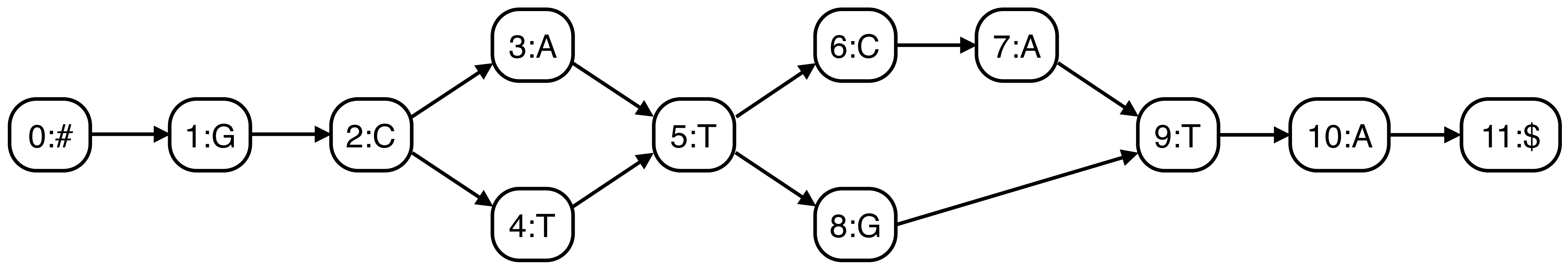
Indexing Graphs for Path Queries

Jouni Sirén
Wellcome Trust Sanger Institute



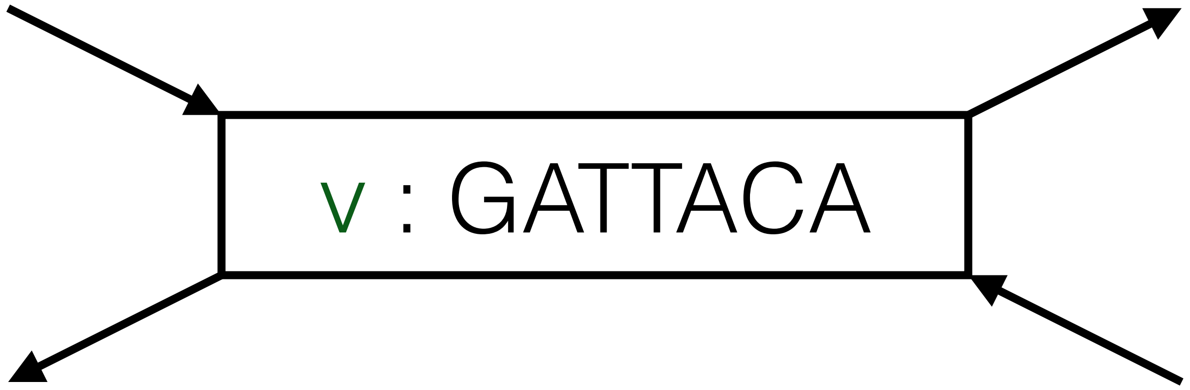
- Graphs with paths labeled by sequences are a natural way of representing genetic variation.

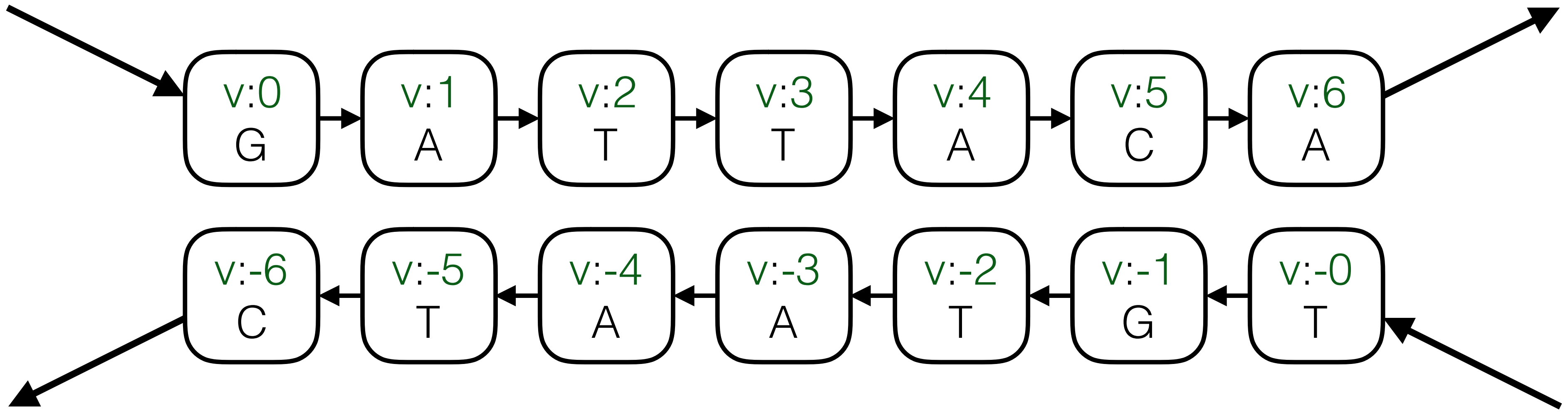
- Reference genomes could eventually become such graphs.

- The variation graph toolkit vg (Erik Garrison et al, https://github.com/vgteam/vg) is a community effort to develop tools for working with such graphs.

- This talk is about GCSA2, the path index used in vg.

Variation graphs

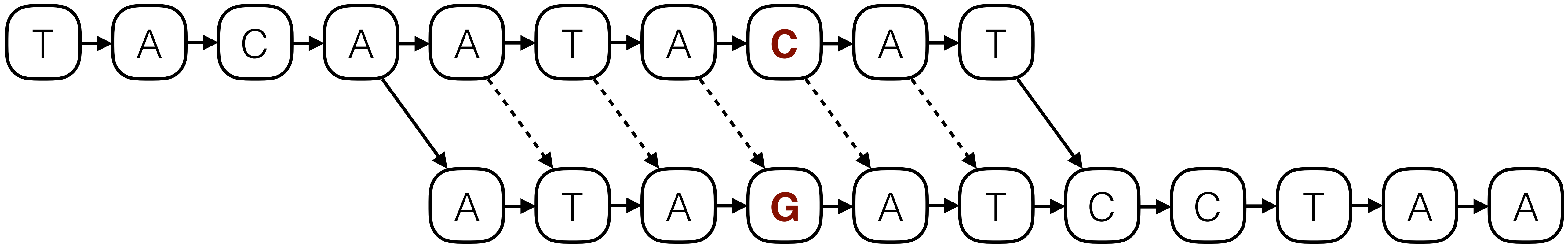




We transform the original graph into a simple directed graph while maintaining a mapping to positions in the original graph.

