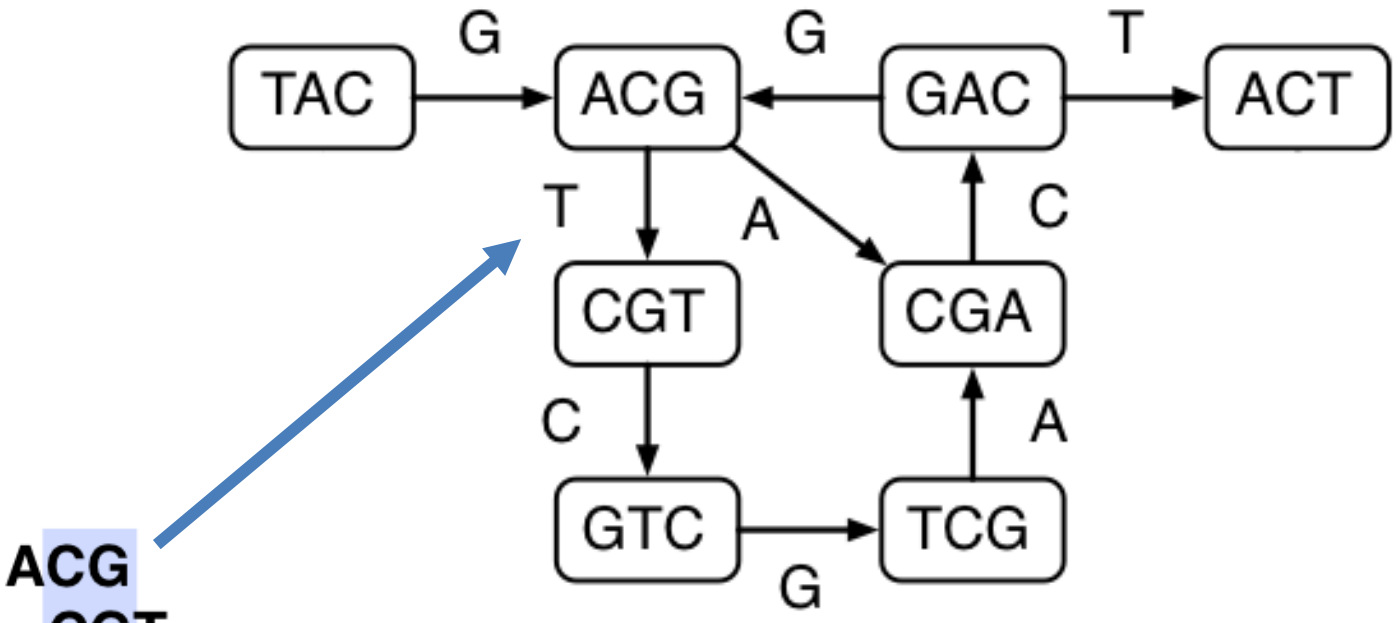
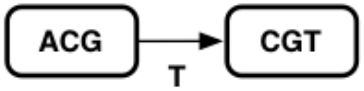
Memory Efficient Genome Assembly

- One problem has been solved but one remains?

- One problem has been solved but one remains?
  - We have solved our time problem using Eulerian (i.e., de Bruijn graph) genome assembly paradigms; however, we have a memory problem
  - The de Bruijn graph requires the construction of a graph over a billion nodes and vertices.
  - We can quickly run out of memory for larger genomes.

