KIT
Karlsruhe Institute of Technology

# Advanced Data Structures

## Lecture 11: Learned Data Structures

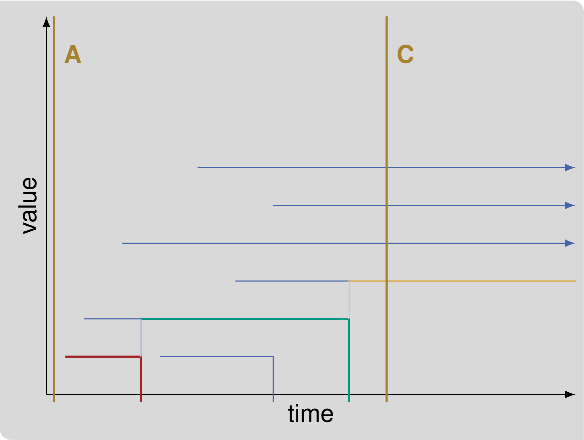
Florian Kurpicz

# PINGO

https://pingo.scc.kit.edu/524651

# Recap: Retroactive Data Structures

- BBST for $Q_\infty$ ⓘ changed for each update
- BBST where leaves are inserts ordered by time augmented with
  - for each node $x$ store $\max\{v' \notin Q_\infty : v' \text{ inserted in subtree of } x\}$
- BBST where leaves are all updates ordered by time augmented with
  - leaves store 0 for inserts with $v \in Q_\infty$, 1 for inserts with $v \notin Q_\infty$ and $-1$ for delete-mins
  - inner nodes store subtree sums
  - inner nodes store smallest prefix sum in subtree



# Setting: Rank and Select Dictionary

Given ordered integers $S$ from a universe $\mathcal{U} = [1, u]$ a rank and select index can answer

- $rank(x) = |\{y \in S \colon y < x\}|$
- $select(i) = S[\arg\min_j(rank(j) = i + 1)]$

- can be use to answer predecessor queries
- a bit vector with rank and select suffices

# Setting: Rank and Select Dictionary

Karlsruhe Institute of Technology

Given ordered integers $S$ from a universe $\mathcal{U} = [1, u]$ a rank and select index can answer

- $rank(x) = |\{y \in S \colon y < x\}|$
- $select(i) = S[\arg\min_j(rank(j) = i + 1)]$

- can be use to answer predecessor queries
- a bit vector with rank and select suffices

- bit vector requires $u + o(u)$ bits
- compressing bit vector to save space (if sparse)
- Elias-Fano requires how much space?
  **PINGO**

# Setting: Rank and Select Dictionary

Given ordered integers $S$ from a universe $\mathcal{U} = [1, u]$ a rank and select index can answer

- $rank(x) = |\{y \in S \colon y < x\}|$
- $select(i) = S[\arg\min_j(rank(j) = i + 1)]$

- can be use to answer predecessor queries
- a bit vector with rank and select suffices

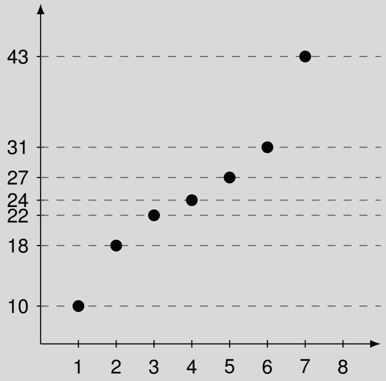
- bit vector requires $u + o(u)$ bits
- compressing bit vector to save space (if sparse)
- Elias-Fano requires how much space?

  **PINGO**

  - $|S|(2 + \log\lceil \frac{u}{|S|} \rceil)$ bits

# Learned Rank and Select Index [BFV21]

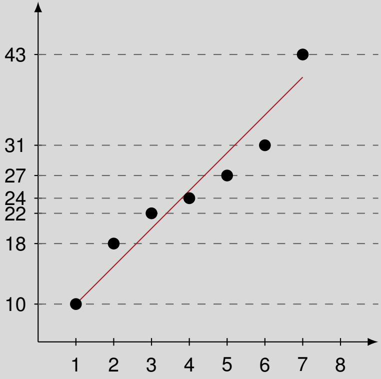
- how to represent $S$ from a universe $\mathcal{U} = [1, u]$
- let $S = \langle x_1, x_2, \ldots, x_n \rangle$ be a sorted sequence
- map each element $x_i \in S$ to point $(i, x_i)$
- points are in Cartesian plane