Assembly graphs

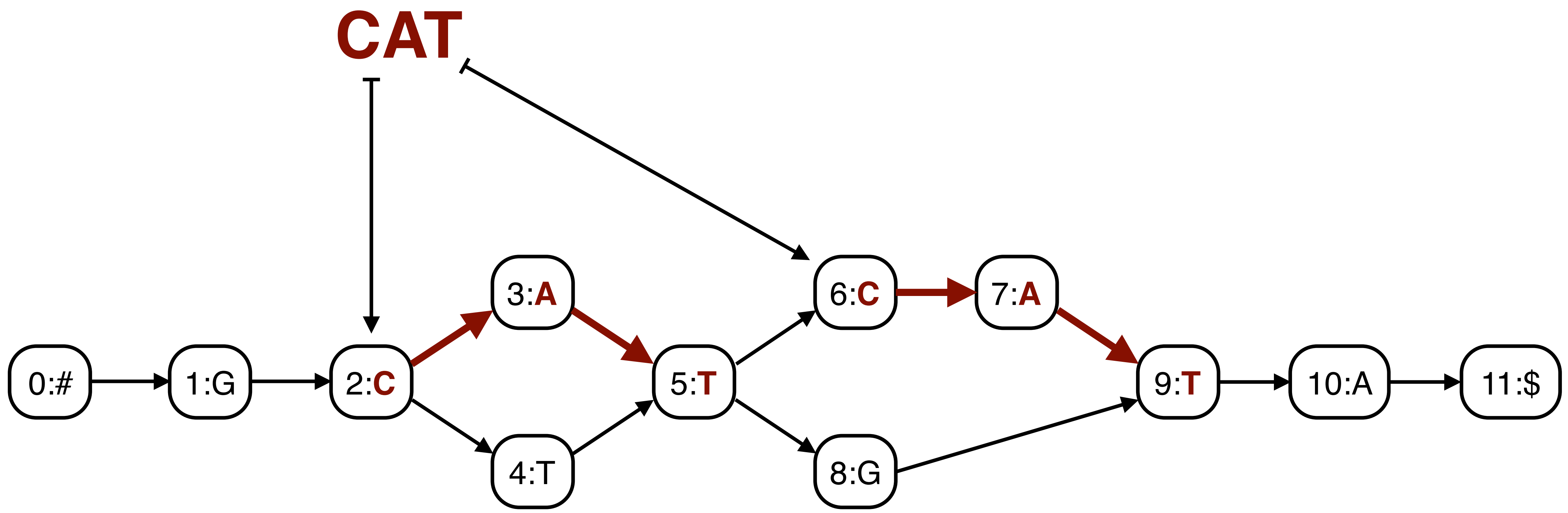
```
TACAATACAT
     ATAGATCCTAA
```



Indexing assembly graphs could also be useful.

Path Indexes

Path indexes



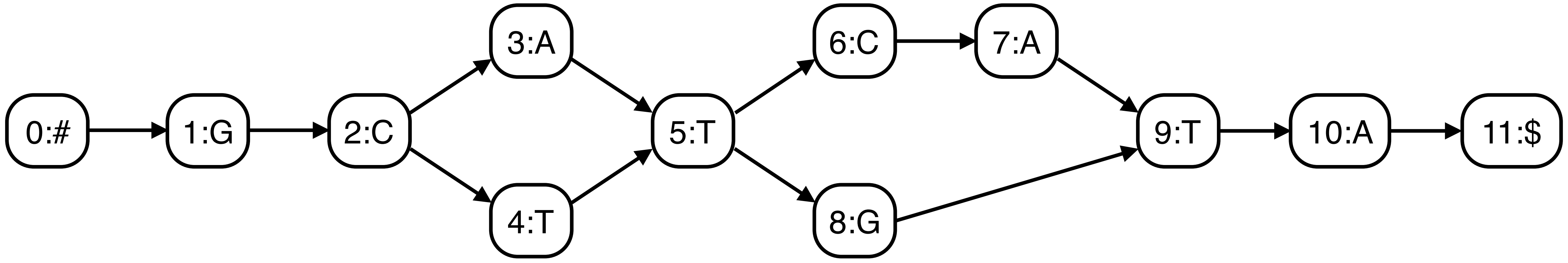
Path indexes are a central tool for working with variation graphs. They are text indexes for the path labels in a graph. The index finds (the start nodes of) the paths labeled by the query string.

Path indexes

- The number of kmers in a graph increases exponentially with k.
- k should be large enough to map perfectly matching short reads in one piece.
- In one human variation graph, the number of kmers is $1.031^k \cdot 2.348$ billion, or 116 billion for $k = 128$.
- The design of a path index is a trade-off between index size, query performance, maximum query length, and ignoring complex regions of the graph.

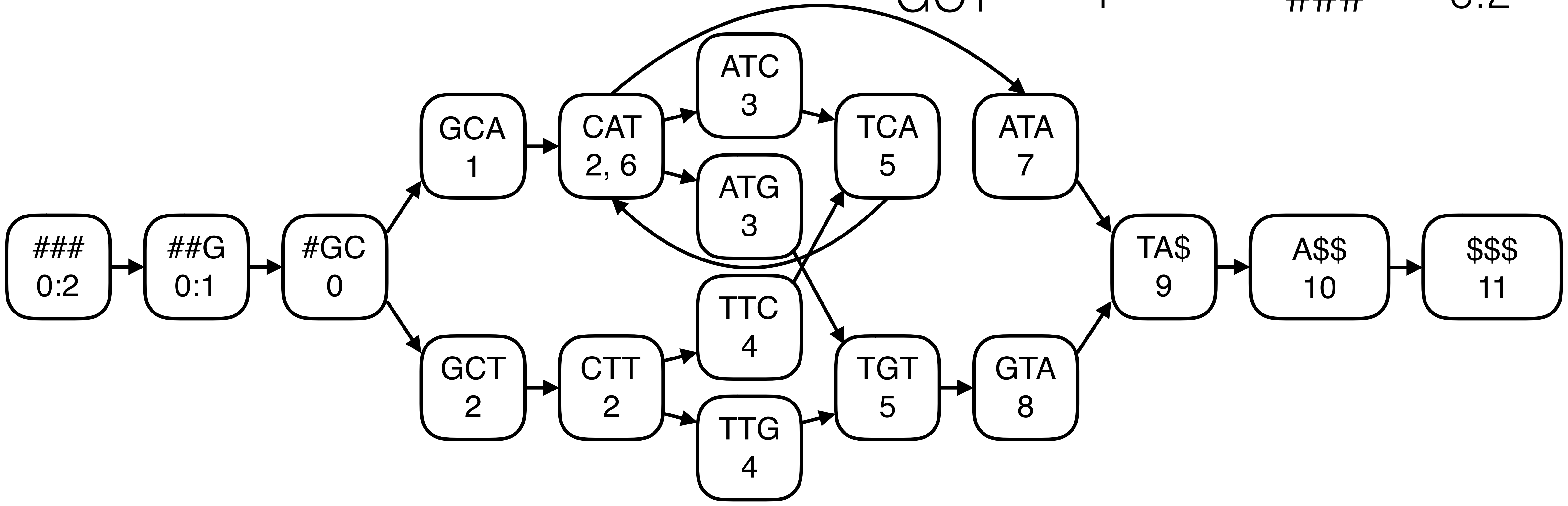
- The kmer index is a simple path index. It consists of a set of key-value pairs.
- A hash table supports fast kmer queries.
- Binary search in a sorted array is slower but supports queries shorter than k.
- Index size: terabytes.