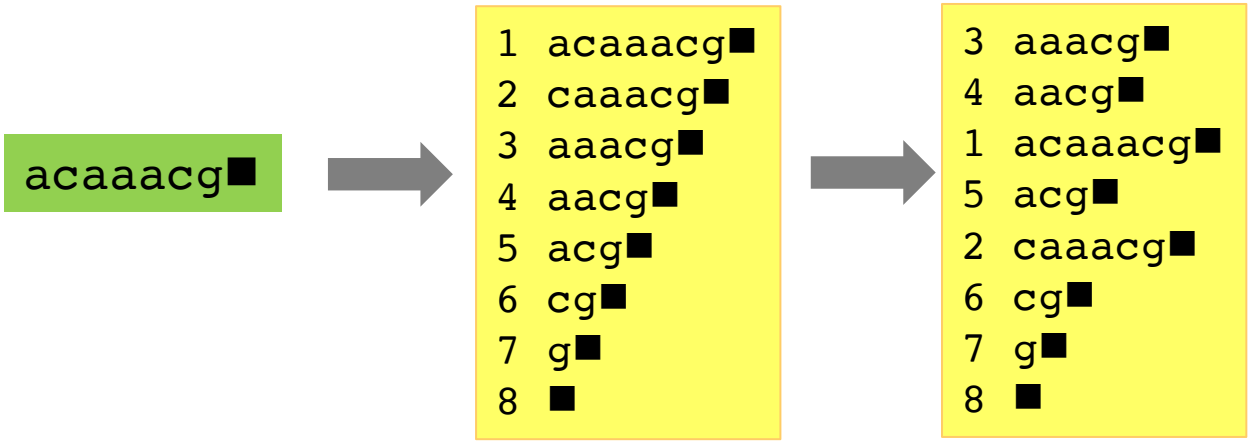
T = TACGACGTCGACT





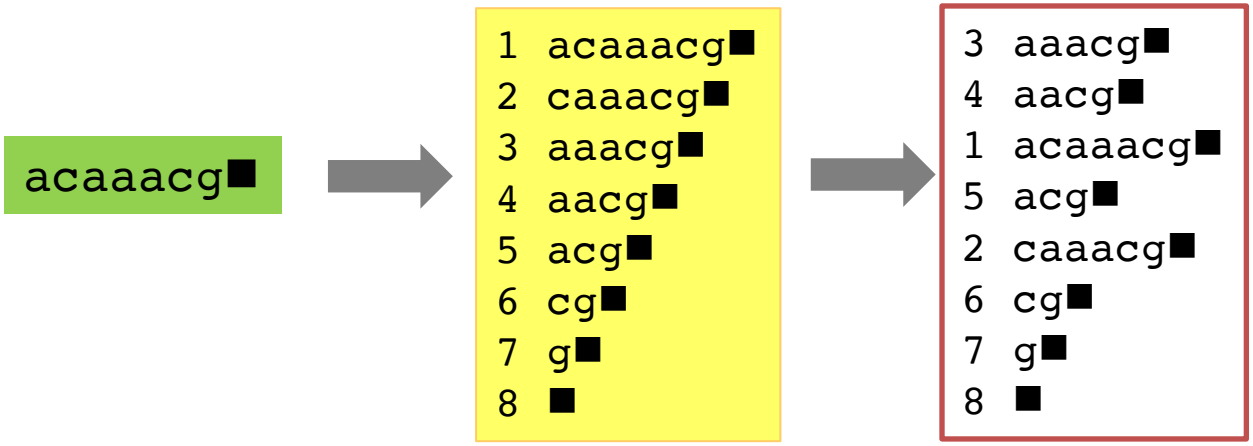
BWT and FM-index

A suffix array (SA) of string S is an array of the suffixes of S sorted into alphabetical order.



