Zip-Tries: Simple Dynamic Data Structures for Strings

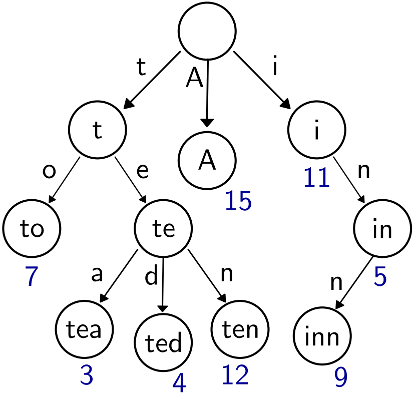
David Eppstein, *Ofek Gila*, Michael T. Goodrich, and Ryuto Kitagawa

University of California, Irvine

ACDA, 2025

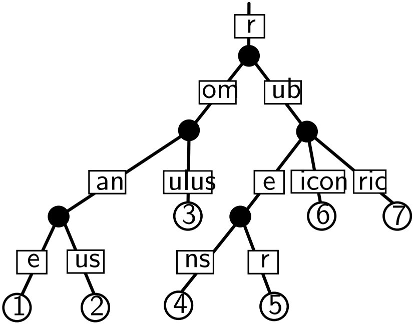
Selected History I

- 1912 / 1959 / 1960 – Trie (Thue, de la Braindais, Fredkin) [3, 5]
  - Query time: $\Theta(k)$



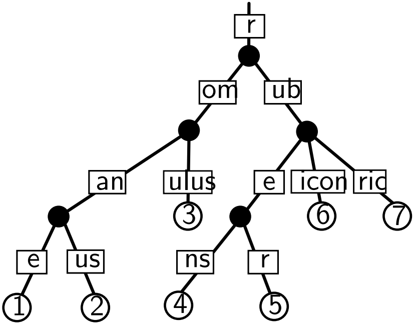
Selected History I

- 1912 / 1959 / 1960 – Trie (Thue, de la Braindais, Fredkin) [3, 5]
  - Query time: $\Theta(k)$
- 1968 – Compressed trie / radix tree (Morrison, Gwehenberger) [8]
  - Query time: $\mathcal{O}(k)$



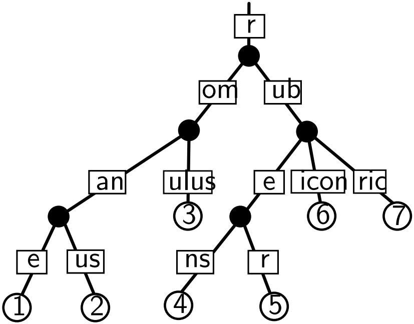
Selected History I

- 1912 / 1959 / 1960 – Trie (Thue, de la Braindais, Fredkin) [3, 5]
  - Query time: $\Theta(k)$
- 1968 – Compressed trie / radix tree (Morrison, Gwehenberger) [8]
  - Query time: $\mathcal{O}(k)$
- 2010+ – Dynamic z-fast tries, packed compact tries, etc. [1, 10]
  - Query time: $\mathcal{O}(k/w + \log k)$



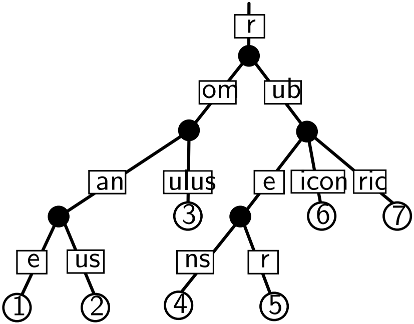
Selected History I

- 1912 / 1959 / 1960 – Trie (Thue, de la Braindais, Fredkin) [3, 5]
    - Query time: $\Theta(k)$
- 1968 – Compressed trie / radix tree (Morrison, Gwehenberger) [8]
    - Query time: $\mathcal{O}(k)$
- 2010+ – Dynamic z-fast tries, packed compact tries, etc. [1, 10]
    - Query time: $\mathcal{O}(k/w + \log k)$
    - Very fast, very complex



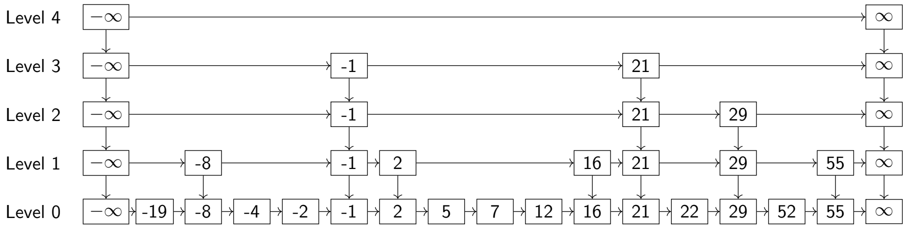
Selected History I

- 1912 / 1959 / 1960 – Trie (Thue, de la Braindais, Fredkin) [3, 5]
  - Query time: $\Theta(k)$
- 1968 – Compressed trie / radix tree (Morrison, Gwehenberger) [8]
  - Query time: $\mathcal{O}(k)$
- 2010+ – Dynamic z-fast tries, packed compact tries, etc. [1, 10]
  - Query time: $\mathcal{O}(k/w + \log k)$
  - Very fast, very complex
  - Lots of branching, $\mathcal{O}(k/w)$



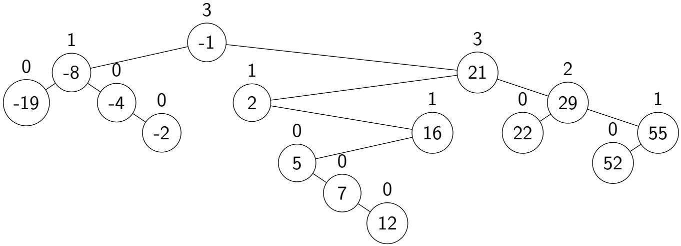
Selected History II

- Meanwhile. . .
- 1989 – Skip list (Pugh) [9]



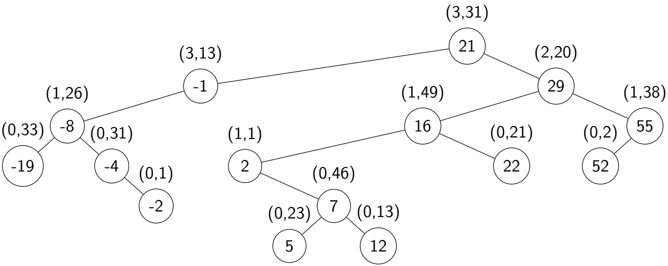
Selected History II

- Meanwhile...
- 1989 – Skip list (Pugh) [9]
- 2018 – Zip tree (Tarjan, Levy, Timmel) [11]
  - Flat-out better than skip lists
  - Only $\mathcal{O}(\log \log n)$ bits metadata per node



Selected History II

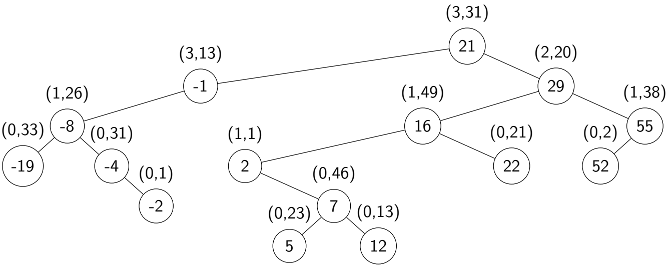
- Meanwhile. . .
- 1989 – Skip list (Pugh) [9]
- 2018 – Zip tree (Tarjan, Levy, Timmel) [11]
  - Flat-out better than skip lists
  - Only $\mathcal{O}(\log \log n)$ bits metadata per node
- 2023 – Zip-zip tree (Gila, Goodrich, Tarjan) [6]
  - Flat-out better than zip trees



Selected History II

- Meanwhile. . .
- 1989 – Skip list (Pugh) [9]
- 2018 – Zip tree (Tarjan, Levy, Timmel) [11]
  - Flat-out better than skip lists
  - Only $\mathcal{O}(\log\log n)$ bits metadata per node
- 2023 – Zip-zip tree (Gila, Goodrich, Tarjan) [6]
  - Flat-out better than zip trees

☑ Simple, practical, and efficient

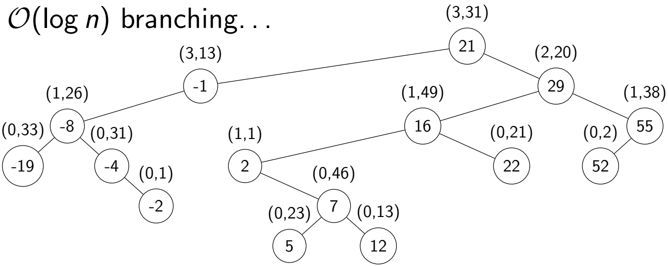


Selected History II

- Meanwhile. . .
- 1989 – Skip list (Pugh) [9]
- 2018 – Zip tree (Tarjan, Levy, Timmel) [11]
  - Flat-out better than skip lists
  - Only $\mathcal{O}(\log \log n)$ bits metadata per node
- 2023 – Zip-zip tree (Gila, Goodrich, Tarjan) [6]
  - Flat-out better than zip trees

☑ Simple, practical, and efficient

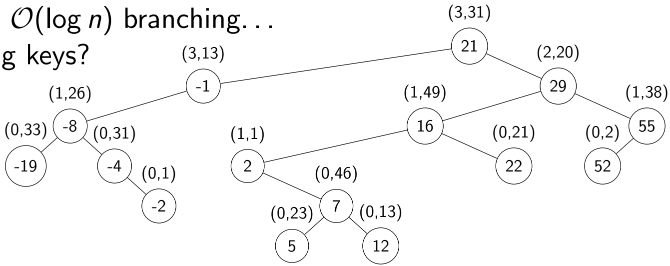
☑ Low, $\mathcal{O}(\log n)$ branching. . .



Selected History II

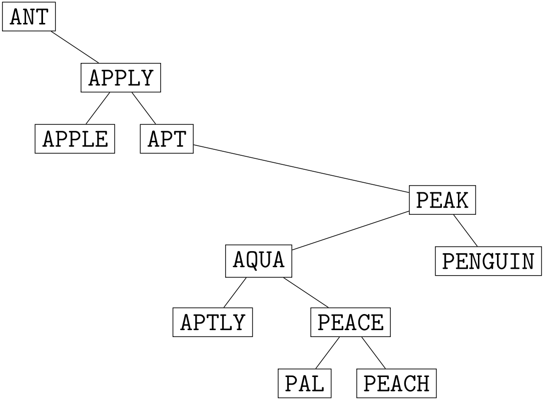
- Meanwhile...
- 1989 – Skip list (Pugh) [9]
- 2018 – Zip tree (Tarjan, Levy, Timmel) [11]
  - Flat-out better than skip lists
  - Only $\mathcal{O}(\log \log n)$ bits metadata per node
- 2023 – Zip-zip tree (Gila, Goodrich, Tarjan) [6]
  - Flat-out better than zip trees

☑ Simple, practical, and efficient
☑ Low, $\mathcal{O}(\log n)$ branching...
? String keys?