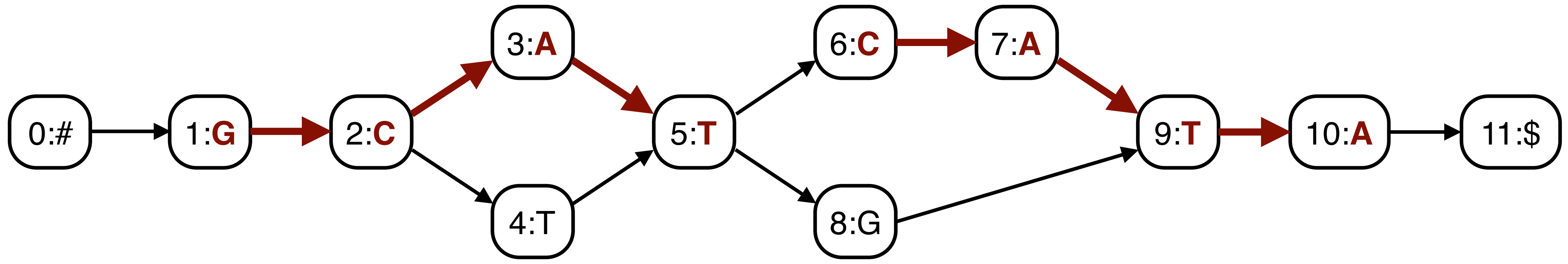
Key	Value	Key	Value
$$$	11	GTA	8
A$$	10	TA$	9
ATA	7	TCA	5
ATC	3	TGT	5
ATG	3	TTC	4
CAT	2, 6	TTG	4
CTT	2	#GC	0
GCA	1	##G	0:1
GCT	1	###	0:2



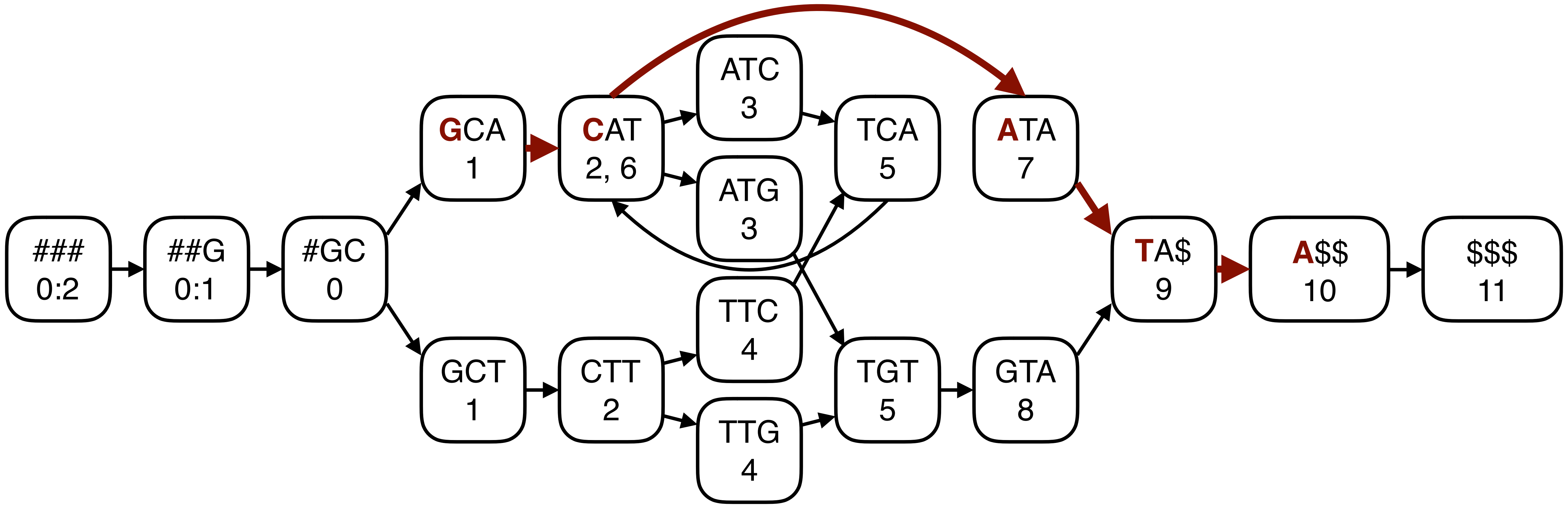
- We can represent the kmer index as a de Bruijn graph.
- We label each node with the first character of the key.
- The de Bruijn graph approximates the variation graph. There are no false negatives, and no false positives shorter than k+1.

Key	Value	Key	Value
$$$	11	GTA	8
A$$	10	TA$	9
ATA	7	TCA	5
ATC	3	TGT	5
ATG	3	TTC	4
CAT	2, 6	TTG	4
CTT	2	#GC	0
GCA	1	##G	0:1
GCT	1	###	0:2





Paths longer than k+1 may be false positives, but we can verify them in the input graph.



Succinct de Bruijn graphs

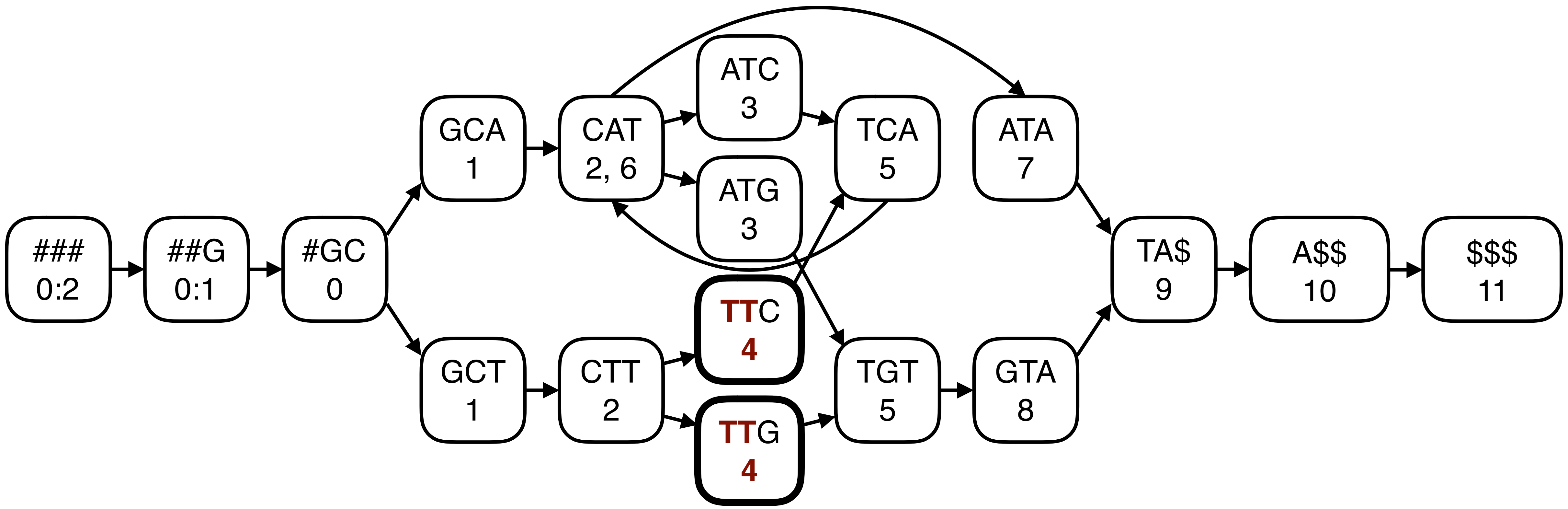
Node	BWT	IN	OUT
$$$	A	1	1
A$$	T	1	1
ATA	C	1	1
ATC	C	1	1
ATG	C	1	1
CAT	GT	01	001
CTT	G	1	01
GCA	#	1	1
GCT	#	1	1
GTA	T	1	1
TA$	AG	01	1
TCA	AT	01	1
TGT	AT	01	1
TTC	C	1	1
TTG	C	1	1
#GC	#	1	01
##G	#	1	1
###	$	1	1

- Sort the nodes, write the predecessor labels to BWT, and encode the indegrees and the outdegrees in unary to bitvectors IN and OUT.
- The result is an FM-index for de Bruijn graphs.
- Bowe et al: Succinct de Bruijn graphs. WABI 2012.
- Index size: hundreds of gigabytes.