BWT and FM-index

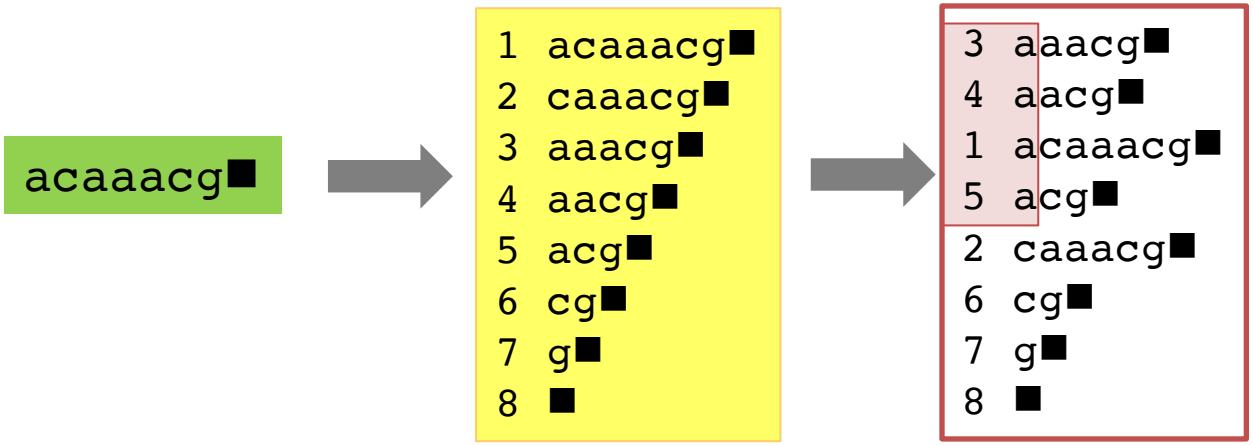
A suffix array (SA) of string S is an array of the suffixes of S sorted into alphabetical order.



The suffix array clusters all the occurrences of every pattern together into a contiguous range!

BWT and FM-index

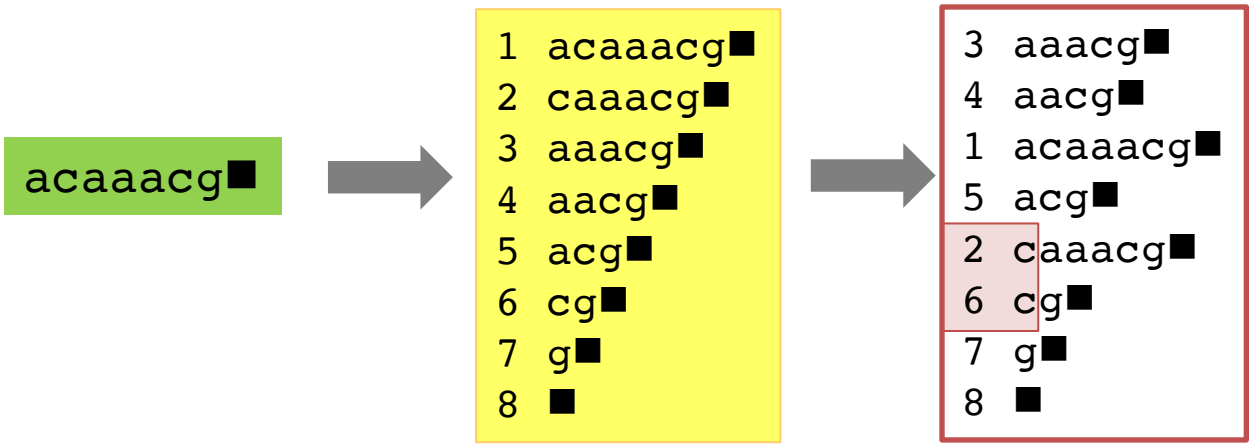
A suffix array (SA) of string S is an array of the suffixes of S sorted into alphabetical order.



The suffix array clusters all the occurrences of every pattern together into a contiguous range!