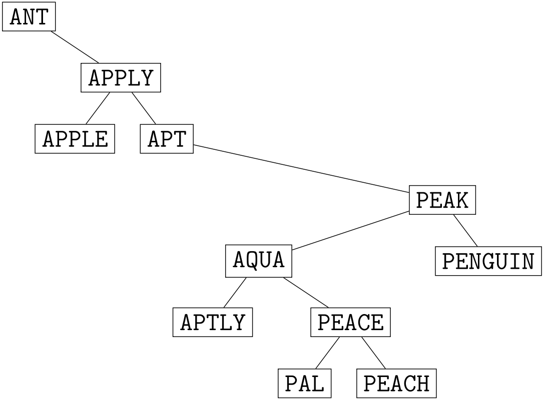
Selected History – Bookend Search I

- Many structures for 1D keys. . .



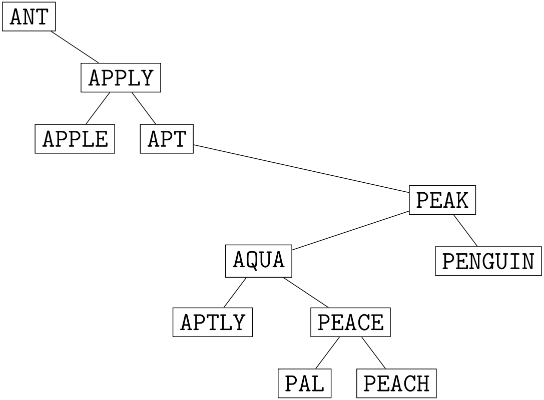
Selected History – Bookend Search I

- Many structures for 1D keys. . . k-D keys?



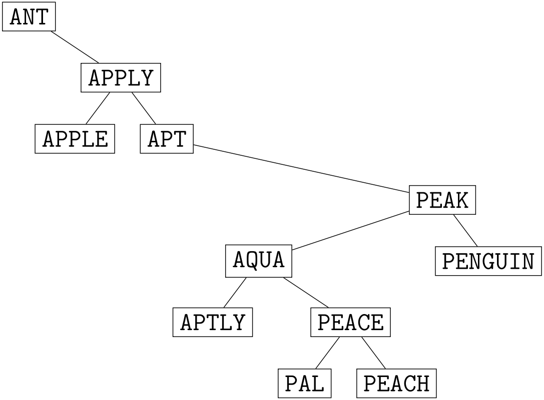
Selected History – Bookend Search I

- Many structures for 1D keys... k-D keys?
- Naively: $\mathcal{O}(\mathcal{B}(n))$ search $\rightarrow \mathcal{O}(k \times \mathcal{B}(n))$



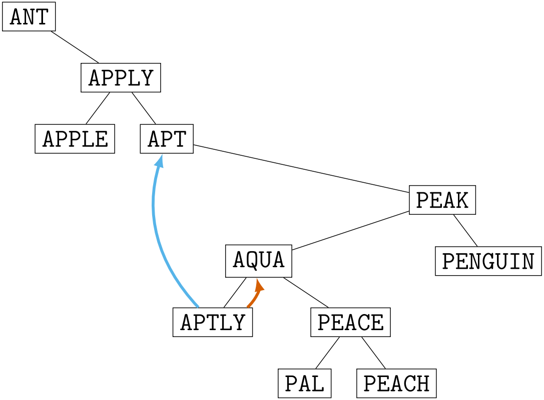
Selected History – Bookend Search I

- Many structures for 1D keys... k-D keys?
- Naively: $\mathcal{O}(\mathcal{B}(n))$ search $\rightarrow \mathcal{O}(k \times \mathcal{B}(n))$
- Grossi and Italiano [7] (2002), $\mathcal{O}(k + \mathcal{B}(n))$



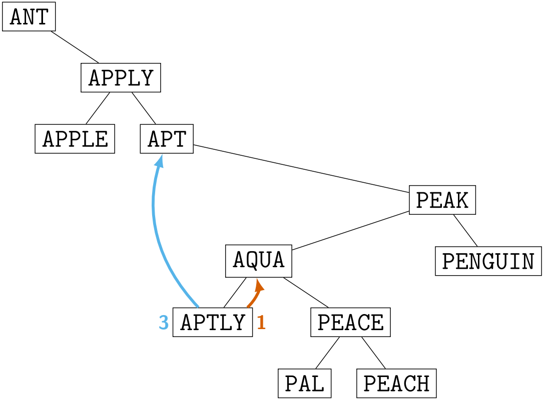
Selected History – Bookend Search I

- Many structures for 1D keys... k-D keys?
- Naively: $\mathcal{O}(\mathcal{B}(n))$ search $\rightarrow \mathcal{O}(k \times \mathcal{B}(n))$
- Grossi and Italiano [7] (2002), $\mathcal{O}(k + \mathcal{B}(n))$
  - Pointers to lexical predecessor and successor



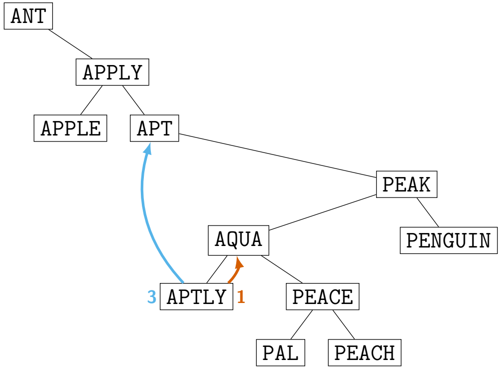
Selected History – Bookend Search I

- Many structures for 1D keys... k-D keys?
- Naively: $\mathcal{O}(\mathcal{B}(n))$ search $\to \mathcal{O}(k \times \mathcal{B}(n))$
- Grossi and Italiano [7] (2002), $\mathcal{O}(k + \mathcal{B}(n))$
  - Pointers to lexical predecessor and successor
  - *Longest Common Prefix* (LCP)



Selected History – Bookend Search II

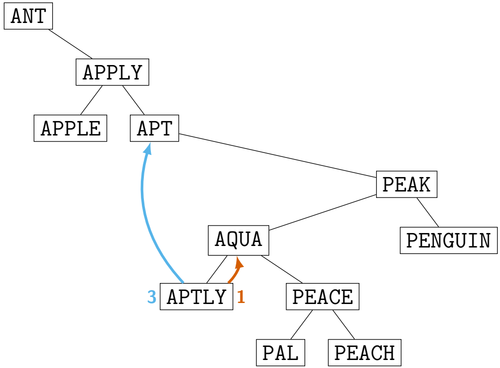
- Avoid unnecessary comparisons, $\mathcal{O}(k + \mathcal{B}(n))$



Selected History – Bookend Search II

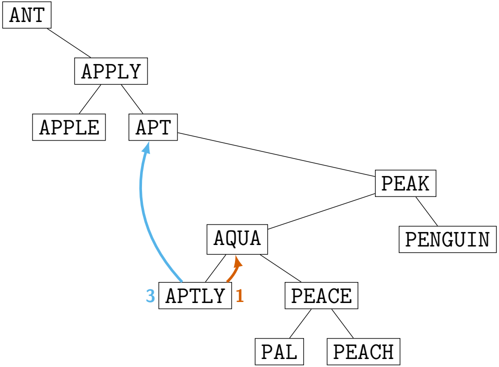
- Avoid unnecessary comparisons, $\mathcal{O}(k + \mathcal{B}(n))$

✗ Not 'best' in theory



Selected History – Bookend Search II

- Avoid unnecessary comparisons, $\mathcal{O}(k + \mathcal{B}(n))$
- ✗ Not 'best' in theory
- ☑ Simple, practical, & efficient [2]



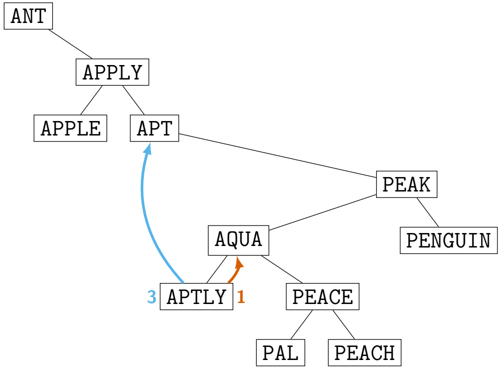
Selected History – Bookend Search II

- Avoid unnecessary comparisons, $\mathcal{O}(k + \mathcal{B}(n))$

✗ Not 'best' in theory

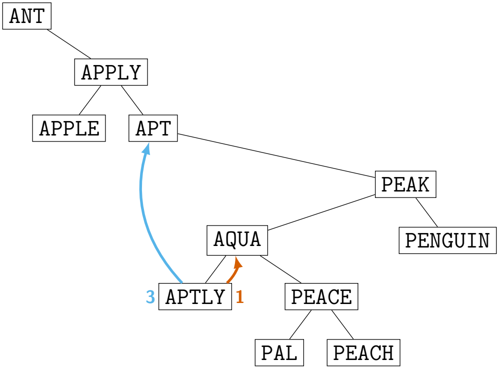
☑ Simple, practical, & efficient [2]

☑ Branching can be low, $\mathcal{O}(\log n)$ or $\mathcal{O}\left(\frac{\log n}{\log \log n}\right)$ [4]



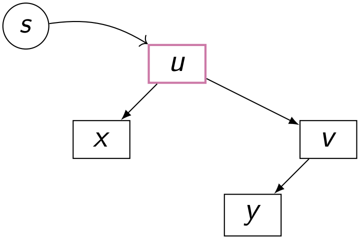
Selected History – Bookend Search II

- Avoid unnecessary comparisons, $\mathcal{O}(k + \mathcal{B}(n))$
- ☒ Not 'best' in theory
- ☑ Simple, practical, & efficient [2]
- ☑ Branching can be low, $\mathcal{O}(\log n)$ or $\mathcal{O}\left(\frac{\log n}{\log\log n}\right)$ [4]
  - ? Parallel comparisons?