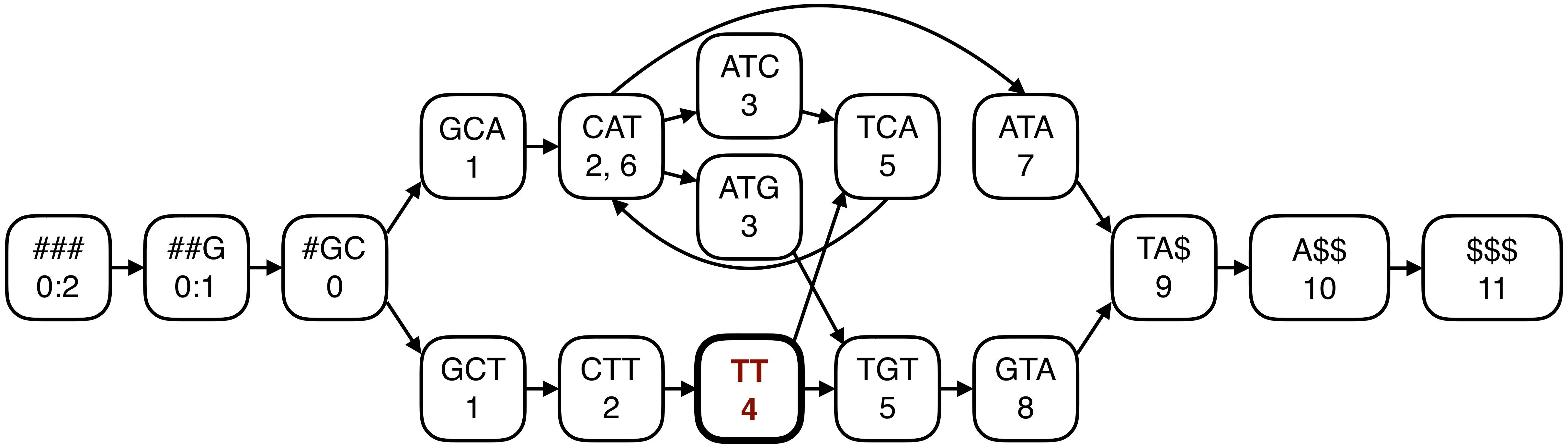
GCSA2

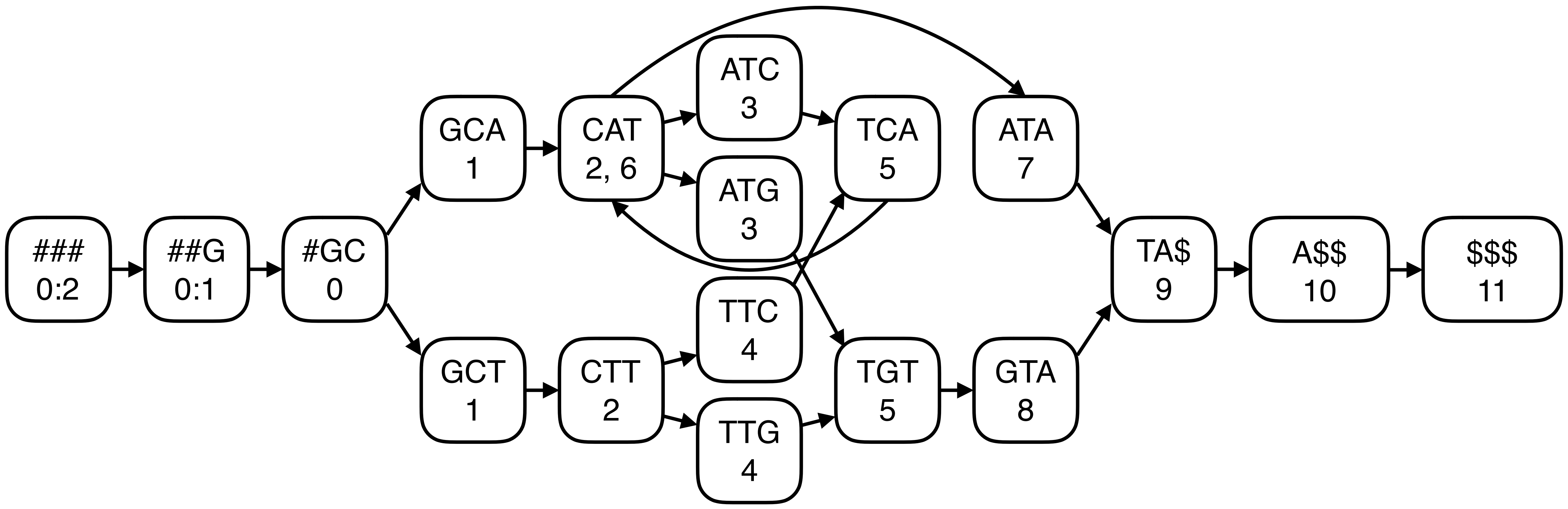
Path graphs

- High-order de Bruijn graphs of a graph have redundant subgraphs, if shorter keys already specify the position uniquely.
- We can compress the de Bruijn graph by merging such subgraphs.
- Path graphs generalize de Bruijn graphs by using any prefix-free set of strings as keys.
- Inspired by: Sirén et al: Indexing Graphs for Path Queries with Applications in Genome Research. TCBB, 2014.