- $S = \langle 10, 18, 22, 24, 27, 31, 43 \rangle$

# Approximating $S$ in Cartesian Plane

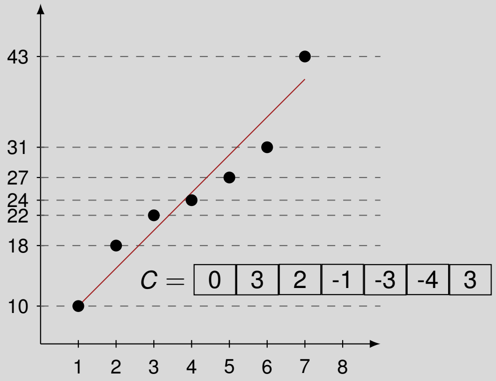
- find function $f$ passing through all points
  - $x_i = f(i)$
- BUT $f$ should be fast to compute and require little space
- use linear approximation with error $\epsilon$




- $S = \langle 10, 18, 22, 24, 27, 31, 43 \rangle$

# Approximating $S$ in Cartesian Plane

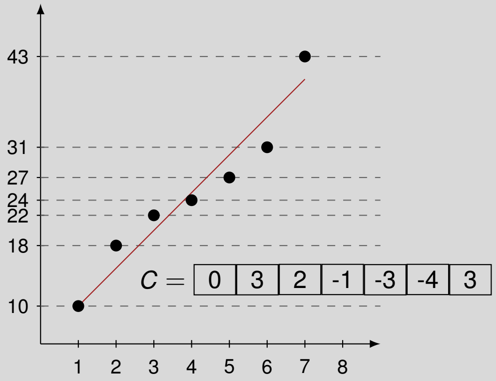
- find function $f$ passing through all points
  - $x_i = f(i)$
- BUT $f$ should be fast to compute and require little space
- use linear approximation with error $\epsilon$
- store error correction in array $C$




- $S = \langle 10, 18, 22, 24, 27, 31, 43 \rangle$

# Approximating $S$ in Cartesian Plane

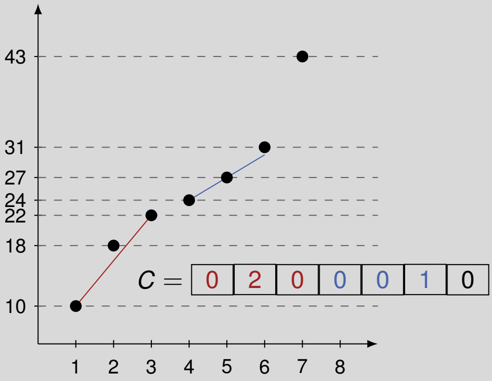
- find function $f$ passing through all points
  - $x_i = f(i)$
- BUT $f$ should be fast to compute and require little space
- use linear approximation with error $\epsilon$
- store error correction in array $C$
- correction can be very big




- $S = \langle 10, 18, 22, 24, 27, 31, 43 \rangle$

# Piece-Wise Linear Approximation (1/2)

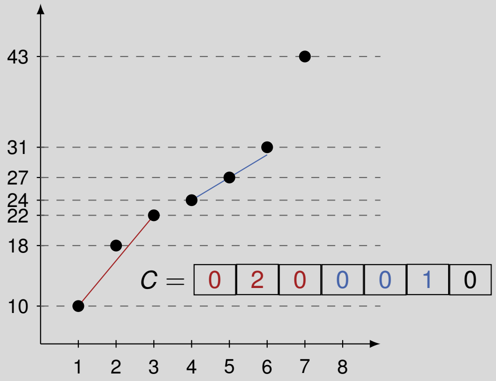
- use piece-wise linear approximation (PLA)
- sequence of segments with error bound by $\epsilon$
- smallest number of segments can be computed in $O(n)$ time [O'R81]
- let there be $\ell$ segments