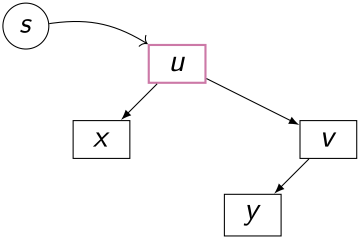
Sequential Longest Common Prefix

- Best way to find LCP?
- $k = |s| = 38$
- # computations: 0
- # branching: 0



Sequential Longest Common Prefix

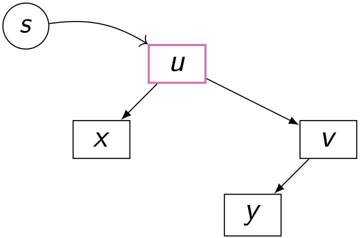
- Best way to find LCP?
- $k = |s| = 38$
- # computations: 0
- # branching: 0



$s =$ G G C T A G C T A C A A C G A G C T G G G C A G C A G C T A G T A G T A G G
$u =$ G G C T A A C T A C A A C G A G C T G G C G T T G T G A G C T A C

Sequential Longest Common Prefix

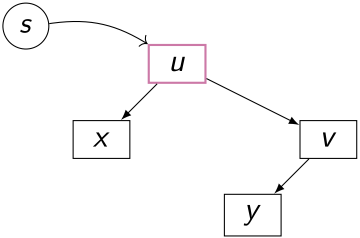
- Best way to find LCP?
- $k = |s| = 38$
- # computations: 1
- # branching: 0



$s =$ **G** G C T A G C T A C A A C G A G C T G G G C A G C A G C T A G T A G T A G G
$u =$ **G** G C T A A C T A C A A C G A G C T G G C G T T G T G A G C T A C

Sequential Longest Common Prefix

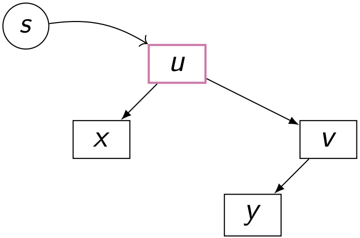
- Best way to find LCP?
- $k = |s| = 38$
- # computations: 2
- # branching: 0



$s =$ **GG**CTAGCTACAACGAGCTGGGCAGCAGCTAGTAGTAGG
$u =$ **GG**CTAACTACAACGAGCTGGCGTTGTGAGCTAC

Sequential Longest Common Prefix

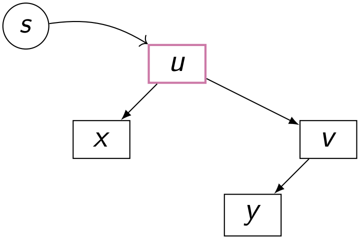
- Best way to find LCP?
- $k = |s| = 38$
- # computations: 3
- # branching: 0



$s =$ **GGC**TAGCTACAACGAGCTGGGCAGCAGCTAGTAGTAGG
$u =$ **GGC**TAACTACAACGAGCTGGCGTTGTGAGCTAC

Sequential Longest Common Prefix

- Best way to find LCP?
- $k = |s| = 38$
- # computations: 4
- # branching: 0

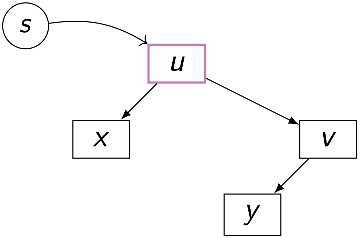


$s =$ **GGCT**AGCTACAACGAGCTGGGCAGCAGCTAGTAGTAGG
$u =$ **GGCT**AACTACAACGAGCTGGCGTTGTGAGCTAC

Sequential Longest Common Prefix

- Best way to find LCP?
- $k = |s| = 38$

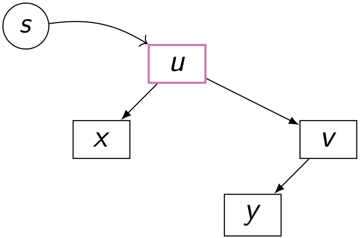
- # computations: 5
- # branching: 0



$s =$ **GGCTA**GCTACAACGAGCTGGGCAGCAGCTAGTAGTAGG
$u =$ **GGCTA**ACTACAACGAGCTGGCGTTGTGAGCTAC

Sequential Longest Common Prefix

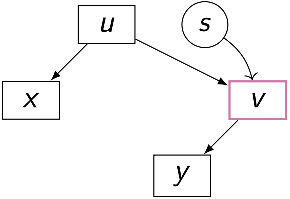
- Best way to find LCP?
- $k = |s| = 38$
- # computations: 6
- # branching: 0



$s =$ **GGCTAG**CTACAACGAGCTGGGCAGCAGCTAGTAGTAGG
$u =$ **GGCTAA**CTACAACGAGCTGGCGTTGTGAGCTAC

Sequential Longest Common Prefix

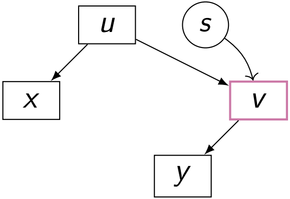
- Best way to find LCP?
- $k = |s| = 38$
- # computations: 6
- # branching: 1



$s =$ **GGCTA**GCTACAACGAGCTGGGCAGCAGCTAGTAGTAGG
$u =$ **GGCTAA**CTACAACGAGCTGGCGTTGTGAGCTAC
$v =$ **GGCTA**GCTACAACGAGCTGGTCAGTAACGAGCTGGG

Sequential Longest Common Prefix

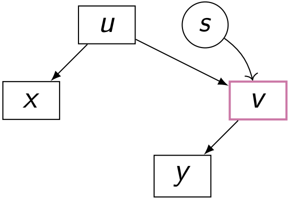
- Best way to find LCP?
- $k = |s| = 38$
- # computations: 7
- # branching: 1



$s =$ **GGCTAG**CTACAACGAGCTGGGCAGCAGCTAGTAGTAGG
$u =$ **GGCTAA**CTACAACGAGCTGGCGTTGTGAGCTAC
$v =$ **GGCTAG**CTACAACGAGCTGGTCAGTAACGAGCTGGG

Sequential Longest Common Prefix

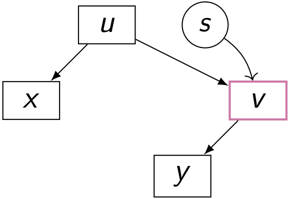
- Best way to find LCP?
- $k = |s| = 38$
- # computations: 8
- # branching: 1



$s =$ **GGCTAGC**TACAACGAGCTGGGCAGCAGCTAGTAGTAGG
$u =$ **GGCTAA**CTACAACGAGCTGGCGTTGTGAGCTAC
$v =$ **GGCTAGC**TACAACGAGCTGGTCAGTAACGAGCTGGG

Sequential Longest Common Prefix

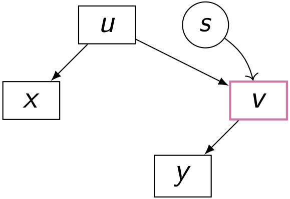
- Best way to find LCP?
- $k = |s| = 38$
- # computations: 9
- # branching: 1



$s =$ **GGCTAGCT**ACAACGAGCTGGGCAGCAGCTAGTAGTAGG
$u =$ **GGCTAA**CTACAACGAGCTGGCGTTGTGAGCTAC
$v =$ **GGCTAGCT**ACAACGAGCTGGTCAGTAACGAGCTGGG

Sequential Longest Common Prefix

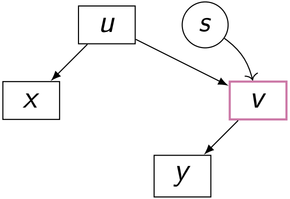
- Best way to find LCP?
- $k = |s| = 38$
- # computations: 10
- # branching: 1



$s =$ **GGCTAGCTA**CAACGAGCTGGGCAGCAGCTAGTAGTAGG
$u =$ **GGCTAA**CTACAACGAGCTGGCGTTGTGAGCTAC
$v =$ **GGCTAGCTA**CAACGAGCTGGTCAGTAACGAGCTGGG

Sequential Longest Common Prefix

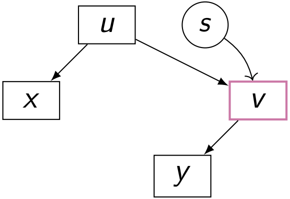
- Best way to find LCP?
- $k = |s| = 38$
- # computations: 11
- # branching: 1



$s =$ **GGCTAGCTAC**AACGAGCTGGGCAGCAGCTAGTAGTAGG
$u =$ **GGCTAA**CTACAACGAGCTGGCGTTGTGAGCTAC
$v =$ **GGCTAGCTAC**AACGAGCTGGTCAGTAACGAGCTGGG

Sequential Longest Common Prefix

- Best way to find LCP?
- $k = |s| = 38$
- # computations: 12
- # branching: 1



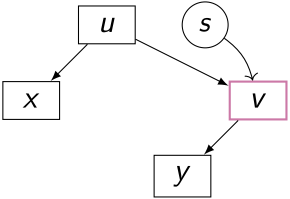
$s =$ **GGCTAGCTACA**ACGAGCTGGGCAGCAGCTAGTAGTAGG

$u =$ **GGCTAA**CTACAACGAGCTGGCGTTGTGAGCTAC

$v =$ **GGCTAGCTACA**ACGAGCTGGTCAGTAACGAGCTGGG

Sequential Longest Common Prefix

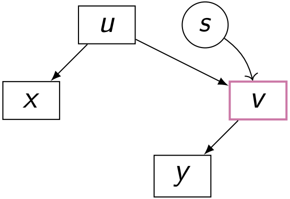
- Best way to find LCP?
- $k = |s| = 38$
- # computations: 13
- # branching: 1



$s =$ **GGCTAGCTACAA**CGAGCTGGGCAGCAGCTAGTAGTAGG
$u =$ **GGCTAA**CTACAACGAGCTGGCGTTGTGAGCTAC
$v =$ **GGCTAGCTACAA**CGAGCTGGTCAGTAACGAGCTGGG

Sequential Longest Common Prefix

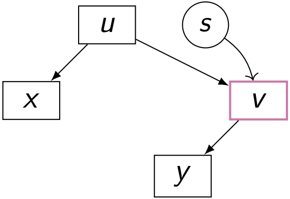
- Best way to find LCP?
- $k = |s| = 38$
- # computations: 14
- # branching: 1



$s =$ **GGCTAGCTACAAC**GAGCTGGGCAGCAGCTAGTAGTAGG
$u =$ **GGCTAA**CTACAACGAGCTGGCGTTGTGAGCTAC
$v =$ **GGCTAGCTACAAC**GAGCTGGTCAGTAACGAGCTGGG

Sequential Longest Common Prefix

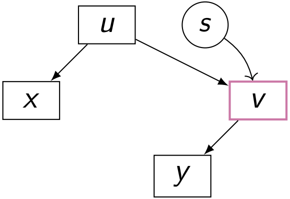
- Best way to find LCP?
- $k = |s| = 38$
- # computations: 15
- # branching: 1



$s =$ **GGCTAGCTACAACG**AGCTGGGCAGCAGCTAGTAGTAGG
$u =$ **GGCTAA**CTACAACGAGCTGGCGTTGTGAGCTAC
$v =$ **GGCTAGCTACAACG**AGCTGGTCAGTAACGAGCTGGG