BWT and FM-index

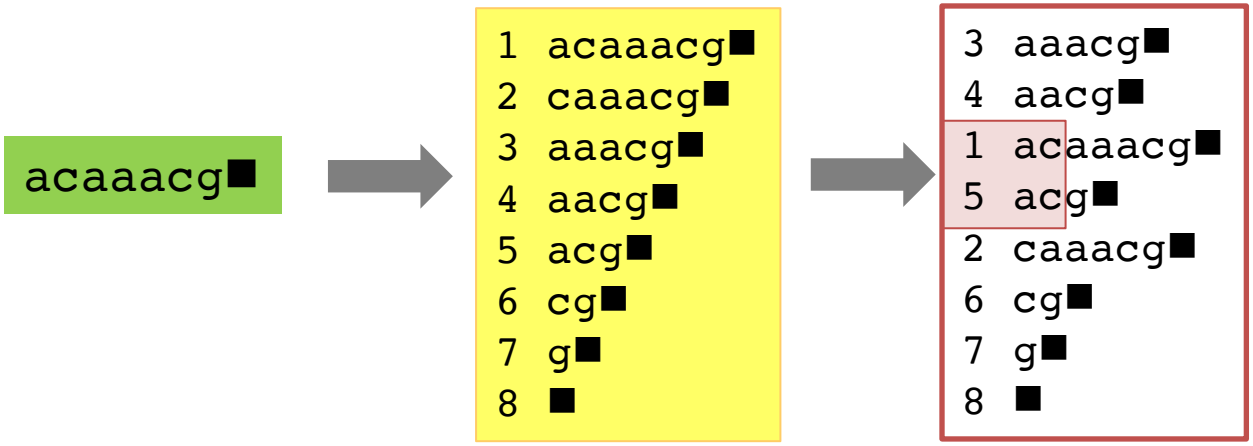
A suffix array (SA) of string S is an array of the suffixes of S sorted into alphabetical order.



The suffix array clusters all the occurrences of <u>every</u> pattern together into a contiguous range!

BWT and FM-index

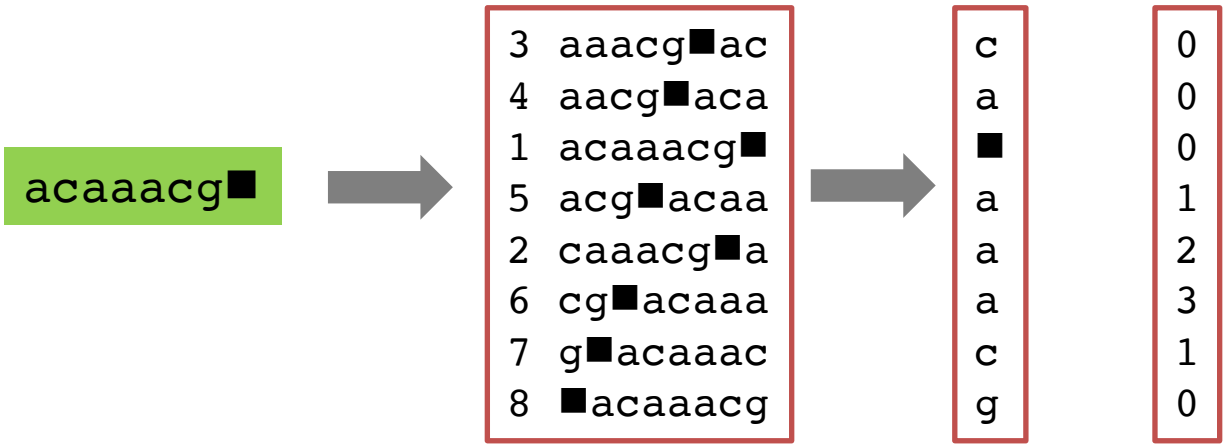
A suffix array (SA) of string S is an array of the suffixes of S sorted into alphabetical order.



The suffix array clusters all the occurrences of <u>every</u> pattern together into a contiguous range!