- $S = \langle 10, 18, 22, 24, 27, 31, 43 \rangle$

# Piece-Wise Linear Approximation (1/2)

- use piece-wise linear approximation (PLA)
- sequence of segments with error bound by $\epsilon$
- smallest number of segments can be computed in $O(n)$ time [O'R81]
- let there be $\ell$ segments




- $S = \langle 10, 18, 22, 24, 27, 31, 43 \rangle$

## Definition: Representation of a Segment

The $i$-th segment starting with $(j, x_j)$ is represented as triple $s_i = (r_i, \alpha_i, \beta_i)$, where

- $r_i = j$,
- $\alpha_i$ is the slope, and
- $\beta_i$ is the intercept

# Piece-Wise Linear Approximation (2/2)

- use function to approximate point in $i$-th segment

$$f_i(j) = (j - r_i) \cdot \alpha_i + \beta_i$$

- use correction to obtain correct value

$$\lfloor f_i(j) \rfloor + C[j] = x_j$$

# Piece-Wise Linear Approximation (2/2)

- use function to approximate point in $i$-th segment

$$f_i(j) = (j - r_i) \cdot \alpha_i + \beta_i$$

- use correction to obtain correct value

$$\lfloor f_i(j) \rfloor + C[j] = x_j$$

- $C[j] = x_j - \lfloor f_i(j) \rfloor$
- $C[j] \in \{-\epsilon, -\epsilon + 1, \ldots, -1, 0, 1, \ldots, \epsilon - 1, \epsilon\}$

# Piece-Wise Linear Approximation (2/2)

- use function to approximate point in $i$-th segment

$$f_i(j) = (j - r_i) \cdot \alpha_i + \beta_i$$

- use correction to obtain correct value

$$\lfloor f_i(j) \rfloor + C[j] = x_j$$

- $C[j] = x_j - \lfloor f_i(j) \rfloor$
- $C[j] \in \{-\epsilon, -\epsilon + 1, \ldots, -1, 0, 1, \ldots, \epsilon - 1, \epsilon\}$

- let $c \geq 2$ be the number of bits used per correction
- $\epsilon = 2^c - 1$
- $c = 0$ results in $\epsilon = 0$

# Piece-Wise Linear Approximation (2/2)

Karlsruhe Institute of Technology

- use function to approximate point in $i$-th segment

$$f_i(j) = (j - r_i) \cdot \alpha_i + \beta_i$$

- use correction to obtain correct value

$$\lfloor f_i(j) \rfloor + C[j] = x_j$$

- $C[j] = x_j - \lfloor f_i(j) \rfloor$
- $C[j] \in \{-\epsilon, -\epsilon + 1, \dots, -1, 0, 1, \dots, \epsilon - 1, \epsilon\}$

- let $c \geq 2$ be the number of bits used per correction
- $\epsilon = 2^c - 1$
- $c = 0$ results in $\epsilon = 0$
- $c = 1$ possible? **PINGO**

# Piece-Wise Linear Approximation (2/2)

- use function to approximate point in $i$-th segment

$$f_i(j) = (j - r_i) \cdot \alpha_i + \beta_i$$

- use correction to obtain correct value

$$\lfloor f_i(j) \rfloor + C[j] = x_j$$

- $C[j] = x_j - \lfloor f_i(j) \rfloor$
- $C[j] \in \{-\epsilon, -\epsilon + 1, \ldots, -1, 0, 1, \ldots, \epsilon - 1, \epsilon\}$

- let $c \geq 2$ be the number of bits used per correction
- $\epsilon = 2^c - 1$
- $c = 0$ results in $\epsilon = 0$
- $c = 1$ possible? **PINGO**

- what time does it take to recover $x_j$? **PINGO**

# Piece-Wise Linear Approximation (2/2)

- use function to approximate point in $i$-th segment

$$f_i(j) = (j - r_i) \cdot \alpha_i + \beta_i$$

- use correction to obtain correct value

$$\lfloor f_i(j) \rfloor + C[j] = x_j$$

- $C[j] = x_j - \lfloor f_i(j) \rfloor$
- $C[j] \in \{-\epsilon, -\epsilon + 1, \ldots, -1, 0, 1, \ldots, \epsilon - 1, \epsilon\}$