Sequential Longest Common Prefix

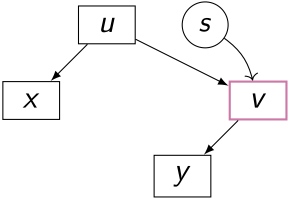
- Best way to find LCP?
- $k = |s| = 38$
- # computations: 16
- # branching: 1



$s =$ **GGCTAGCTACAACGA**GCTGGGCAGCAGCTAGTAGTAGG
$u =$ **GGCTAA**CTACAACGAGCTGGCGTTGTGAGCTAC
$v =$ **GGCTAGCTACAACGA**GCTGGTCAGTAACGAGCTGGG

Sequential Longest Common Prefix

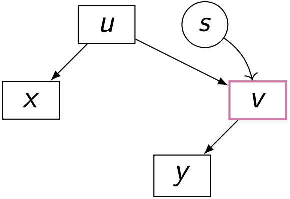
- Best way to find LCP?
- $k = |s| = 38$
- # computations: 17
- # branching: 1



$s =$ **GGCTAGCTACAACGAG**CTGGGCAGCAGCTAGTAGTAGG
$u =$ **GGCTAA**CTACAACGAGCTGGCGTTGTGAGCTAC
$v =$ **GGCTAGCTACAACGAG**CTGGTCAGTAACGAGCTGGG

Sequential Longest Common Prefix

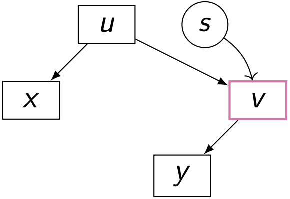
- Best way to find LCP?
- $k = |s| = 38$
- # computations: 18
- # branching: 1



$s =$ **GGCTAGCTACAACGAGC**TGGGCAGCAGCTAGTAGTAGG
$u =$ **GGCTAA**CTACAACGAGCTGGCGTTGTGAGCTAC
$v =$ **GGCTAGCTACAACGAGC**TGGTCAGTAACGAGCTGGG

Sequential Longest Common Prefix

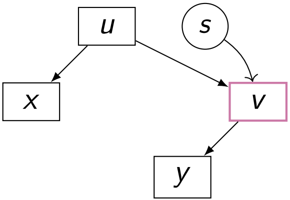
- Best way to find LCP?
- $k = |s| = 38$
- # computations: 19
- # branching: 1



$s =$ **GGCTAGCTACAACGAGCT**GGGCAGCAGCTAGTAGTAGG
$u =$ **GGCTAA**CTACAACGAGCTGGCGTTGTGAGCTAC
$v =$ **GGCTAGCTACAACGAGCT**GGTCAGTAACGAGCTGGG

Sequential Longest Common Prefix

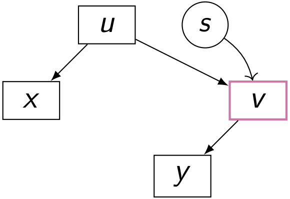
- Best way to find LCP?
- $k = |s| = 38$
- # computations: 20
- # branching: 1



$s =$ **GGCTAGCTACAACGAGCTG**GGCAGCAGCTAGTAGTAGG
$u =$ **GGCTAA**CTACAACGAGCTGGCGTTGTGAGCTAC
$v =$ **GGCTAGCTACAACGAGCTG**GTCAGTAACGAGCTGGG

Sequential Longest Common Prefix

- Best way to find LCP?
- $k = |s| = 38$
- # computations: 21
- # branching: 1

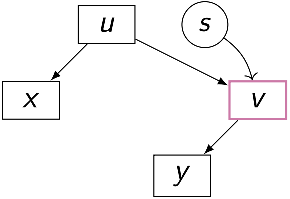


$s =$ **GGCTAGCTACAACGAGCTGG**GCAGCAGCTAGTAGTAGG
$u =$ **GGCTAA**CTACAACGAGCTGGCGTTGTGAGCTAC
$v =$ **GGCTAGCTACAACGAGCTGG**TCAGTAACGAGCTGGG

Sequential Longest Common Prefix

- Best way to find LCP?
- $k = |s| = 38$

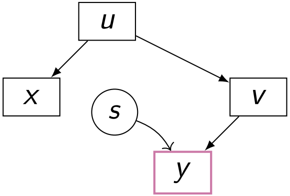
- # computations: 23
- # branching: 1



$s =$ **GGCTAGCTACAACGAGCTGGG**CAGCAGCTAGTAGTAGG
$u =$ **GGCTAA**CTACAACGAGCTGGCGTTGTGAGCTAC
$v =$ **GGCTAGCTACAACGAGCTGGT**CAGTAACGAGCTGGG

Sequential Longest Common Prefix

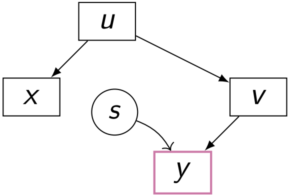
- Best way to find LCP?
- $k = |s| = 38$
- # computations: 23
- # branching: 2



$s =$ **GGCTAGCTACAACGAGCTGG**GCAGCAGCTAGTAGTAGG
$u =$ **GGCTAA**CTACAACGAGCTGGCGTTGTGAGCTAC
$v =$ **GGCTAGCTACAACGAGCTGGT**CAGTAACGAGCTGGG
$y =$ **GGCTAGCTACAACGAGCTGG**GCAGCAGCTAGTAGTAGG

Sequential Longest Common Prefix

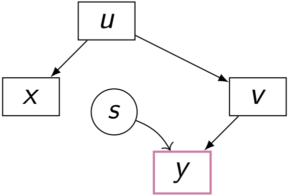
- Best way to find LCP?
- $k = |s| = 38$
- # computations: 23
- # branching: 2



$s =$ **GGCTAGCTACAACGAGCTGGG**CAGCAGCTAGTAGTAGG
$u =$ **GGCTAA**CTACAACGAGCTGGCGTTGTGAGCTAC
$v =$ **GGCTAGCTACAACGAGCTGGT**CAGTAACGAGCTGGG
$y =$ **GGCTAGCTACAACGAGCTGGG**CAGCAGCTAGTAGTAGG

Sequential Longest Common Prefix

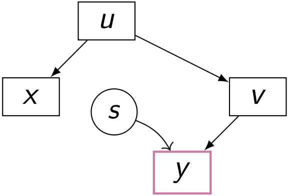
- Best way to find LCP?
- $k = |s| = 38$
- # computations: 24
- # branching: 2



$s =$ **GGCTAGCTACAACGAGCTGGGC**AGCAGCTAGTAGTAGG
$u =$ **GGCTAA**CTACAACGAGCTGGCGTTGTGAGCTAC
$v =$ **GGCTAGCTACAACGAGCTGGT**CAGTAACGAGCTGGG
$y =$ **GGCTAGCTACAACGAGCTGGGC**AGCAGCTAGTAGTAGG

Sequential Longest Common Prefix

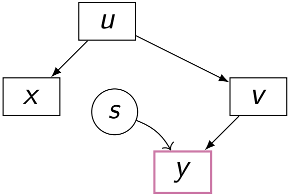
- Best way to find LCP?
- $k = |s| = 38$
- # computations: 25
- # branching: 2



$s =$ **GGCTAGCTACAACGAGCTGGGCA**GCAGCTAGTAGTAGG
$u =$ **GGCTAA**CTACAACGAGCTGGCGTTGTGAGCTAC
$v =$ **GGCTAGCTACAACGAGCTGGT**CAGTAACGAGCTGGG
$y =$ **GGCTAGCTACAACGAGCTGGGCA**GCAGCTAGTAGTAGG

Sequential Longest Common Prefix

- Best way to find LCP?
- $k = |s| = 38$
- # computations: 26
- # branching: 2



$s =$ **GGCTAGCTACAACGAGCTGGGCAG**CAGCTAGTAGTAGG

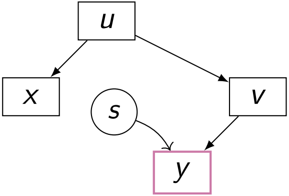
$u =$ **GGCTAA**CTACAACGAGCTGGCGTTGTGAGCTAC

$v =$ **GGCTAGCTACAACGAGCTGGT**CAGTAACGAGCTGGG

$y =$ **GGCTAGCTACAACGAGCTGGGCAG**CAGCTAGTAGTAGG

Sequential Longest Common Prefix

- Best way to find LCP?
- $k = |s| = 38$
- # computations: 27
- # branching: 2



$s =$ **GGCTAGCTACAACGAGCTGGGCAGC**AGCTAGTAGTAGG

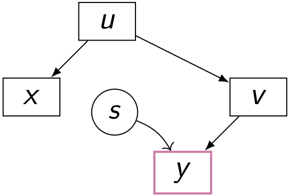
$u =$ **GGCTAA**CTACAACGAGCTGGCGTTGTGAGCTAC

$v =$ **GGCTAGCTACAACGAGCTGGT**CAGTAACGAGCTGGG

$y =$ **GGCTAGCTACAACGAGCTGGGCAGC**AGCTAGTAGTAGG

Sequential Longest Common Prefix

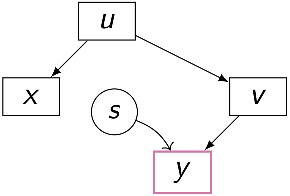
- Best way to find LCP?
- $k = |s| = 38$
- # computations: 28
- # branching: 2



$s =$ **GGCTAGCTACAACGAGCTGGGCAGCA**GCTAGTAGTAGG
$u =$ **GGCTAA**CTACAACGAGCTGGCGTTGTGAGCTAC
$v =$ **GGCTAGCTACAACGAGCTGGT**CAGTAACGAGCTGGG
$y =$ **GGCTAGCTACAACGAGCTGGGCAGCA**GCTAGTAGTAGG

Sequential Longest Common Prefix

- Best way to find LCP?
- $k = |s| = 38$
- # computations: 29
- # branching: 2



$s =$ **GGCTAGCTACAACGAGCTGGGCAGCAG**CTAGTAGTAGG

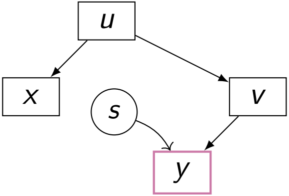
$u =$ **GGCTAA**CTACAACGAGCTGGCGTTGTGAGCTAC

$v =$ **GGCTAGCTACAACGAGCTGGT**CAGTAACGAGCTGGG

$y =$ **GGCTAGCTACAACGAGCTGGGCAGCAG**CTAGTAGTAGG

Sequential Longest Common Prefix

- Best way to find LCP?
- $k = |s| = 38$
- # computations: 30
- # branching: 2

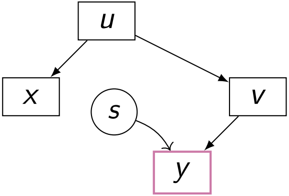


$s =$ **GGCTAGCTACAACGAGCTGGGCAGCAGC**TAGTAGTAGG
$u =$ **GGCTAA**CTACAACGAGCTGGCGTTGTGAGCTAC
$v =$ **GGCTAGCTACAACGAGCTGGT**CAGTAACGAGCTGGG
$y =$ **GGCTAGCTACAACGAGCTGGGCAGCAGC**TAGTAGTAGG