BWT and FM-index

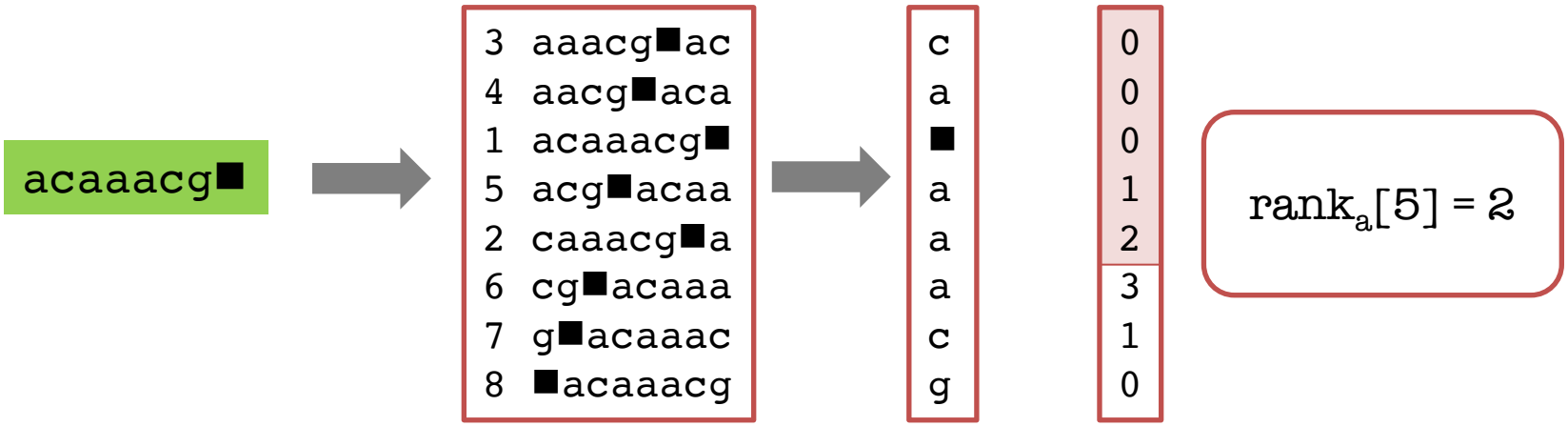
The Burrows-Wheeler Transform (BWT) is a permutation of the string such that BWT[i] = S[SA[i] - 1].



$\mathrm{rank}_K(i)$: return the number of K's in S[1,i]

BWT and FM-index

The Burrows-Wheeler Transform (BWT) is a permutation of the string such that BWT[i] = S[SA[i] - 1].



$rank_K(i)$: return the number of K's in S[1,i]