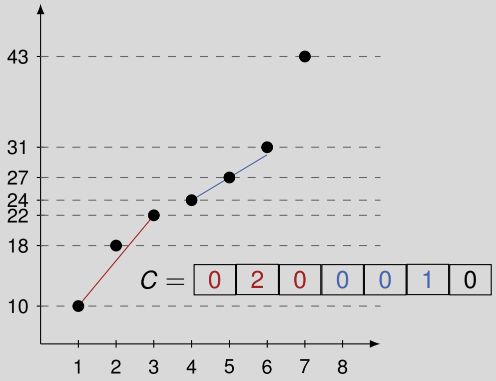
- let $c \geq 2$ be the number of bits used per correction
- $\epsilon = 2^c - 1$
- $c = 0$ results in $\epsilon = 0$
- $c = 1$ possible? **PINGO**

- what time does it take to recover $x_j$? **PINGO**
- $O(\log \ell)$ time to find the segment
- constant time within segment

# What is Missing?

- use linear functions to approximate values
- corrections allow recovering values

- compression of data structure
- rank and select support
- (space-)optimal segmentation




- $S = \langle 10, 18, 22, 24, 27, 31, 43 \rangle$

# Compressing the Representation (1/2)

- larger $\epsilon$ results in smaller "expected" number of segments $\ell$
- smaller $c$ results in smaller correction and in larger $\ell$ ⓘ $\epsilon = \max\{0, 2^c - 1\}$

# Compressing the Representation (1/2)

Karlsruhe Institute of Technology

- larger $\epsilon$ results in smaller “expected” number of segments $\ell$
- smaller $c$ results in smaller correction and in larger $\ell$ ℹ $\epsilon = \max\{0, 2^c - 1\}$

- $\ell$ depends on the distribution of points
- $\ell \leq \min\{u/(2\epsilon), n/2\}$ [FV20]

# Compressing the Representation (1/2)

- larger $\epsilon$ results in smaller "expected" number of segments $\ell$
- smaller $c$ results in smaller correction and in larger $\ell$ ⓘ $\epsilon = \max\{0, 2^c - 1\}$

- $\ell$ depends on the distribution of points
- $\ell \leq \min\{u/(2\epsilon), n/2\}$ [FV20]

**Definition: Number of Segments**

Let $\{x_1, \ldots, x_n\}$ be a sorted sequence of distinct integers from in $[1, u]$. Given $0 \leq c \leq \lfloor \log u \rfloor$, there are $\ell$ segments in the optimal PLA of maximum error $\epsilon = \max\{0, 2^c - 1\}$ for $\{(i, x_i) \colon i = 1, \ldots, n\}$

# Compressing the Representation (1/2)

- larger $\epsilon$ results in smaller "expected" number of segments $\ell$
- smaller $c$ results in smaller correction and in larger $\ell$ ℹ $\epsilon = \max\{0, 2^c - 1\}$

- $\ell$ depends on the distribution of points
- $\ell \leq \min\{u/(2\epsilon), n/2\}$ [FV20]

**Definition: Number of Segments**

Let $\{x_1, \ldots, x_n\}$ be a sorted sequence of distinct integers from in $[1, u]$. Given $0 \leq c \leq \lfloor \log u \rfloor$, there are $\ell$ segments in the optimal PLA of maximum error $\epsilon = \max\{0, 2^c - 1\}$ for $\{(i, x_i) : i = 1, \ldots, n\}$

**Lemma: Space-Requirements (uncompressed)**

$\{x_1, \ldots, x_n\}$ as defined before can be represented using $nc + 2\ell(\log n + \log u)$ bits of space.

# Compressing the Representation (1/2)

- larger $\epsilon$ results in smaller "expected" number of segments $\ell$
- smaller $c$ results in smaller correction and in larger $\ell$ ⓘ $\epsilon = \max\{0, 2^c - 1\}$

- $\ell$ depends on the distribution of points
- $\ell \leq \min\{u/(2\epsilon), n/2\}$ [FV20]

**Definition: Number of Segments**

Let $\{x_1, \ldots, x_n\}$ be a sorted sequence of distinct integers from in $[1, u]$. Given $0 \leq c \leq \lfloor \log u \rfloor$, there are $\ell$ segments in the optimal PLA of maximum error $\epsilon = \max\{0, 2^c - 1\}$ for $\{(i, x_i) \colon i = 1, \ldots, n\}$

**Lemma: Space-Requirements (uncompressed)**

$\{x_1, \ldots, x_n\}$ as defined before can be represented using $nc + 2\ell(\log n + \log u)$ bits of space.