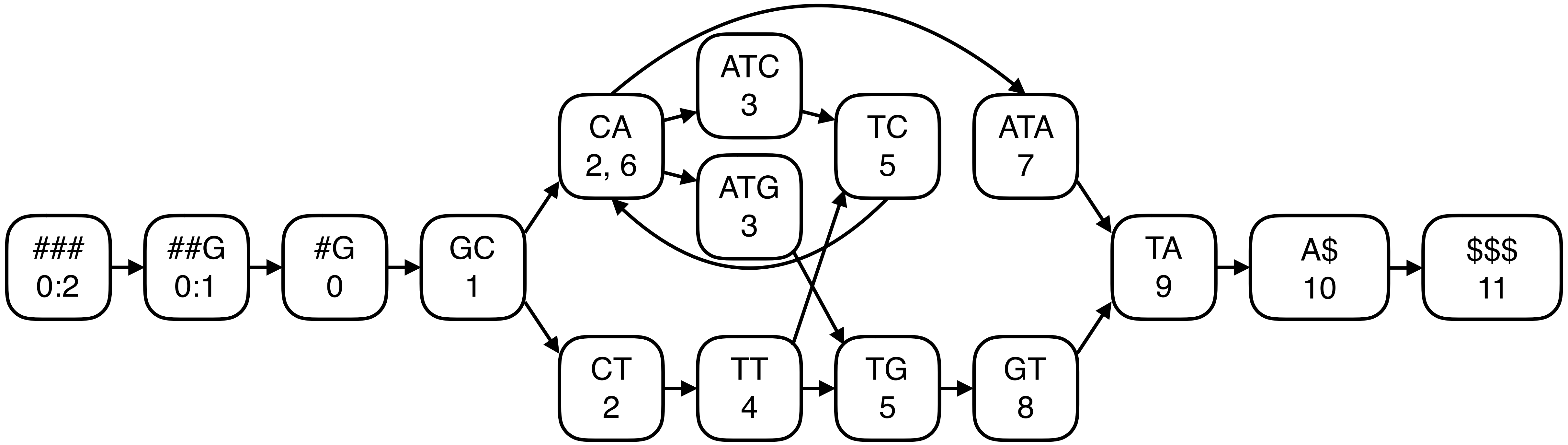


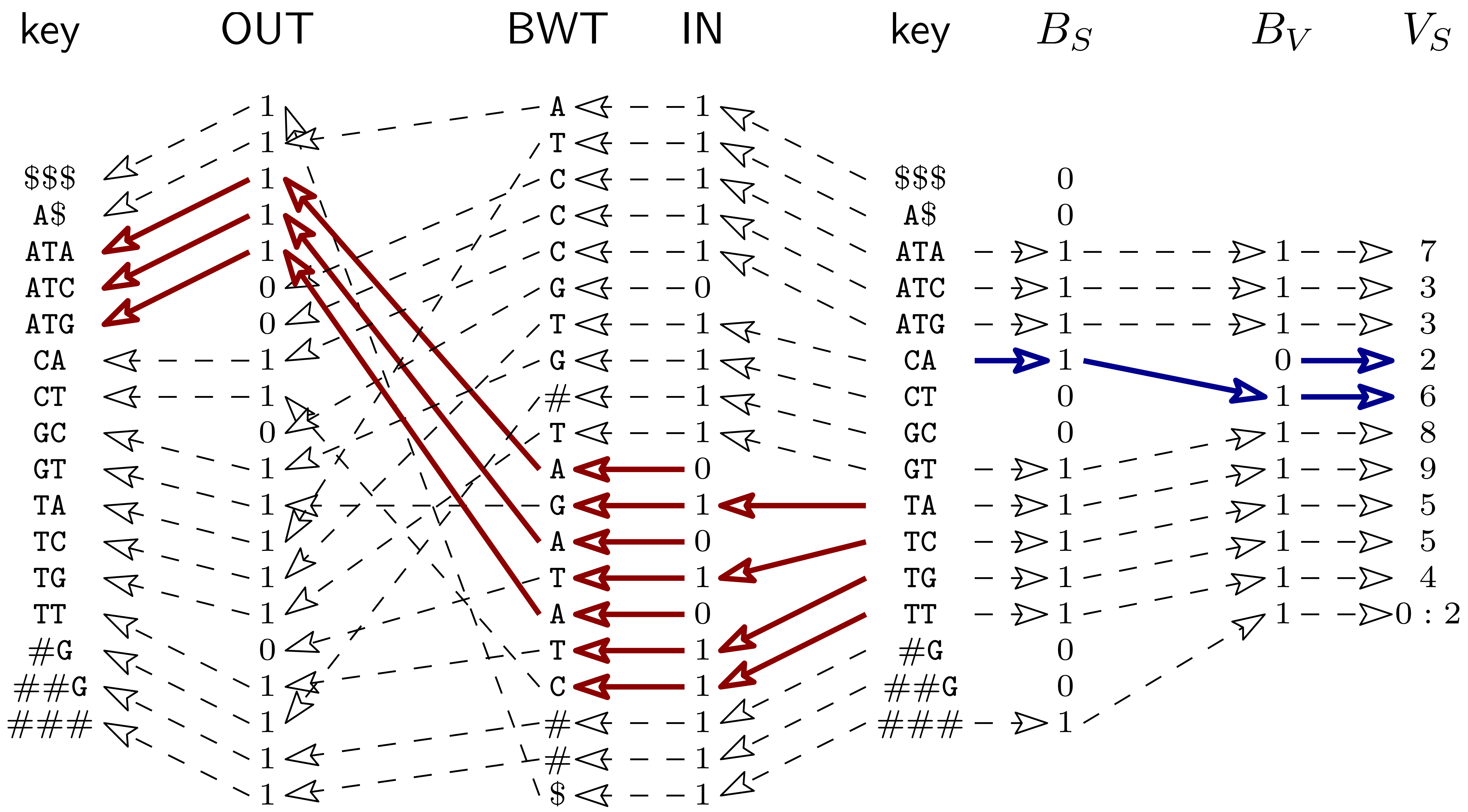
We can merge nodes sharing a prefix without affecting queries, if the value sets are identical.



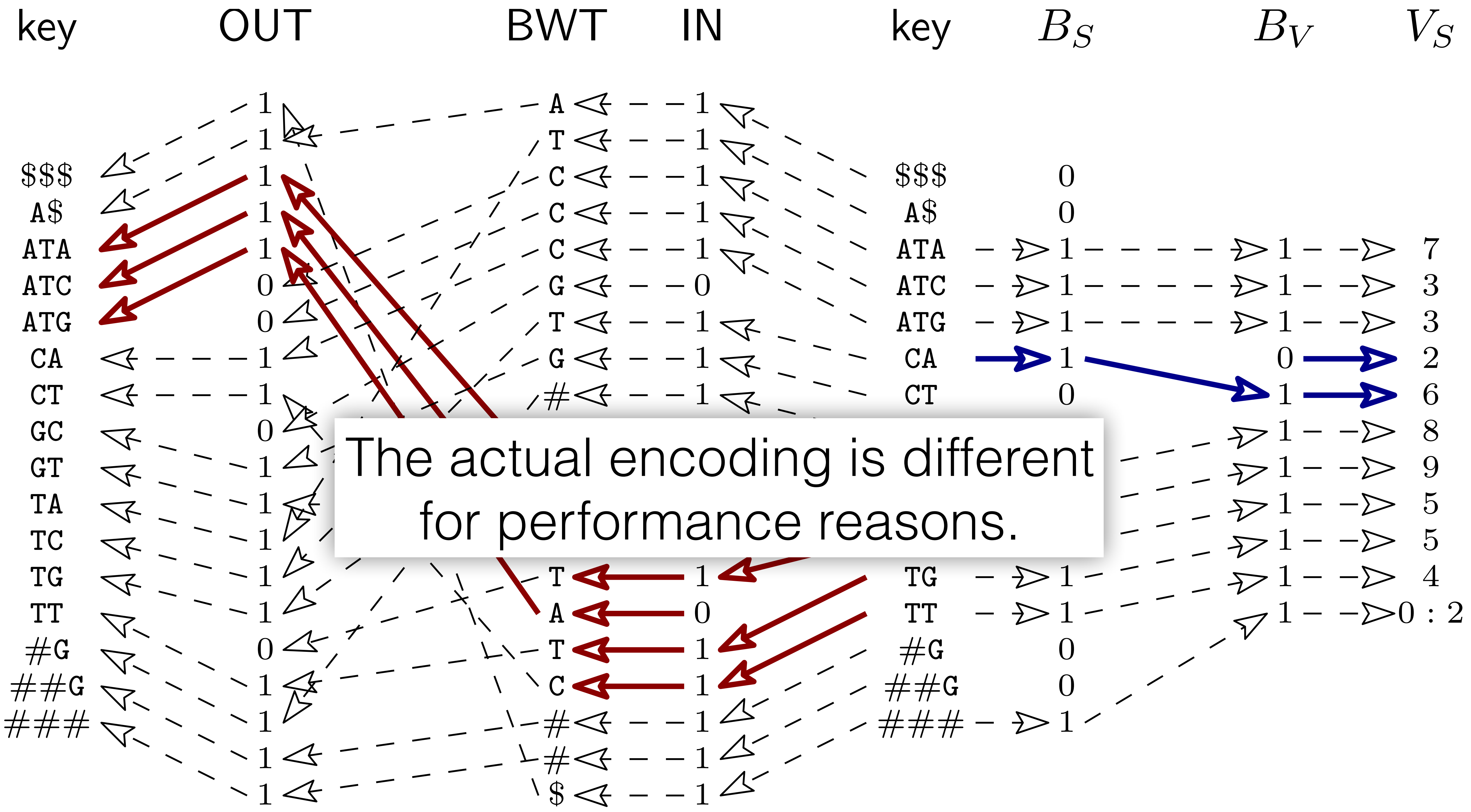


If we keep merging the nodes, we get a (maximally) pruned de Bruijn graph, which behaves intuitively.





We can encode the result in the same way as in the succinct de Bruijn graph / GCSA.



The actual encoding is different for performance reasons.