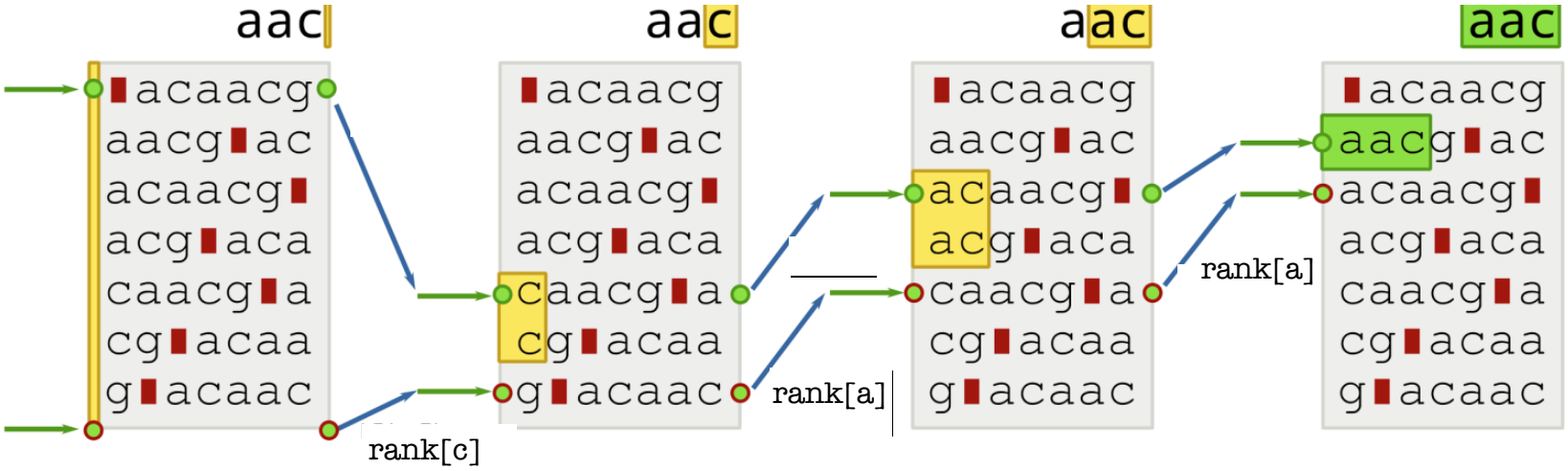
BWT and FM-index

The Burrows-Wheeler Transform (BWT) is a permutation of the string such that BWT[i] = S[SA[i] - 1].

FM-index is the compressed version of the BWT and rank.

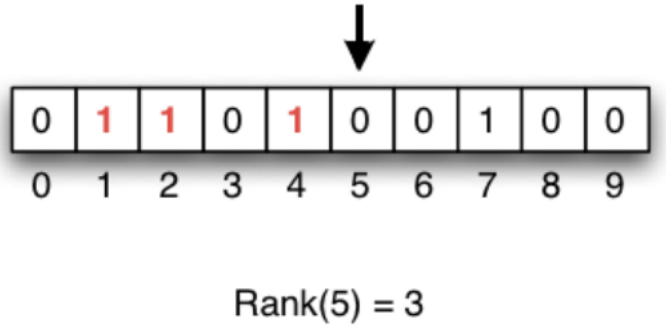
FM-Index Backward Search

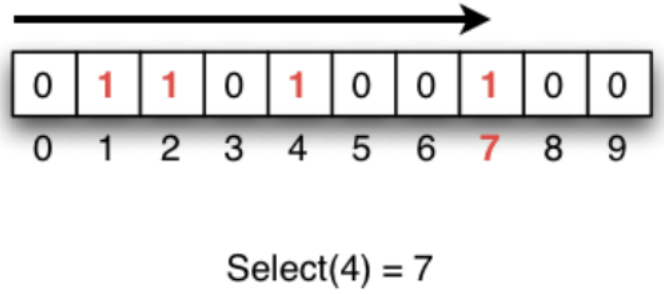
A recursive algorithm for finding substrings using rank and BWT



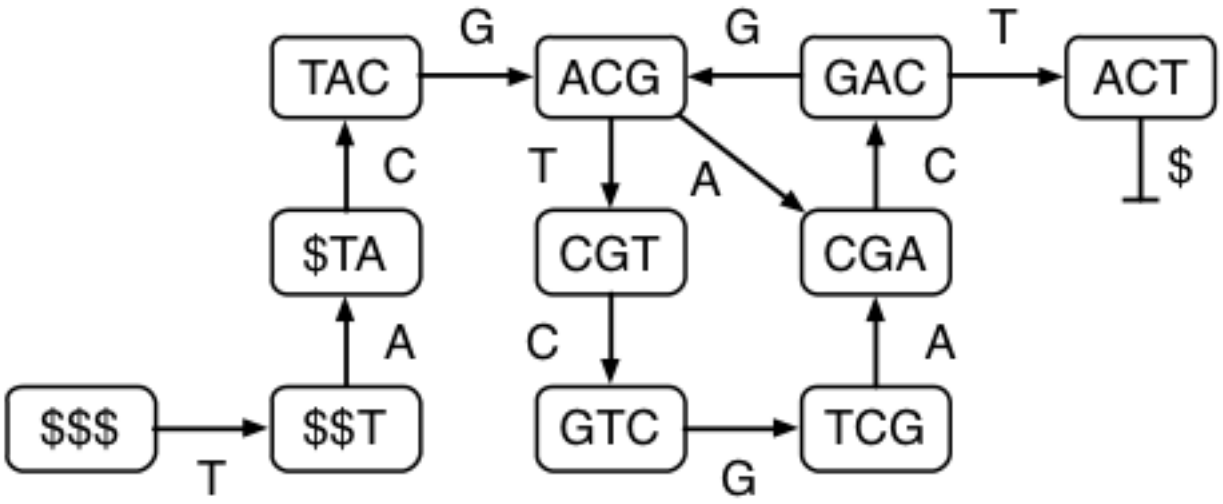
Rank and Select

select$_c$(i): returns the position of the ith occurrence of symbol c. (It is ***a bit like*** the inverse function of rank).