**Proof (Sketch)**

Each segment $s_i = (r_i, \alpha_i, \beta_i)$ requires

- $r_i$: $\log n$ bits of space,
- $\alpha_i$: $\log u + \log n$ bits of space ⓘ rational number
- $\beta_i$: $\log u$ bits of space

# Compressing the Representation (1/2)

- larger $\epsilon$ results in smaller "expected" number of segments $\ell$
- smaller $c$ results in smaller correction and in larger $\ell$ ℹ $\epsilon = \max\{0, 2^c - 1\}$

- $\ell$ depends on the distribution of points
- $\ell \leq \min\{u/(2\epsilon), n/2\}$ [FV20]

**Definition: Number of Segments**

Let $\{x_1, \ldots, x_n\}$ be a sorted sequence of distinct integers from in $[1, u]$. Given $0 \leq c \leq \lfloor \log u \rfloor$, there are $\ell$ segments in the optimal PLA of maximum error $\epsilon = \max\{0, 2^c - 1\}$ for $\{(i, x_i) \colon i = 1, \ldots, n\}$

**Lemma: Space-Requirements (uncompressed)**

$\{x_1, \ldots, x_n\}$ as defined before can be represented using $nc + 2\ell(\log n + \log u)$ bits of space.

**Proof (Sketch)**

Each segment $s_i = (r_i, \alpha_i, \beta_i)$ requires

- $r_i$: $\log n$ bits of space,
- $\alpha_i$: $\log u + \log n$ bits of space ℹ rational number
- $\beta_i$: $\log u$ bits of space

- can we compress $r_i$'s, $\alpha_i$'s, or $\beta_i$'s? **PINGO**

# Compressing the Representation (2/2)

**Lemma: Space-Requirements (Elias-Fano)**

$\{x_1, \ldots, x_n\}$ as defined before can be represented using $nc + \ell(2\log\frac{un}{\ell} + 4 + o(1))$ bits of space.

# Compressing the Representation (2/2)

**Lemma: Space-Requirements (Elias-Fano)**

$\{x_1, \ldots, x_n\}$ as defined before can be represented using $nc + \ell(2 \log \frac{un}{\ell} + 4 + o(1))$ bits of space.

**Proof (Sketch)**

- $r_i$'s are increasing sequence of $\ell$ integers in $[1, n]$
- $\beta_i$'s are increasing sequence of $\ell$ integers in $[1, u]$
- use Elias-Fano coding

# Compressing the Representation (2/2)

**Lemma: Space-Requirements (Elias-Fano)**

$\{x_1, \ldots, x_n\}$ as defined before can be represented using $nc + \ell(2\log\frac{un}{\ell} + 4 + o(1))$ bits of space.

**Proof (Sketch)**

- $r_i$'s are increasing sequence of $\ell$ integers in $[1, n]$
- $\beta_i$'s are increasing sequence of $\ell$ integers in $[1, u]$
- use Elias-Fano coding

- $C$ can also be compressed
- using entropy compressed indices

# Compressing the Representation (2/2)

**Lemma: Space-Requirements (Elias-Fano)**

$\{x_1, \ldots, x_n\}$ as defined before can be represented using $nc + \ell(2 \log \frac{un}{\ell} + 4 + o(1))$ bits of space.

**Proof (Sketch)**

- $r_i$'s are increasing sequence of $\ell$ integers in $[1, n]$
- $\beta_i$'s are increasing sequence of $\ell$ integers in $[1, u]$
- use Elias-Fano coding

- $C$ can also be compressed
- using entropy compressed indices

**Lemma: Space-Requirements (Compressed)**

$\{x_1, \ldots, x_n\}$ as defined before can be represented using $nH_0(C) + o(nc) + \ell(2 \log \frac{un}{\ell} + 4 + o(1))$ bits of space. Access time is $O(c)$.