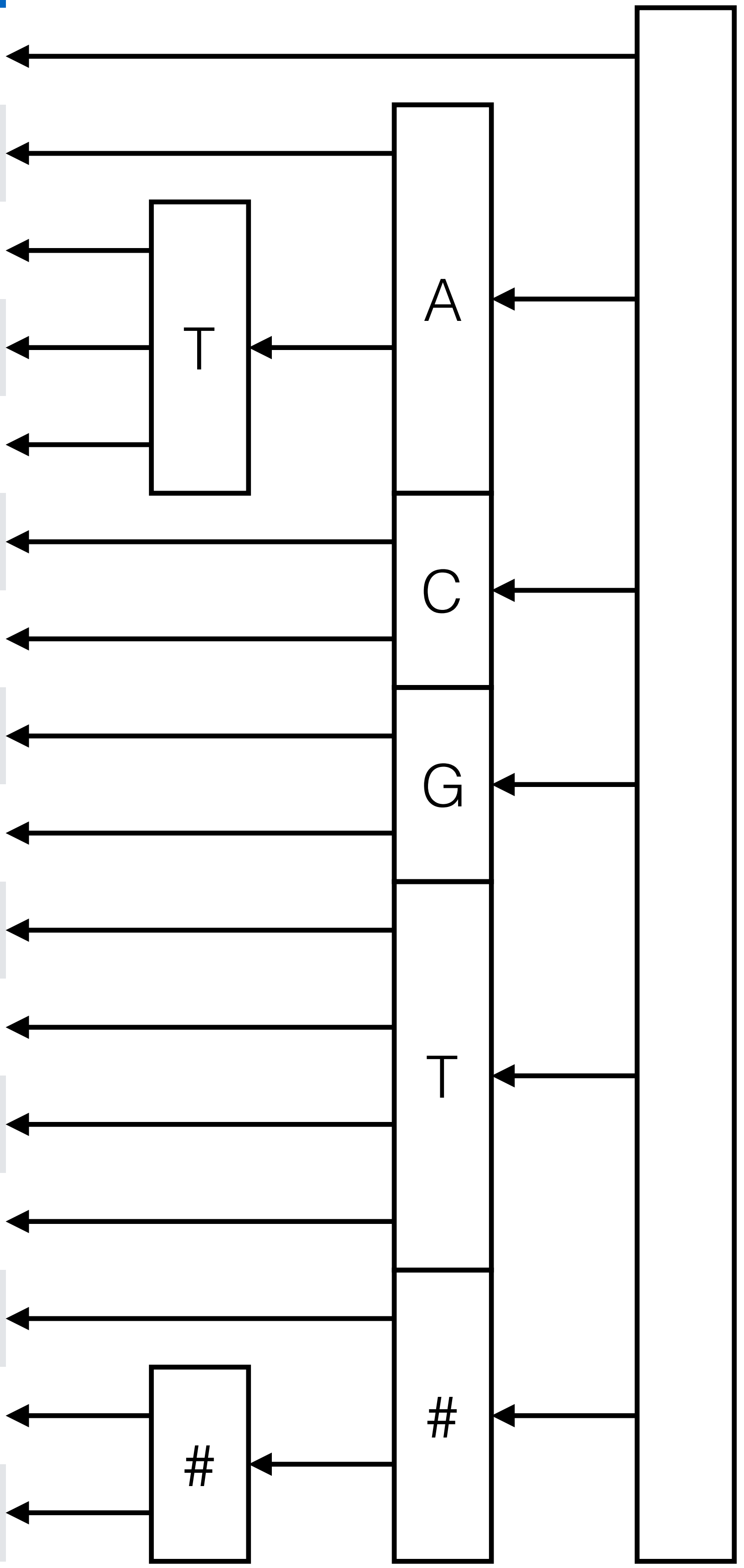
We can encode the result in the same way as in the succinct de Bruijn graph / GCSA.

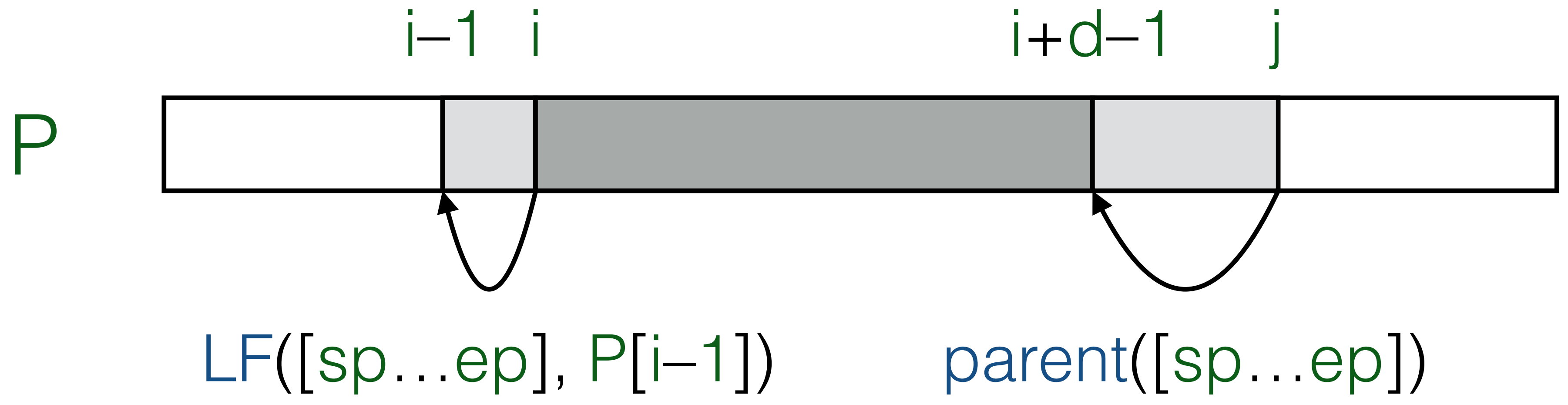
The compacted trie of keys resembles a suffix tree, especially if the graph is a maximally pruned de Bruijn graph.

The LCP interval tree is equivalent to the suffix tree. (Abouelhoda et al: Replacing suffix trees with enhanced suffix arrays. JDA, 2004.)

We can simulate the suffix tree with next/previous smaller value queries and range minimum queries in the LCP array. (Fischer et al: Faster entropy-bounded compressed suffix trees. TCS, 2009)

Key	LCP
$$$	0
A$	0
ATA	1
ATC	2
ATG	2
CA	0
CT	1
GC	0
GT	1
TA	0
TC	1
TG	1
TT	1
#G	0
##G	1
###	2





If lexicographic range [sp…ep] **matches** substring P[i…j] of the **pattern**, we can

- **extend** the match to the **left** with LF(); and
- **remove** characters from the **right** with parent().

This allows us to find **maximal exact matches**, which can be used e.g. as seeds in read alignment.

Ohlebusch et al: **Computing Matching Statistics and Maximal Exact Matches on Compressed Full-Text Indexes**. SPIRE 2010.

Path length	16→32	16→64	16→128
Kmers Nodes	6.20G 4.37G	16.7G 5.24G	**116G** **5.73G**
Index size	13.2 GB 18.2 bits / kmer	13.5 GB 6.99 bits / kmer	**14.6 GB** **1.08 bits / kmer**
Construction: Time Memory Disk	7.44 h 59.8 GB 387 GB	10.4 h 51.9 GB 415 GB	**14.1 h** **52.3 GB** **478 GB**
I/O: Read Write	1.37 TB 0.88 TB	2.03 TB 1.51 TB	**2.78 TB** **2.25 TB**

1000GP human variation (forward strand only)

```
vg mod -p -l 16 -e 4 | vg mod -S -l 100
```

32 cores, 256 GB memory, distributed Lustre file system

k	Index	kmers	Matched	find()	locate()
16	GCSA2	351584	347453	4.75 μs	5.85 μs
	BWA	351584	320764	3.64 μs	4.65 μs
	csa_wt	351584	301538	6.00 μs	2.43 μs
32	GCSA2	351555	333258	10.8 μs	5.44 μs
	BWA	351555	156080	6.57 μs	3.19 μs
	csa_wt	351555	153957	10.9 μs	2.16 μs

GCSA2: Order-128 index for the pruned variation graph

BWA: The FM-index from BWA v0.7.15 for the reference and its reverse complement

csa_wt: Fast FM-index from SDSL for the reference

Average time for find queries (per query) and locate queries (per distinct occurrence) with kmers extracted from the non-pruned variation graph.