Sequential Longest Common Prefix

- Best way to find LCP?
- $k = |s| = 38$

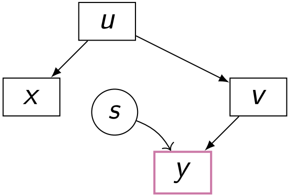
- # computations: 31
- # branching: 2



$s =$ **GGCTAGCTACAACGAGCTGGGCAGCAGCT**AGTAGTAGG
$u =$ **GGCTAA**CTACAACGAGCTGGCGTTGTGAGCTAC
$v =$ **GGCTAGCTACAACGAGCTGGT**CAGTAACGAGCTGGG
$y =$ **GGCTAGCTACAACGAGCTGGGCAGCAGCT**AGTAGTAGG

Sequential Longest Common Prefix

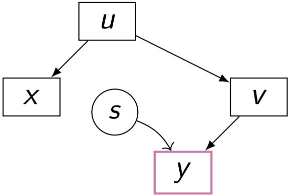
- Best way to find LCP?
- $k = |s| = 38$
- # computations: 32
- # branching: 2



$s =$ **GGCTAGCTACAACGAGCTGGGCAGCAGCTA**GTAGTAGG
$u =$ **GGCTAA**CTACAACGAGCTGGCGTTGTGAGCTAC
$v =$ **GGCTAGCTACAACGAGCTGGT**CAGTAACGAGCTGGG
$y =$ **GGCTAGCTACAACGAGCTGGGCAGCAGCTA**GTAGTAGG

Sequential Longest Common Prefix

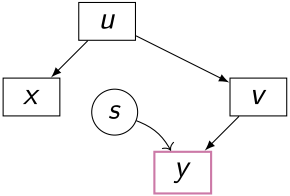
- Best way to find LCP?
- $k = |s| = 38$
- # computations: 33
- # branching: 2



$s =$ **GGCTAGCTACAACGAGCTGGGCAGCAGCTAG**TAGTAGG
$u =$ **GGCTAA**CTACAACGAGCTGGCGTTGTGAGCTAC
$v =$ **GGCTAGCTACAACGAGCTGGT**CAGTAACGAGCTGGG
$y =$ **GGCTAGCTACAACGAGCTGGGCAGCAGCTAG**TAGTAGG

Sequential Longest Common Prefix

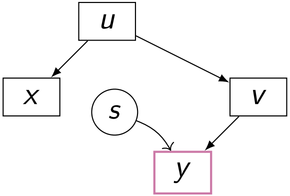
- Best way to find LCP?
- $k = |s| = 38$
- # computations: 34
- # branching: 2



$s =$ **GGCTAGCTACAACGAGCTGGGCAGCAGCTAGT**AGTAGG
$u =$ **GGCTAA**CTACAACGAGCTGGCGTTGTGAGCTAC
$v =$ **GGCTAGCTACAACGAGCTGGT**CAGTAACGAGCTGGG
$y =$ **GGCTAGCTACAACGAGCTGGGCAGCAGCTAGT**AGTAGG

Sequential Longest Common Prefix

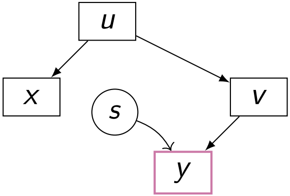
- Best way to find LCP?
- $k = |s| = 38$
- # computations: 35
- # branching: 2



$s =$ **GGCTAGCTACAACGAGCTGGGCAGCAGCTAGTA**GTAGG
$u =$ **GGCTAA**CTACAACGAGCTGGCGTTGTGAGCTAC
$v =$ **GGCTAGCTACAACGAGCTGGT**CAGTAACGAGCTGGG
$y =$ **GGCTAGCTACAACGAGCTGGGCAGCAGCTAGTA**GTAGG

Sequential Longest Common Prefix

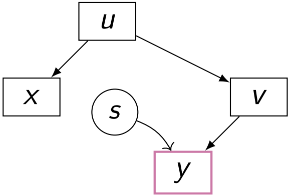
- Best way to find LCP?
- $k = |s| = 38$
- # computations: 36
- # branching: 2



$s =$ **GGCTAGCTACAACGAGCTGGGCAGCAGCTAGTAG**TAGG
$u =$ **GGCTAA**CTACAACGAGCTGGCGTTGTGAGCTAC
$v =$ **GGCTAGCTACAACGAGCTGGT**CAGTAACGAGCTGGG
$y =$ **GGCTAGCTACAACGAGCTGGGCAGCAGCTAGTAG**TAGG

Sequential Longest Common Prefix

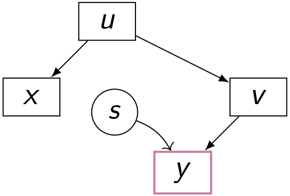
- Best way to find LCP?
- $k = |s| = 38$
- # computations: 37
- # branching: 2



$s =$ **GGCTAGCTACAACGAGCTGGGCAGCAGCTAGTAGT**AGG
$u =$ **GGCTAA**CTACAACGAGCTGGCGTTGTGAGCTAC
$v =$ **GGCTAGCTACAACGAGCTGGT**CAGTAACGAGCTGGG
$y =$ **GGCTAGCTACAACGAGCTGGGCAGCAGCTAGTAGT**AGG

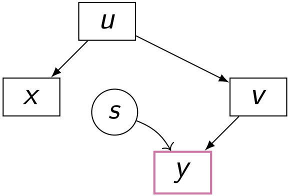
Sequential Longest Common Prefix

- Best way to find LCP?
- $k = |s| = 38$
- # computations: 38
- # branching: 2



$s =$ **GGCTAGCTACAACGAGCTGGGCAGCAGCTAGTAGTA**GG
$u =$ **GGCTAA**CTACAACGAGCTGGCGTTGTGAGCTAC
$v =$ **GGCTAGCTACAACGAGCTGGT**CAGTAACGAGCTGGG
$y =$ **GGCTAGCTACAACGAGCTGGGCAGCAGCTAGTAGTA**GG

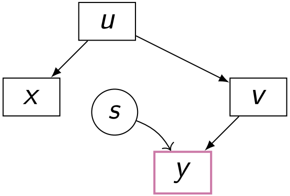
- Best way to find LCP?
- $k = |s| = 38$
- # computations: 39
- # branching: 2



$s =$ **GGCTAGCTACAACGAGCTGGGCAGCAGCTAGTAGTAG**G
$u =$ **GGCTAA**CTACAACGAGCTGGCGTTGTGAGCTAC
$v =$ **GGCTAGCTACAACGAGCTGGT**CAGTAACGAGCTGGG
$y =$ **GGCTAGCTACAACGAGCTGGGCAGCAGCTAGTAGTAG**G

Sequential Longest Common Prefix

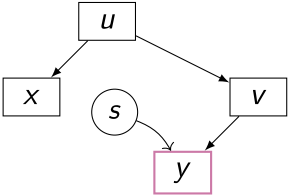
- Best way to find LCP?
- $k = |s| = 38$
- # computations: 40
- # branching: 2



$s =$ **GGCTAGCTACAACGAGCTGGGCAGCAGCTAGTAGTAGG**
$u =$ **GGCTAA**CTACAACGAGCTGGCGTTGTGAGCTAC
$v =$ **GGCTAGCTACAACGAGCTGGT**CAGTAACGAGCTGGG
$y =$ **GGCTAGCTACAACGAGCTGGGCAGCAGCTAGTAGTAGG**

Sequential Longest Common Prefix

- Best way to find LCP?
- $k = |s| = 38$
- # computations: 40
- # branching: 2
- # computations $=k+$ # branching

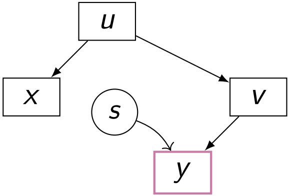


$s =$ **GGCTAGCTACAACGAGCTGGGCAGCAGCTAGTAGTAGG**
$u =$ **GGCTAA**CTACAACGAGCTGGCGTTGTGAGCTAC
$v =$ **GGCTAGCTACAACGAGCTGGT**CAGTAACGAGCTGGG
$y =$ **GGCTAGCTACAACGAGCTGGGCAGCAGCTAGTAGTAGG**

Sequential Longest Common Prefix

- Best way to find LCP?
- $k = |s| = 38$
- # computations: 40
- # branching: 2

- # computations $= k + \#$ branching
- ☑ # computations $= \mathcal{O}(k + \#$ branching$)$



$s =$ **GGCTAGCTACAACGAGCTGGGCAGCAGCTAGTAGTAGG**
$u =$ **GGCTAA**CTACAACGAGCTGGCGTTGTGAGCTAC
$v =$ **GGCTAGCTACAACGAGCTGGT**CAGTAACGAGCTGGG
$y =$ **GGCTAGCTACAACGAGCTGGGCAGCAGCTAGTAGTAGG**

Naive Parallel LCP

- Best way to find LCP?
- $k = |s| = 38$

- # computations: 0
- # branching: 0
- # steps: 0

$s =$ G G C T A G C T A C A A C G A G C T G G G C A G C A G C T A G T A G T A G G
$u =$ G G C T A A C T A C A A C G A G C T G G C G T T G T G A G C T A C

Naive Parallel LCP

- Best way to find LCP?
- $k = |s| = 38$

- # computations: 33
- # branching: 0
- # steps: 1

$s =$ GGCTAGCTACAACGAGCTGGGCAGCAGCTAGTAGTAGG
$u =$ GGCTAACTACAACGAGCTGGCGTTGTGAGCTAC

33

Naive Parallel LCP

- Best way to find LCP?
- $k = |s| = 38$

- # computations: 33
- # branching: 1
- # steps: 1

$s =$ **GGCTA**GCTACAACGAGCTGGGCAGCAGCTAGTAGTAGG
$u =$ **GGCTAACTACAACGAGCTGGCGTTGTGAGCTAC**
$v =$ **GGCTA**GCTACAACGAGCTGGTCAGTAACGAGCTGGG

Naive Parallel LCP

- Best way to find LCP?
- $k = |s| = 38$

- # computations: 64
- # branching: 1
- # steps: 2

$s =$ GGCTAGCTACAACGAGCTGGGCAGCAGCTAGTAGTAGG
$u =$ GGCTAACTACAACGAGCTGGCGTTGTGAGCTAC
$v =$ GGCTAGCTACAACGAGCTGGTCAGTAACGAGCTGGG