# Rank and Select Support

- rank and select use predecessor data structure on $r_i$'s
- select is "easier" than rank

**Lemma: Learned Select**

Select on $\{x_1, \ldots, x_n\}$ as defined before is supported in $O(1)$ time requiring $n(c + 1 + o(1)) + \ell(2 \log u + \log n)$ bits of space.

# Rank and Select Support

- rank and select use predecessor data structure on $r_i$'s
- select is "easier" than rank

**Lemma: Learned Select**

Select on $\{x_1, \ldots, x_n\}$ as defined before is supported in $O(1)$ time requiring $n(c + 1 + o(1)) + \ell(2 \log u + \log n)$ bits of space.

**Proof (Sketch)**

- use bit vector marking $r_i$'s
- $n + o(n)$ bits of space
- about one bit per element in $S$

# Rank and Select Support

- rank and select use predecessor data structure on $r_i$'s
- select is "easier" than rank

- naive $\mathit{rank}(x)$ needs binary search
- find maximum $i$ with $\mathit{select}(i) \leq x$
- requires $O(\log n)$ time

**Lemma: Learned Select**

Select on $\{x_1, \ldots, x_n\}$ as defined before is supported in $O(1)$ time requiring $n(c + 1 + o(1)) + \ell(2 \log u + \log n)$ bits of space.

**Proof (Sketch)**

- use bit vector marking $r_i$'s
- $n + o(n)$ bits of space
- about one bit per element in $S$

# Rank and Select Support

- rank and select use predecessor data structure on $r_i$'s
- select is "easier" than rank

**Lemma: Learned Select**

Select on $\{x_1, \ldots, x_n\}$ as defined before is supported in $O(1)$ time requiring $n(c + 1 + o(1)) + \ell(2 \log u + \log n)$ bits of space.

**Proof (Sketch)**

- use bit vector marking $r_i$'s
- $n + o(n)$ bits of space
- about one bit per element in $S$

- naive $rank(x)$ needs binary search
- find maximum $i$ with $select(i) \leq x$
- requires $O(\log n)$ time

- better: binary search on segments
- within segment: get "position" of $x$
- use maximum error to find interval for binary search

# Rank and Select Support

Karlsruhe Institute of Technology

- rank and select use predecessor data structure on $r_i$'s
- select is "easier" than rank

**Lemma: Learned Select**

Select on $\{x_1, \dots, x_n\}$ as defined before is supported in $O(1)$ time requiring $n(c + 1 + o(1)) + \ell(2 \log u + \log n)$ bits of space.

**Proof (Sketch)**

- use bit vector marking $r_i$'s
- $n + o(n)$ bits of space
- about one bit per element in $S$

- naive $\mathit{rank}(x)$ needs binary search
- find maximum $i$ with $\mathit{select}(i) \leq x$
- requires $O(\log n)$ time

- better: binary search on segments
- within segment: get "position" of $x$
- use maximum error to find interval for binary search

**Lemma: Learned Rank**

Rank on $\{x_1, \dots, x_n\}$ as defined before is supported in $O(\log \ell + c)$ time requiring no additional space.

# Details on Rank Support

- error is bounded: $|f_j(i) - x_i| \leq \epsilon$
- search for $x_i \leq x < x_{i+1}$
- rank is one $i$ with $f_j(i) - \epsilon \leq x \leq f_j(i) + \epsilon$

$f_j(i) = (i - r_i) \cdot \alpha_j + \beta_j$

- $(i - r_j) \cdot \alpha_j + \beta_j - \epsilon \leq x < (i + 1 - r_j) \cdot \alpha_j + \beta_j + \epsilon$
- solve for $i$
- $\frac{x-\beta_j}{\alpha_j} + r_j - (\frac{\epsilon}{\alpha_j} + 1) < i \leq \frac{x-\beta_j}{\alpha_j} + r_j + (\frac{\epsilon}{\alpha_j})$

# Finding Optimal Data Partitioning

Karlsruhe Institute of Technology

- fixed number $c$ for corrections
- now: choose different error $\epsilon = \max\{0, 2^c - 1\}$ for each segment
- how to find optimal partitioning?