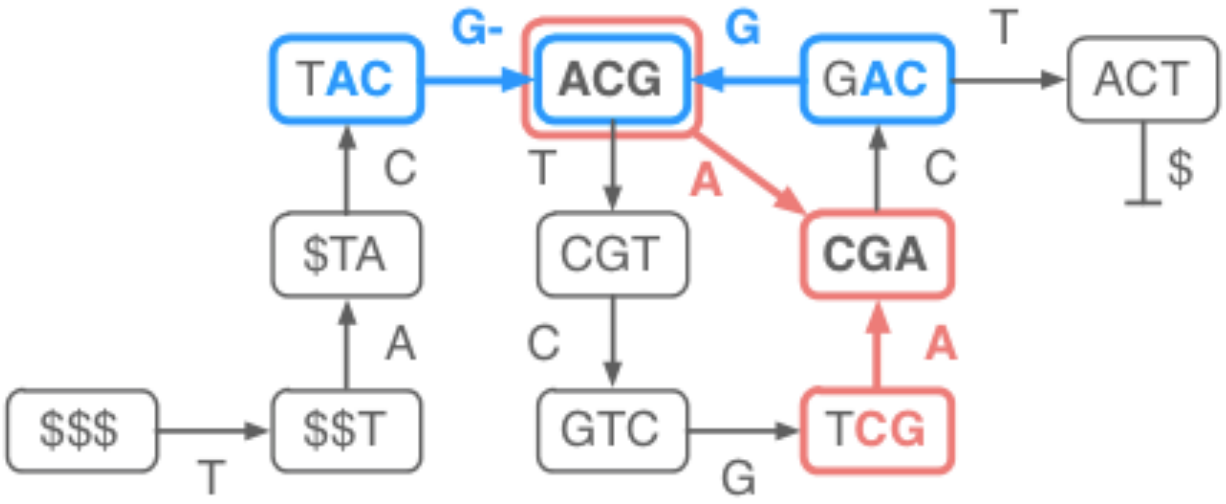


T = $$$TACGACGTCGACT$



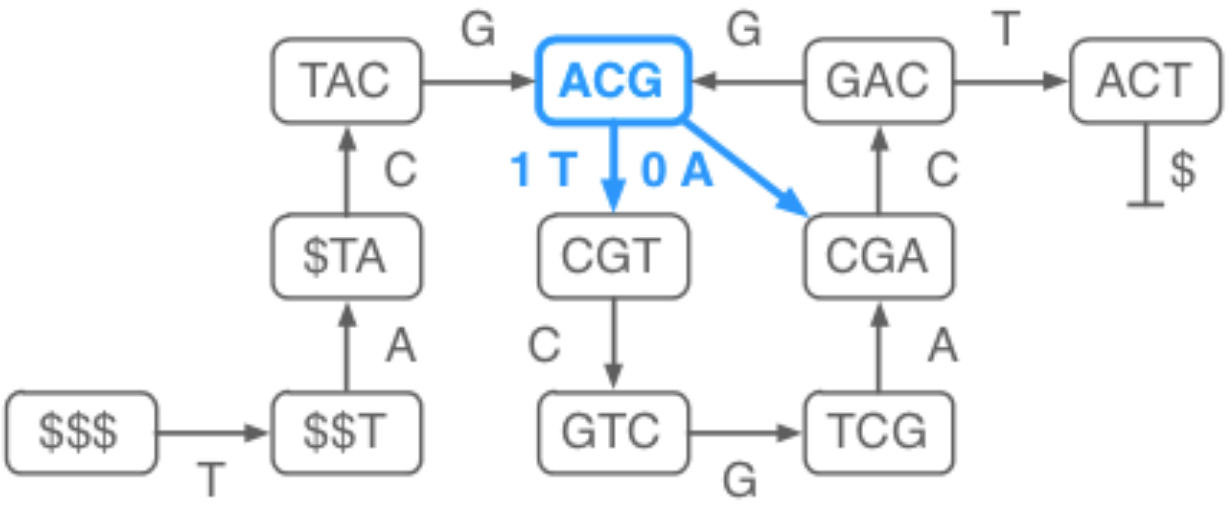
Node			*W*
$	$	$	T
C	G	A	C
$	T	A	C
G	A	C	G
G	A	C	T
T	A	C	G
G	T	C	G
A	C	G	A
A	C	G	T
T	C	G	A
$	$	T	A
A	C	T	$
C	G	T	C