31

Naive Parallel LCP

- Best way to find LCP?
- $k = |s| = 38$

- # computations: 64
- # branching: 2
- # steps: 2

$s =$ **GGCTAGCTACAACGAGCTGG**GCAGCAGCTAGTAGTAGG
$u =$ **GGCTAACTACAACGAGCTGGCGTTGTGAGCTAC**
$v =$ **GGCTAGCTACAACGAGCTGGTCAGTAACGAGCTGGG**
$y =$ **GGCTAGCTACAACGAGCTGG**GCAGCAGCTAGTAGTAGG

Naive Parallel LCP

- Best way to find LCP?
- $k = |s| = 38$

- # computations: 82
- # branching: 2
- # steps: 3

$s =$ **GGCTAGCTACAACGAGCTGGGCAGCAGCTAGTAGTAGG**
$u =$ **GGCTAACTACAACGAGCTGGCGTTGTGAGCTAC**
$v =$ **GGCTAGCTACAACGAGCTGGTCAGTAACGAGCTGGG**
$y =$ **GGCTAGCTACAACGAGCTGGGCAGCAGCTAGTAGTAGG**

18

Naive Parallel LCP

- Best way to find LCP?
- $k = |s| = 38$

- # computations: 82
- # branching: 2
- # steps: 3

☑ # steps = # branching + 1 = # comparisons

$s =$ GGCTAGCTACAACGAGCTGGGCAGCAGCTAGTAGTAGG
$u =$ GGCTAACTACAACGAGCTGGCGTTGTGAGCTAC
$v =$ GGCTAGCTACAACGAGCTGGTCAGTAACGAGCTGGG
$y =$ GGCTAGCTACAACGAGCTGGGCAGCAGCTAGTAGTAGG

18

Naive Parallel LCP

- Best way to find LCP?
- $k = |s| = 38$

- # computations: 82
- # branching: 2
- # steps: 3

☑ # steps = # branching + 1 = # comparisons

☐ # computations?

$s =$ GGCTAGCTACAACGAGCTGGGCAGCAGCTAGTAGTAGG
$u =$ GGCTAACTACAACGAGCTGGCGTTGTGAGCTAC
$v =$ GGCTAGCTACAACGAGCTGGTCAGTAACGAGCTGGG
$y =$ GGCTAGCTACAACGAGCTGGGCAGCAGCTAGTAGTAGG

18

Naive Parallel LCP

- Best way to find LCP?
- $k = |s| = 38$

- # computations: 82
- # branching: 2
- # steps: 3

☑ # steps = # branching + 1 = # comparisons

☒ # computations? – $\mathcal{O}(k \times \#\ \text{comparisons}) = \mathcal{O}(k \times \#\ \text{branching})$

$s =$ GGCTAGCTACAACGAGCTGGGCAGCAGCTAGTAGTAGG

$u =$ GGCTAACTACAACGAGCTGGCGTTGTGAGCTAC

$v =$ GGCTAGCTACAACGAGCTGGTCAGTAACGAGCTGGG

$y =$ GGCTAGCTACAACGAGCTGG$\underbrace{\text{GCAGCAGCTAGTAGTAGG}}_{18}$

Parallel LCP

- Best way to find LCP?
- $k = |s| = 38$

- # computations: 0
- # branching: 0
- # steps: 0

- Only compare $\frac{k}{\#\text{ comparisons}} = 13$ characters?

$s =$ G G C T A G C T A C A A C G A G C T G G G C A G C A G C T A G T A G T A G G
$u =$ G G C T A A C T A C A A C G A G C T G G C G T T G T G A G C T A C

Parallel LCP

- Best way to find LCP?
- $k = |s| = 38$

- # computations: 13
- # branching: 0
- # steps: 1

- Only compare $\frac{k}{\#\ \text{comparisons}} = 13$ characters?

$s =$ **GGCTAGCTACAAC**GAGCTGGGCAGCAGCTAGTAGTAGG
$u =$ **GGCTAACTACAAC**GAGCTGGCGTTGTGAGCTAC

$\underbrace{}_{13}$

Parallel LCP

- Best way to find LCP?
- $k = |s| = 38$

- # computations: 13
- # branching: 1
- # steps: 1

- Only compare $\frac{k}{\#\ \text{comparisons}} = 13$ characters?

$s =$ **GGCTA**GCTACAACGAGCTGGGCAGCAGCTAGTAGTAGG
$u =$ **GGCTAACTACAAC**GAGCTGGCGTTGTGAGCTAC
$v =$ **GGCTA**GCTACAACGAGCTGGTCAGTAACGAGCTGGG

Parallel LCP

- Best way to find LCP?
- $k = |s| = 38$

- # computations: 26
- # branching: 1
- # steps: 2

- Only compare $\frac{k}{\#\ \text{comparisons}} = 13$ characters?

$s =$ **GGCTAGCTACAACGAGCT**GGGCAGCAGCTAGTAGTAGG
$u =$ **GGCTAACTACAAC**GAGCTGGCGTTGTGAGCTAC
$v =$ **GGCTAGCTACAACGAGCT**GGTCAGTAACGAGCTGGG

13

Parallel LCP

- Best way to find LCP?
- $k = |s| = 38$

- # computations: 39
- # branching: 1
- # steps: 3

- Only compare $\frac{k}{\#\ \text{comparisons}} = 13$ characters?

$s =$ GGCTAGCTACAACGAGCTGGGCAGCAGCTAGTAGTAGG
$u =$ GGCTAACTACAACGAGCTGGCGTTGTGAGCTAC
$v =$ GGCTAGCTACAACGAGCTGGTCAGTAACGAGCTGGG

13

Parallel LCP

- Best way to find LCP?
- $k = |s| = 38$

- # computations: 39
- # branching: 2
- # steps: 3

- Only compare $\frac{k}{\#\ \text{comparisons}} = 13$ characters?

$s =$ **GGCTAGCTACAACGAGCTGG**GCAGCAGCTAGTAGTAGG

$u =$ **GGCTAACTACAAC**GAGCTGGCGTTGTGAGCTAC

$v =$ **GGCTAGCTACAACGAGCTGGTCAGTAACGAG**CTGGG

$y =$ **GGCTAGCTACAACGAGCTGG**GCAGCAGCTAGTAGTAGG

Parallel LCP

- Best way to find LCP?
- $k = |s| = 38$

- # computations: 52
- # branching: 2
- # steps: 4

- Only compare $\frac{k}{\#\ \text{comparisons}} = 13$ characters?

$s =$ GGCTAGCTACAACGAGCTGGGCAGCAGCTAGTAGTAGG
$u =$ GGCTAACTACAACGAGCTGGCGTTGTGAGCTAC
$v =$ GGCTAGCTACAACGAGCTGGTCAGTAACGAGCTGGG
$y =$ GGCTAGCTACAACGAGCTGGGCAGCAGCTAGTAGTAGG

13

Parallel LCP

- Best way to find LCP?
- $k = |s| = 38$

- # computations: 57
- # branching: 2
- # steps: 5

- Only compare $\frac{k}{\#\text{ comparisons}} = 13$ characters?

$s =$ GGCTAGCTACAACGAGCTGGGCAGCAGCTAGTAGTAGG
$u =$ GGCTAACTACAACGAGCTGGCGTTGTGAGCTAC
$v =$ GGCTAGCTACAACGAGCTGGTCAGTAACGAGCTGGG
$y =$ GGCTAGCTACAACGAGCTGGGCAGCAGCTAGTAGTAGG

5

Parallel LCP

- Best way to find LCP?
- $k = |s| = 38$
- # computations: 57
- # branching: 2
- # steps: 5
- Only compare $\frac{k}{\#\ \text{comparisons}} = 13$ characters?

☐ # steps?

$s =$ **GGCTAGCTACAACGAGCTGGGCAGCAGCTAGTAGTAGG**
$u =$ **GGCTAACTACAAC**GAGCTGGCGTTGTGAGCTAC
$v =$ **GGCTAGCTACAACGAGCTGGTCAGTAACGAG**CTGGG
$y =$ **GGCTAGCTACAACGAGCTGGGCAGCAGCTAGTAGTAGG**

5

Parallel LCP

- Best way to find LCP?
- $k = |s| = 38$
- # computations: 57
- # branching: 2
- # steps: 5
- Only compare $\frac{k}{\#\text{ comparisons}} = 13$ characters?

☑ # steps? $2 \times \#$ comparisons $= \mathcal{O}(\#\text{ branching})$

$s =$ GGCTAGCTACAACGAGCTGGGCAGCAGCTAGTAGTAGG
$u =$ GGCTAACTACAACGAGCTGGCGTTGTGAGCTAC
$v =$ GGCTAGCTACAACGAGCTGGTCAGTAACGAGCTGGG
$y =$ GGCTAGCTACAACGAGCTGGGCAGCAGCTAGTAGTAGG
5

Parallel LCP

- Best way to find LCP?
- $k = |s| = 38$

- # computations: 57
- # branching: 2
- # steps: 5

- Only compare $\frac{k}{\#\ \text{comparisons}} = 13$ characters?

☑ # steps? $2 \times \#$ comparisons $= \mathcal{O}(\#\ \text{branching})$
☐ # computations?

$s =$ GGCTAGCTACAACGAGCTGGGCAGCAGCTAGTAGTAGG
$u =$ GGCTAACTACAACGAGCTGGCGTTGTGAGCTAC
$v =$ GGCTAGCTACAACGAGCTGGTCAGTAACGAGCTGGG
$y =$ GGCTAGCTACAACGAGCTGGGCAGCAGCTAGTAGTAGG

5

Parallel LCP

- Best way to find LCP?
- $k = |s| = 38$

- # computations: 57
- # branching: 2
- # steps: 5

- Only compare $\frac{k}{\#\ \text{comparisons}} = 13$ characters?

☑ # steps? $2 \times \#$ comparisons $= \mathcal{O}(\#\ \text{branching})$
☑ # computations? $\frac{k}{\#\ \text{comparisons}} \times \#$ steps $= \mathcal{O}(k)$

$s =$ GGCTAGCTACAACGAGCTGGGCAGCAGCTAGTAGTAGG
$u =$ GGCTAACTACAACGAGCTGGCGTTGTGAGCTAC
$v =$ GGCTAGCTACAACGAGCTGGTCAGTAACGAGCTGGG
$y =$ GGCTAGCTACAACGAGCTGGGCAGCAGCTAGTAGTAGG

5

Parallel LCP

- Best way to find LCP?
- $k = |s| = 38$

- # computations: 57
- # branching: 2
- # steps: 5

- Only compare $\frac{k}{\#\text{ comparisons}} = 13$ characters?

☑ # steps? $2 \times \#\text{ comparisons} = \mathcal{O}(\#\text{ branching})$
☑ # computations? $\frac{k}{\#\text{ comparisons}} \times \#\text{ steps} = \mathcal{O}(k)$

- What if LCP $\ll k$?