# Finding Optimal Data Partitioning

- fixed number $c$ for corrections
- now: choose different error $\epsilon = \max\{0, 2^c - 1\}$ for each segment
- how to find optimal partitioning?

- let $G$ be directed acyclic graph
- one node for each $\{x_1, \ldots, x_n\}$ plus sink node at end of sequence
- edge $(i, j)$ with weight $w(i, j, c)$ indicates that there exists a segment compressing $x_i, \ldots, x_j$ using $w(i, j, c) = (j - i)c + \kappa$ bits of space

# Finding Optimal Data Partitioning

- fixed number $c$ for corrections
- now: choose different error $\epsilon = \max\{0, 2^c - 1\}$ for each segment
- how to find optimal partitioning?

- let $G$ be directed acyclic graph
- one node for each $\{x_1, \ldots, x_n\}$ plus sink node at end of sequence
- edge $(i, j)$ with weight $w(i, j, c)$ indicates that there exists a segment compressing $x_i, \ldots, x_j$ using $w(i, j, c) = (j - i)c + \kappa$ bits of space

**Lemma: Optimal Partitioning**

The shortest path in $G$ from node 1 to $n + 1$ corresponds to the PLA with minimal cost for $\{x_1, \ldots, x_n\}$

# Finding Optimal Data Partitioning

- fixed number $c$ for corrections
- now: choose different error $\epsilon = \max\{0, 2^c - 1\}$ for each segment
- how to find optimal partitioning?

- let $G$ be directed acyclic graph
- one node for each $\{x_1, \ldots, x_n\}$ plus sink node at end of sequence
- edge $(i, j)$ with weight $w(i, j, c)$ indicates that there exists a segment compressing $x_i, \ldots, x_j$ using $w(i, j, c) = (j - i)c + \kappa$ bits of space

**Lemma: Optimal Partitioning**

The shortest path in $G$ from node 1 to $n + 1$ corresponds to the PLA with minimal cost for $\{x_1, \ldots, x_n\}$

- finding shortest using brute-force not feasible
- requires $O(n^2 \log u)$ time [O'R81]
- can be done in $O(n \log u)$ time
  - solution is at most $\kappa\ell$ bits larger than optimal solution

# From Encoding to Indexing

- data has been encoded (and compressed)
- now: indexing data
- in external memory model
- learned index is alternative to B-tree

**Definition: External Memory Model (Recap)**

- internal memory of $M$ words
- instances of size $N \gg M$
- unlimited external memory
- transfer blocks of size $B$ between memories

- measure number of blocks I/Os
- scanning $N$ elements: $\Theta(N/B)$
- sorting $N$ elements: $\Theta(\frac{N}{B} \log_{\frac{M}{B}} \frac{N}{B})$

# From Encoding to Indexing

- data has been encoded (and compressed)
- now: indexing data
- in external memory model
- learned index is alternative to B-tree

| Data Structure | Space | I/Os |
|---|---|---|
| B=tree | $\Theta(n)$ | $O(\log_B(n))$ |
| PGM-Index | $\Theta(m_{\text{opt}})$ | $O(\log_B(m_{\text{opt}}))$ |

- $m_{\text{opt}} \leq n$ is optimal number of segments

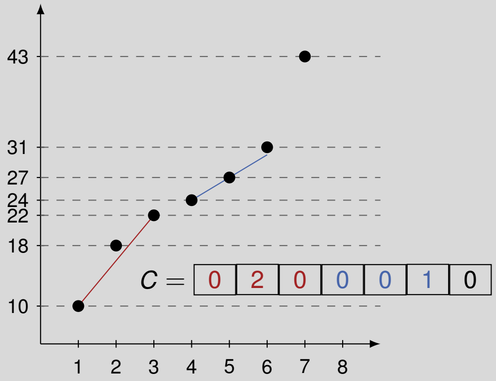
**Definition: External Memory Model (Recap)**

- internal memory of $M$ words
- instances of size $N \gg M$
- unlimited external memory
- transfer blocks of size $B$ between memories