Pruning the Variation Graph

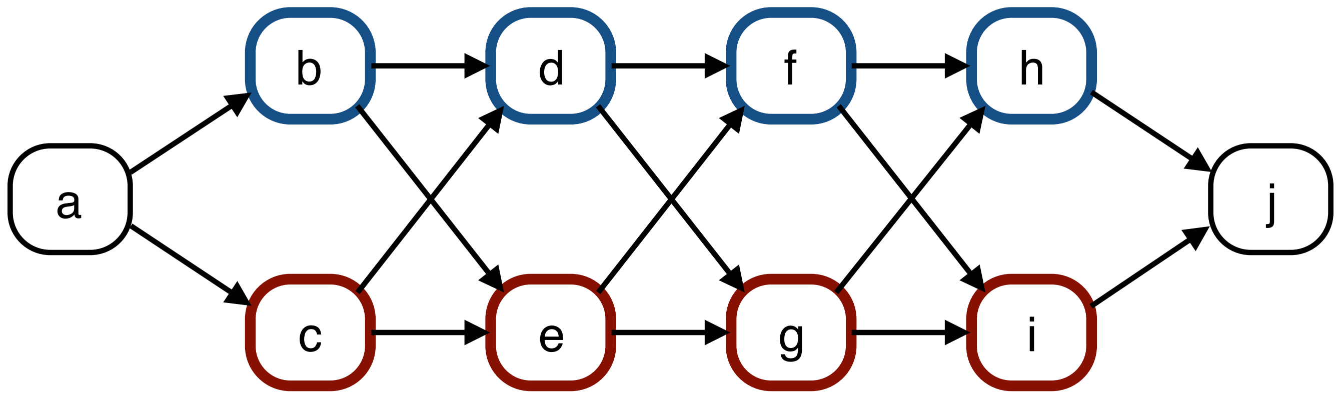
Complex regions

- A whole-genome human variation graph based on 1000GP variation contains trillions (quadrillions?) of distinct 128-mers.
- Almost all of them are from a few complex regions.
- We cannot index all potential recombinations in such regions. Even if we could, the resulting index would probably be too biased.
- vg and GCSA2 have several ways for dealing with the complex regions.

Pruning

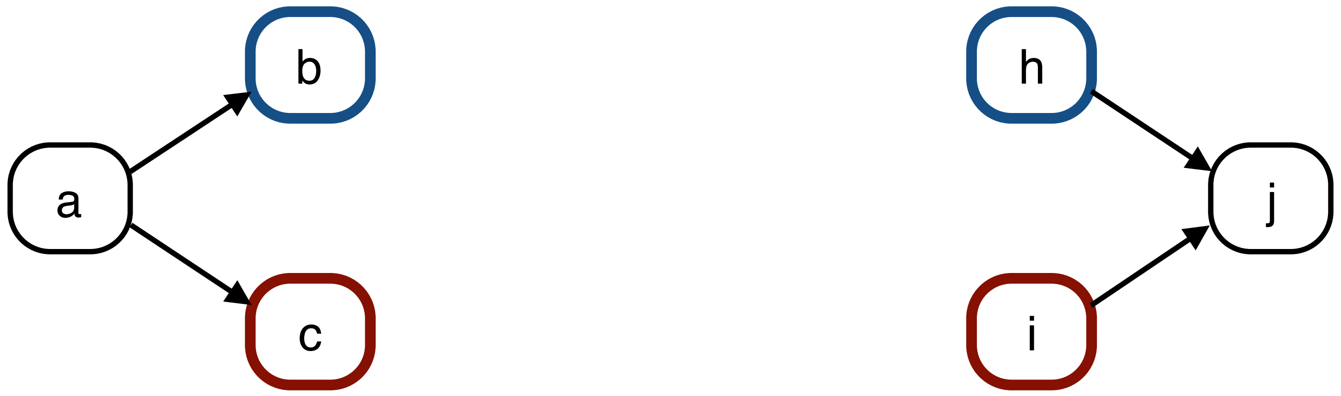
```
vg mod -p -l 16 -e 4
```

Remove paths of length 16 crossing more than 4 nontrivial edges.



```
vg mod -S -l 100
```

Remove subgraphs shorter than 100 bases.

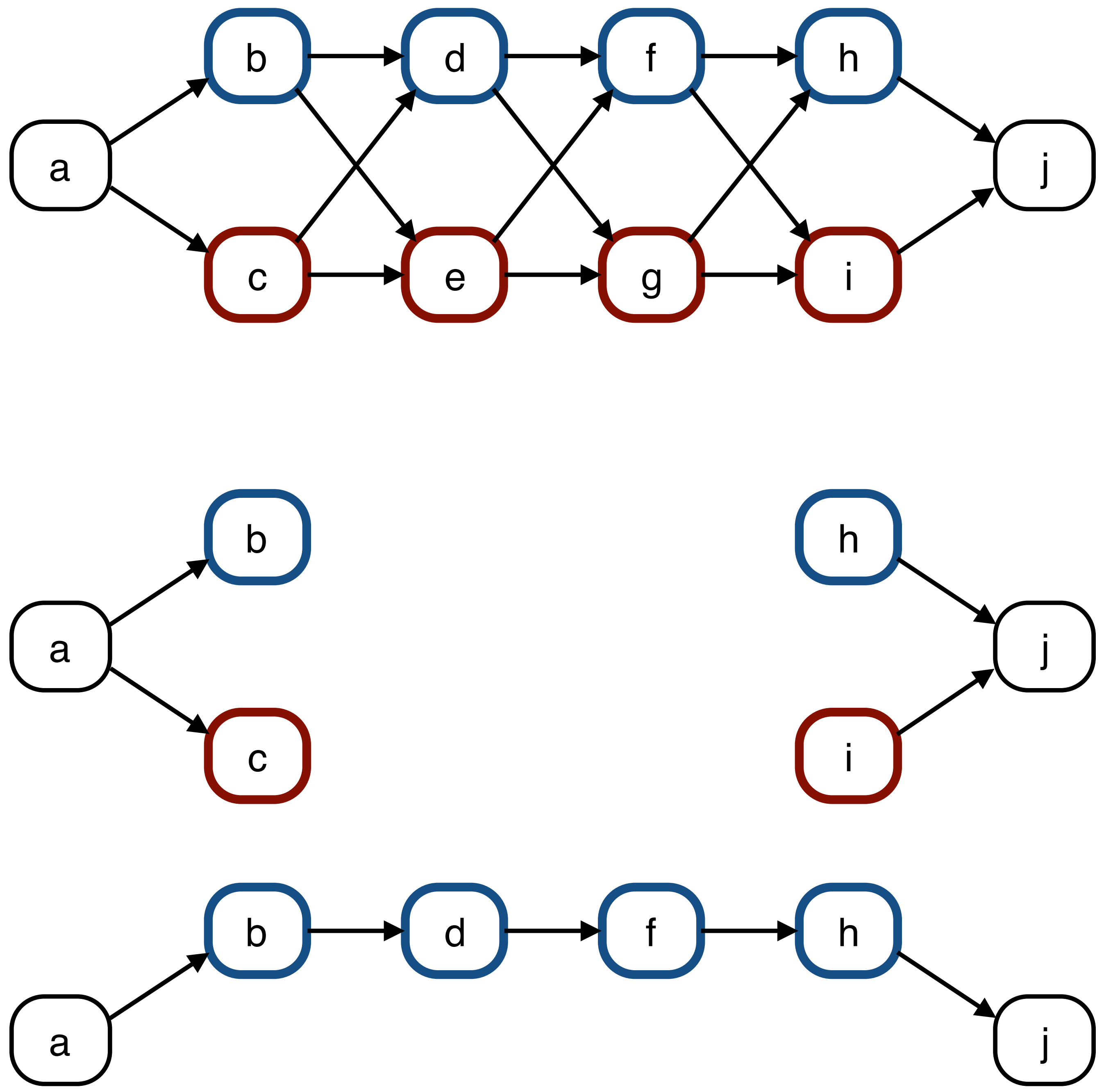


- Easy and efficient.
- Complex regions may be removed completely.