$s =$ **GGCTAGCTACAACGAGCTGGGCAGCAGCTAGTAGTAGG**
$u =$ **GGCTAACTACAAC**GAGCTGGCGTTGTGAGCTAC
$v =$ **GGCTAGCTACAACGAGCTGGTCAGTAACGAG**CTGGG
$y =$ **GGCTAGCTACAACGAGCTGGGCAGCAGCTAGTAGTAGG**

$\underbrace{}_{5}$

Parallel LCP-Aware LCP

- Best way to find LCP?
- $k = |s| = 38$
- Should be *LCP-aware*

- # computations: 0
- # branching: 0
- # steps: 0

- Double and halve # compared characters

$s =$ G G C T A G C T A C A A C G A G C T G G G C A G C A G C T A G T A G T A G G
$u =$ G G C T A A C T A C A A C G A G C T G G C G T T G T G A G C T A C

Parallel LCP-Aware LCP

- Best way to find LCP?
- $k = |s| = 38$
- Should be *LCP-aware*

- # computations: 1
- # branching: 0
- # steps: 1

- Double and halve # compared characters

$s =$ **G** G C T A G C T A C A A C G A G C T G G G C A G C A G C T A G T A G T A G G
$u =$ **G** G C T A A C T A C A A C G A G C T G G C G T T G T G A G C T A C
1

Parallel LCP-Aware LCP

- Best way to find LCP?
- $k = |s| = 38$
- Should be *LCP-aware*

- # computations: 3
- # branching: 0
- # steps: 2

- Double and halve # compared characters

$s =$ **GGC**TAGCTACAACGAGCTGGGCAGCAGCTAGTAGTAGG
$u =$ **GGC**TAACTACAACGAGCTGGCGTTGTGAGCTAC
2

Parallel LCP-Aware LCP

- Best way to find LCP?
- $k = |s| = 38$
- Should be *LCP-aware*

- # computations: 7
- # branching: 1
- # steps: 3

- Double and halve # compared characters

$s =$ **GGCTAGC**TACAACGAGCTGGGCAGCAGCTAGTAGTAGG
$u =$ **GGCTAAC**TACAACGAGCTGGCGTTGTGAGCTAC

4

Parallel LCP-Aware LCP

- Best way to find LCP?
- $k = |s| = 38$
- Should be *LCP-aware*

- # computations: 7
- # branching: 1
- # steps: 3

- Double and halve # compared characters

$s =$ **GGCTA**GCTACAACGAGCTGGGCAGCAGCTAGTAGTAGG
$u =$ **GGCTAAC**TACAACGAGCTGGCGTTGTGAGCTAC
$v =$ **GGCTA**GCTACAACGAGCTGGTCAGTAACGAGCTGGG

Parallel LCP-Aware LCP

- Best way to find LCP?
- $k = |s| = 38$
- Should be *LCP-aware*

- # computations: 9
- # branching: 1
- # steps: 4

- Double and halve # compared characters

$s =$ **GGCTAGC**TACAACGAGCTGGGCAGCAGCTAGTAGTAGG
$u =$ **GGCTAAC**TACAACGAGCTGGCGTTGTGAGCTAC
$v =$ **GGCTAGC**TACAACGAGCTGGTCAGTAACGAGCTGGG

2

Parallel LCP-Aware LCP

- Best way to find LCP?
- $k = |s| = 38$
- Should be *LCP-aware*

- # computations: 13
- # branching: 1
- # steps: 5

- Double and halve # compared characters

$s =$ **GGCTAGCTACA**ACGAGCTGGGCAGCAGCTAGTAGTAGG
$u =$ **GGCTAAC**TACAACGAGCTGGCGTTGTGAGCTAC
$v =$ **GGCTAGC$\underbrace{\text{TACA}}_{4}$**ACGAGCTGGTCAGTAACGAGCTGGG

Parallel LCP-Aware LCP

- Best way to find LCP?
- $k = |s| = 38$
- Should be *LCP-aware*

- # computations: 21
- # branching: 1
- # steps: 6

- Double and halve # compared characters

$s =$ **GGCTAGCTACAACGAGCTG**GGCAGCAGCTAGTAGTAGG
$u =$ **GGCTAAC**TACAACGAGCTGGCGTTGTGAGCTAC
$v =$ **GGCTAGCTACAACGAGCTG**GTCAGTAACGAGCTGGG

8

Parallel LCP-Aware LCP

- Best way to find LCP?
- $k = |s| = 38$
- Should be *LCP-aware*

- # computations: 37
- # branching: 2
- # steps: 7

- Double and halve # compared characters

$s =$ **GGCTAGCTACAACGAGCTGGGCAGCAGCTAG**TAGTAGG
$u =$ **GGCTAAC**TACAACGAGCTGGCGTTGTGAGCTAC
$v =$ **GGCTAGCTACAACGAGCTGGTCAGTAACGAG**CTGGG

$\underbrace{\hspace{4em}}_{16}$

Parallel LCP-Aware LCP

- Best way to find LCP?
- $k = |s| = 38$
- Should be *LCP-aware*

- # computations: 37
- # branching: 2
- # steps: 7

- Double and halve # compared characters

$s =$ **GGCTAGCTACAACGAGCTGG**GCAGCAGCTAGTAGTAGG
$u =$ **GGCTAAC**TACAACGAGCTGGCGTTGTGAGCTAC
$v =$ **GGCTAGCTACAACGAGCTGGTCAGTAACGAG**CTGGG
$y =$ **GGCTAGCTACAACGAGCTGG**GCAGCAGCTAGTAGTAGG

Parallel LCP-Aware LCP

- Best way to find LCP?
- $k = |s| = 38$
- Should be *LCP-aware*

- # computations: 45
- # branching: 2
- # steps: 8

- Double and halve # compared characters

$s =$ **GGCTAGCTACAACGAGCTGGGCAGCAGC**TAGTAGTAGG
$u =$ **GGCTAAC**TACAACGAGCTGGCGTTGTGAGCTAC
$v =$ **GGCTAGCTACAACGAGCTGGTCAGTAACGAG**CTGGG
$y =$ **GGCTAGCTACAACGAGCTGGGCAGCAGC**TAGTAGTAGG

8

Parallel LCP-Aware LCP

- Best way to find LCP?
- $k = |s| = 38$
- Should be *LCP-aware*

- # computations: 55
- # branching: 3
- # steps: 9

- Double and halve # compared characters

$s =$ GGCTAGCTACAACGAGCTGGGCAGCAGCTAGTAGTAGG
$u =$ GGCTAACTACAACGAGCTGGCGTTGTGAGCTAC
$v =$ GGCTAGCTACAACGAGCTGGTCAGTAACGAGCTGGG
$y =$ GGCTAGCTACAACGAGCTGGGCAGCAGCTAGTAGTAGG

10

Parallel LCP-Aware LCP

- Best way to find LCP?
- $k = |s| = 38$
- Should be *LCP-aware*

- # computations: 55
- # branching: 3
- # steps: 9

- Double and halve # compared characters

☐ # steps?

$s =$ **GGCTAGCTACAACGAGCTGGGCAGCAGCTAGTAGTAGG**
$u =$ **GGCTAAC**TACAACGAGCTGGCGTTGTGAGCTAC
$v =$ **GGCTAGCTACAACGAGCTGGTCAGTAACGAG**CTGGG
$y =$ **GGCTAGCTACAACGAGCTGGGCAGCAGCTAGTAGTAGG**

Parallel LCP-Aware LCP

- Best way to find LCP?
- $k = |s| = 38$
- Should be *LCP-aware*

- # computations: 55
- # branching: 3
- # steps: 9

- Double and halve # compared characters

☑ # steps? $\log \ell + 2 \times \#$ branching $= \log \ell + \mathcal{O}(\#$ branching$)$

$s =$ **GGCTAGCTACAACGAGCTGGGCAGCAGCTAGTAGTAGG**
$u =$ **GGCTAAC**TACAACGAGCTGGCGTTGTGAGCTAC
$v =$ **GGCTAGCTACAACGAGCTGGTCAGTAACGAG**CTGGG
$y =$ **GGCTAGCTACAACGAGCTGGGCAGCAGCTAGTAGTAGG**

Parallel LCP-Aware LCP

- Best way to find LCP?
- $k = |s| = 38$
- Should be *LCP-aware*

- \# computations: 55
- \# branching: 3
- \# steps: 9

- Double and halve # compared characters

☑ # steps? $\log \ell + 2 \times \#$ branching $= \log \ell + \mathcal{O}(\#$ branching$)$

☐ # computations?

$s =$ **GGCTAGCTACAACGAGCTGGGCAGCAGCTAGTAGTAGG**

$u =$ **GGCTAAC**TACAACGAGCTGGCGTTGTGAGCTAC

$v =$ **GGCTAGCTACAACGAGCTGGTCAGTAACGAG**CTGGG

$y =$ **GGCTAGCTACAACGAGCTGGGCAGCAGCTAGTAGTAGG**

Parallel LCP-Aware LCP

- Best way to find LCP?
- $k = |s| = 38$
- Should be *LCP-aware*

- # computations: 55
- # branching: 3
- # steps: 9

- Double and halve # compared characters

☑ # steps? $\log \ell + 2 \times \#$ branching $= \log \ell + \mathcal{O}(\#$ branching$)$

☑ # computations? $2\ell + \#$ branching $= \mathcal{O}(\ell + \#$ branching$)$

$s =$ **GGCTAGCTACAACGAGCTGGGCAGCAGCTAGTAGTAGG**
$u =$ **GGCTAAC**TACAACGAGCTGGCGTTGTGAGCTAC
$v =$ **GGCTAGCTACAACGAGCTGGTCAGTAACGAG**CTGGG
$y =$ **GGCTAGCTACAACGAGCTGGGCAGCAGCTAGTAGTAGG**

LCP Overview

- Subsequent LCP calculations

LCP Overview

- Subsequent LCP calculations

- **Sequential:**
  - ☑ # computations (work): $\mathcal{O}(\ell + \#\ \text{branching})$
  - ☒ # steps (time): $\mathcal{O}(\ell + \#\ \text{branching})$

LCP Overview

- Subsequent LCP calculations
- **Sequential:**
  - ☑ # computations (work): $\mathcal{O}(\ell + \#\ \text{branching})$
  - ☒ # steps (time): $\mathcal{O}(\ell + \#\ \text{branching})$
- **Parallel:**
  - ? # computations (work): $\mathcal{O}(k + \#\ \text{branching})$
  - ☑ # steps (time): $\mathcal{O}(\#\ \text{branching})$

LCP Overview

- Subsequent LCP calculations

- **Sequential:**
  - ☑ $\#$ computations (work): $\mathcal{O}(\ell + \#\text{ branching})$
  - ☒ $\#$ steps (time): $\mathcal{O}(\ell + \#\text{ branching})$

- **Parallel:**
  - ? $\#$ computations (work): $\mathcal{O}(k + \#\text{ branching})$
  - ☑ $\#$ steps (time): $\mathcal{O}(\#\text{ branching})$