- measure number of blocks I/Os
- scanning $N$ elements: $\Theta(N/B)$
- sorting $N$ elements: $\Theta(\frac{N}{B} \log_{\frac{M}{B}} \frac{N}{B})$

# The PGM-Index [FV20]

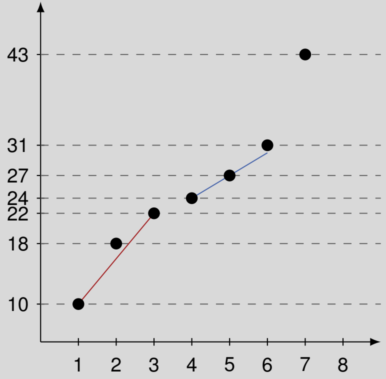
- what do we not need when indexing instead of encoding? **PINGO**




- $S = \langle 10, 18, 22, 24, 27, 31, 43 \rangle$

# The PGM-Index [FV20]

- what do we not need when indexing instead of encoding? **PINGO**
- now $S$ has to be stored
- how do we access elements in $S$
  - e.g., predecessor
- trick used before requires too much space

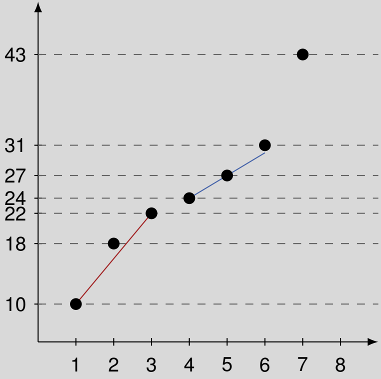



- $S = \langle 10, 18, 22, 24, 27, 31, 43 \rangle$

# The PGM-Index [FV20]

- what do we not need when indexing instead of encoding? **PINGO**
- now $S$ has to be stored
- how do we access elements in $S$
  - e.g., predecessor
- trick used before requires too much space

- store *key* instead position
- recurs on first *keys* of each segment




- $S = \langle 10, 18, 22, 24, 27, 31, 43 \rangle$

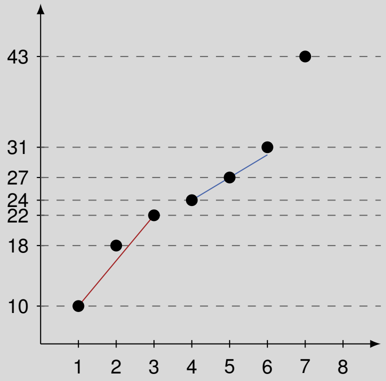
# The PGM-Index [FV20]

- what do we not need when indexing instead of encoding? **PINGO**
- now $S$ has to be stored
- how do we access elements in $S$
  - e.g., predecessor
- trick used before requires too much space

- store *key* instead position
- recurs on first *keys* of each segment

## For Queries

- $\epsilon = \Theta(B)$
- load $2\epsilon + 1$ blocks per level




- $S = \langle 10, 18, 22, 24, 27, 31, 43 \rangle$

# Evaluation

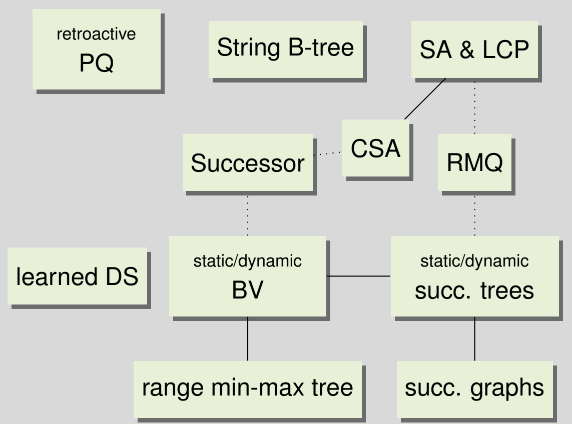
https://onlineumfrage.kit.edu/evasys/online/online.php?p=CZSUW

# Conclusion and Outlook

## This Lecture

- learned data structures

## Advanced Data Structures



# Conclusion and Outlook

## This Lecture

- learned data structures

## Next Lecture

- one more interesting data structure
- results of the project/competition
- Q&A

## Advanced Data Structures