ the smallest element $A[r] \geq x$ with $r \in [i, j]$.

```
00  range_next(i, j, x, A)
01      return rnv(v_root, i, j, x)[0]
```

rnv returns a triple consisting of

- the smallest element $A[r] \geq x$ with $r \in [i, j]$
- the frequency of $A[r]$
- the smallest rank of $A[r]$ in $A[i, j]$

Range Next Value Queries

```
00  rnv(v, i, j, p, x)
01      if i > j then
02          return ⟨⊥, 0, 0⟩
03      if is_leaf(v)
04          return ⟨x, j − i + 1, p⟩
05      ⟨v_ℓ, v_r, [i_ℓ, j_ℓ], [i_r, j_r]⟩ ← expand(v, i, j)
06      n_ℓ = j_ℓ − i_ℓ + 1
07      if x ∈ symbols(v_r) then
08          return rnv(v_r, i_r, j_r, p + n_ℓ, x)
09      ⟨y, f, p′⟩ ← rnv(v_ℓ, i_ℓ, j_ℓ, p, x)
10      if y ≠ ⊥ then
11          return (y, f, p′)
12      return rnv(v_r, i_r, j_r, p + n_ℓ, min(symbols(v_r)))
```