Indexing subgraphs

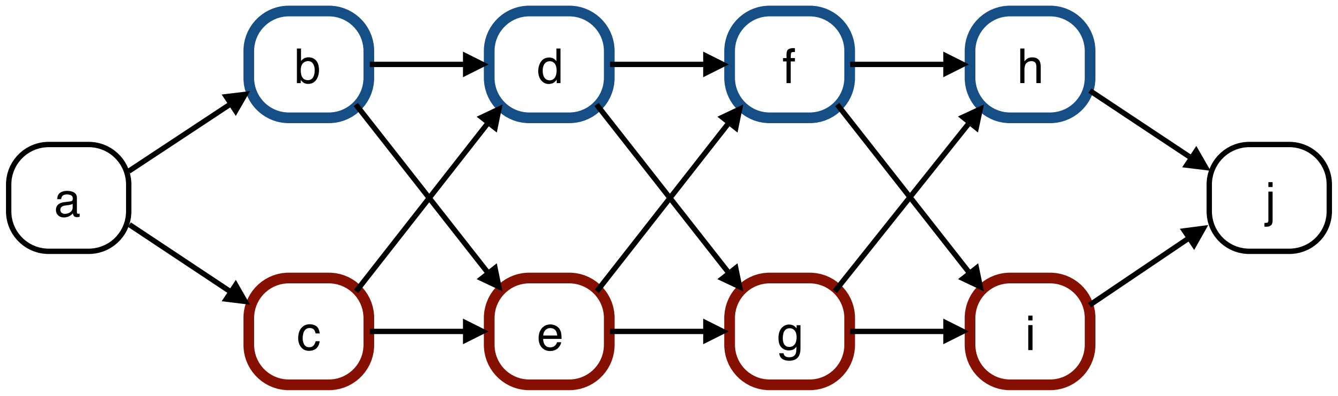
We can index **overlapping subgraphs** (e.g. a pruned variation graph and the reference path) and merge the results into a single index.

- Guarantees that the **entire genome** is indexed.
- **Redundant paths** can make index construction more expensive.
- Requires a **reverse deterministic** graph for the fast GCSA encoding.



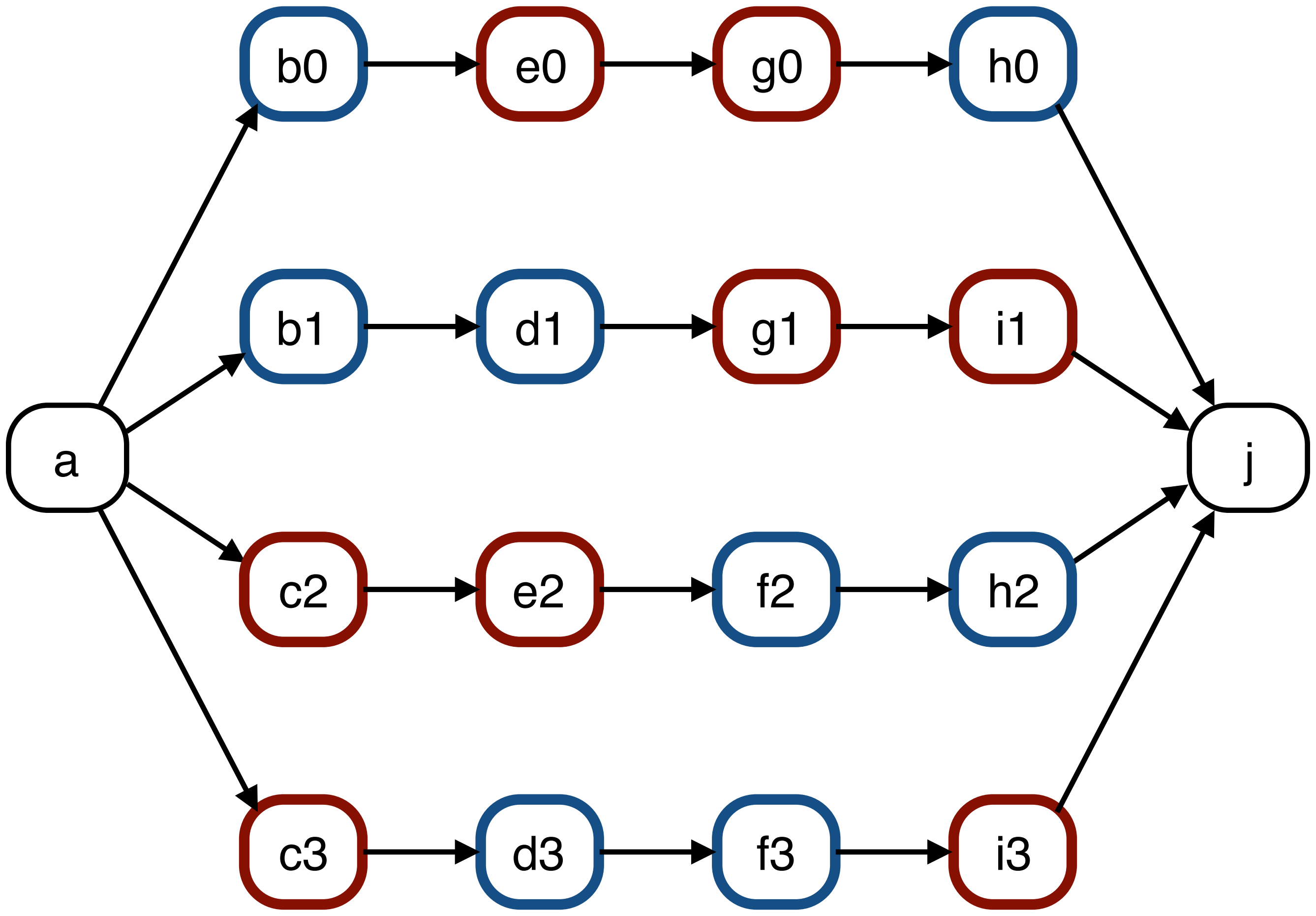
Indexing haplotypes

Index only paths corresponding to known haplotypes in complex regions.



Multiple nodes of the input graph map to the same node in the variation graph.

- Guarantees that the entire genome and all observed variation is indexed.
- Not implemented yet in vg.



Conclusions

- The design of a path index is a trade-off between index size, query performance, maximum query length, and ignoring complex regions of the graph.
- GCSA2 prioritizes performance and size, while supporting queries long enough to map short reads in one piece.
- It uses a de Bruijn graph as a kmer index, compresses it by merging redundant subgraphs, and encodes the result as a compressed suffix tree.
- Sirén: Indexing Variation Graphs. arXiv:1604.06605, 2016. Accepted to ALENEX 2017. https://github.com/jltsiren/gcsa2