- **LCP-aware Parallel:**
  - ☑ $\#$ computations (work): $\mathcal{O}(\ell + \#\text{ branching})$
  - ? $\#$ steps (time): $\log \ell + \mathcal{O}(\#\text{ branching})$

LCP Overview

- Subsequent LCP calculations

- **Sequential:**
  - ☑ # computations (work): $\mathcal{O}(\ell + \#\ \text{branching})$
  - ☒ # steps (time): $\mathcal{O}(\ell + \#\ \text{branching})$

- **Parallel:**
  - ? # computations (work): $\mathcal{O}(k + \#\ \text{branching})$
  - ☑ # steps (time): $\mathcal{O}(\#\ \text{branching})$

- **LCP-aware Parallel:**
  - ☑ # computations (work): $\mathcal{O}(\ell + \#\ \text{branching})$
  - ? # steps (time): $\log \ell + \mathcal{O}(\#\ \text{branching})$

- Zip-trie: $\mathcal{O}(\log n)$ branching
- Parallel String B-Tree: $\mathcal{O}(\log_B n) = \mathcal{O}(\frac{\log n}{\log \log n})$ branching

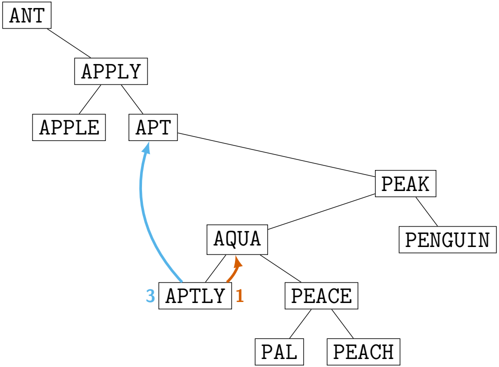
LCP Requirement

- Requirement: LCP length *never* decreases
- How?

$s =$ GGCTAGCTACAACGAGCTGGGCAGCAGCTAGTAGTAGG
$u =$ **GGCTAA**CTACAACGAGCTGGCGTTGTGAGCTAC
$v =$ **GGCTAGCTACAACGAGCTGGT**CAGTAACGAGCTGGG
$y =$ **GGCTAGCTACAACGAGCTGGGCAGCAGCTAGTAGTAGG**

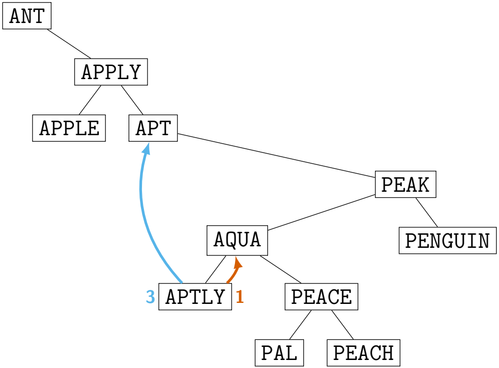
LCP Requirement

- Requirement: LCP length *never* decreases
- How? – LCP length metadata & pointers



LCP Requirement

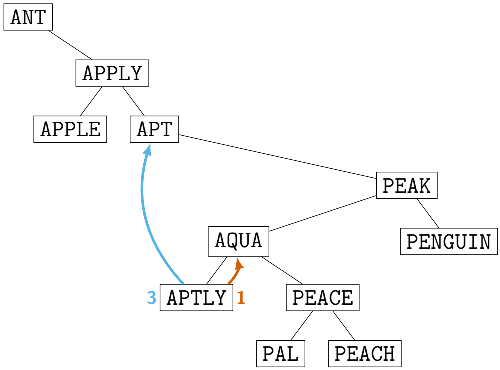
- Requirement: LCP length *never* decreases
- How? – LCP length metadata & pointers
- Space usage?



LCP Requirement

- Requirement: LCP length *never* decreases

- How? – LCP length metadata & pointers
- Space usage? – $\mathcal{O}(\log \max k) +$ pointers

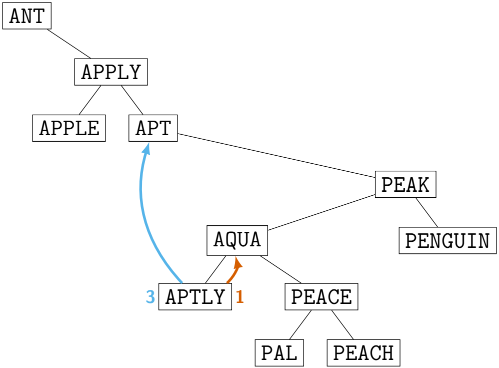


LCP Requirement

- Requirement: LCP length *never* decreases

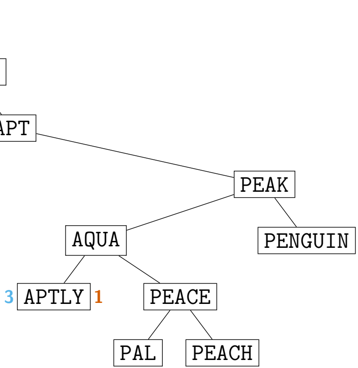
- How? – LCP length metadata & pointers
- Space usage? – $\mathcal{O}(\log \max k)$ + pointers

- Zip trie: $\log\log$ bits of metadata



LCP Requirement

- Requirement: LCP length *never* decreases
- How? – LCP length metadata ~~& pointers~~
- Space usage? – $\mathcal{O}(\log \max k)$ ~~+ pointers~~
- Zip trie: $\log \log$ bits of metadata



LCP Problem

- Zip tries require LCP length metadata
- Max LCP length is max $k \leftrightarrow \log(\max k)$ bits

LCP Problem

- Zip tries require LCP length metadata
- Max LCP length is $\max k \leftrightarrow \log(\max k)$ bits

- **Goal:** Use only $\mathcal{O}(\log\log(\max k))$ bits of metadata

LCP Problem

- Zip tries require LCP length metadata
- Max LCP length is max $k \leftrightarrow \log(\max k)$ bits

- **Goal:** Use only $\mathcal{O}(\log\log(\max k))$ bits of metadata
- We can only represents $\mathcal{O}(\log(\max k))$ values!

LCP Problem

- Zip tries require LCP length metadata
- Max LCP length is max $k \leftrightarrow \log(\max k)$ bits

- **Goal:** Use only $\mathcal{O}(\log\log(\max k))$ bits of metadata
- We can only represents $\mathcal{O}(\log(\max k))$ values!

- Which values to pick?

LCP Problem

- Zip tries require LCP length metadata
- Max LCP length is max $k \leftrightarrow \log(\max k)$ bits

- **Goal:** Use only $\mathcal{O}(\log\log(\max k))$ bits of metadata
- We can only represents $\mathcal{O}(\log(\max k))$ values!

- Which values to pick? – see paper!

$s =$ G G C T A G C T A C A A C G A G C T G G C A G C A G C T A G T A G T A G G
$u =$ G G C T A A C T A C A A C G A G C T G G C G T T G T G A G C T A C

Results

Lemma

A data structure, $\mathcal{D}_S$, spending $\mathcal{O}(\ell + A(n))$ time on comparisons using exact LCP lengths will spend at most $\mathcal{O}(\ell)$ additional time using approximate LCP lengths.

Theorem

Let ℓ be the length of the longest common prefix between a string key x of length k, and the stored keys in a parallel zip-trie, T. T can perform prefix search, predecessor/successor queries, and update operations on x in $\mathcal{O}(\log n)$ span and $\mathcal{O}(\frac{k}{\alpha} + \log n)$ work under the practical PRAM model, or alternatively in $\mathcal{O}(\log \frac{\ell}{\alpha} + \log n)$ span and $\mathcal{O}(\frac{\ell}{\alpha} + \log n)$ work.

- `https://github.com/ofekih/ZipAndSkipTries`

Results – Parallel String B-Tree

Theorem

By setting $B = \log n$, a parallel string B-tree can perform prefix search in $\mathcal{O}(\frac{\log n}{\log \log n})$ span and $\mathcal{O}(\frac{k}{\alpha} + \log^2 n)$ work in the practical CRCW PRAM model. Operations that return m keys can be done in the same span and in $\mathcal{O}(m)$ additional work.

Theorem

A parallel string B-tree can perform prefix search in $\mathcal{O}(\log_B n)$ I/O span and $\mathcal{O}(\frac{k}{\alpha B} + \log_B n)$ I/O work in the practical CRCW PEM model. Operations that return m keys can be done in the same span and in $\mathcal{O}(m/B)$ additional I/O work.

Open Problems

- Concurrent updates
- Compression techniques, e.g., DNA-specific
- Update operations in the string B-tree
- Reduce gap between RAM and PRAM work for string B-tree

References I

D. Belazzougui, P. Boldi, and S. Vigna, *Dynamic z-fast tries*, in String Processing and Information Retrieval, E. Chavez and S. Lonardi, eds., Berlin, Heidelberg, 2010, Springer Berlin Heidelberg, pp. 159–172.

P. Crescenzi, R. Grossi, and G. F. Italiano, *Search data structures for skewed strings*, in International Workshop on Experimental and Efficient Algorithms, Springer, 2003, pp. 81–96.

R. De La Briandais, *File searching using variable length keys*, in Papers Presented at the the March 3-5, 1959, Western Joint Computer Conference, IRE-AIEE-ACM '59 (Western), New York, NY, USA, Mar. 1959, Association for Computing Machinery, pp. 295–298.

P. Ferragina and R. Grossi, *The string b-tree: A new data structure for string search in external memory and its applications*, Journal of the ACM (JACM), 46 (1999), pp. 236–280.

References II

- E. Fredkin, *Trie memory*, sep 1960.
- O. Gila, M. T. Goodrich, and R. E. Tarjan, *Zip-zip Trees: Making Zip Trees More Balanced, Biased, Compact, or Persistent*, May 2024.
- R. Grossi and G. F. Italiano, *Efficient techniques for maintaining multidimensional keys in linked data structures*, in Automata, Languages and Programming: 26th International Colloquium, ICALP'99 Prague, Czech Republic, July 11–15, 1999 Proceedings, Springer, 2002, pp. 372–381.
- D. R. Morrison, *Patricia—practical algorithm to retrieve information coded in alphanumeric*, J. ACM, 15 (1968), p. 514–534.
- W. Pugh, *Skip lists: A probabilistic alternative to balanced trees*, Commun. ACM, 33 (1990), pp. 668–676.

References III

T. Takagi, S. Inenaga, K. Sadakane, and H. Arimura, *Packed compact tries: A fast and efficient data structure for online string processing*, IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences, E100.A (2017), pp. 1785–1793.

R. E. Tarjan, C. C. Levy, and S. Timmel, *Zip Trees*, ACM Trans. Algorithms, 17 (2021), pp. 1–12.