T = $$$TACGACGTCGACT$



L	Node			W
1	$	$	$	T
1	C	G	A	C
1	$	T	A	C
0	G	A	C	G
1	G	A	C	T
1	T	A	C	G-
1	G	T	C	G
0	A	C	G	A
1	A	C	G	T
1	T	C	G	A-
1	$	$	T	A
1	A	C	T	$
1	C	G	T	C

T = $$$TACGACGTCGACT$



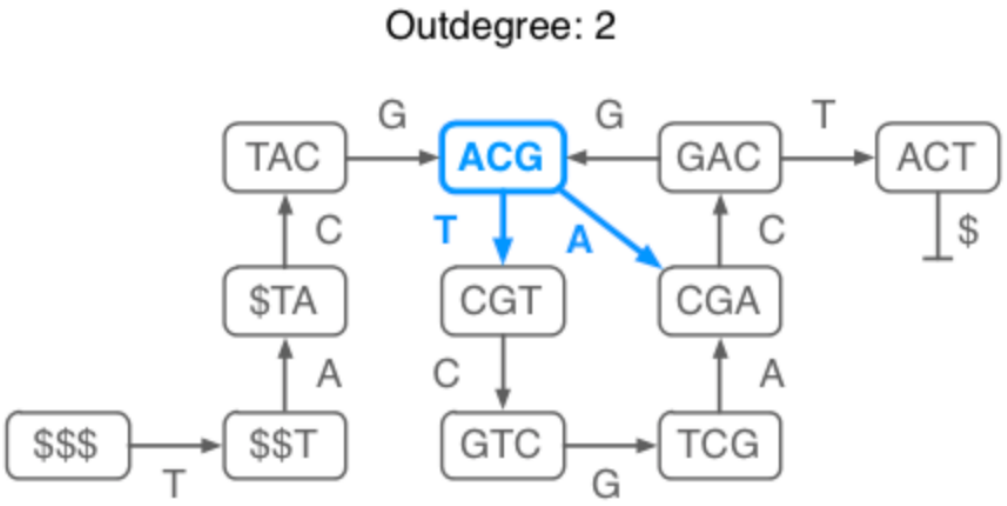
L	*Node*			*W*
1	$	$	$	T
1	C	G	A	C
1	$	T	A	C
0	G	A	C	G
1	G	A	C	T
1	T	A	C	G-
1	G	T	C	G
0	A	C	G	A
1	A	C	G	T
1	T	C	G	A-
1	$	$	T	A
1	A	C	T	$
1	C	G	T	C

Operation	Description	Complexity
$forward(i)$	Return index of the *last edge* of the *node pointed to* by edge i.	$\mathcal{O}(1)$
$backward(i)$	Return index of the *first edge* that *points to the node* that the edge at i *exits*.	$\mathcal{O}(1)$

Using these two functions, we can implement the less confusing public interface below, which operate on node indexes:

Operation	Description	Complexity
$outdegree(v)$	Return number of outgoing edges from node v.	$\mathcal{O}(1)$
$outgoing(v, c)$	From node v, follow the edge labeled by symbol c.	$\mathcal{O}(1)$
$label(v)$	Return (string) label of node v.	$\mathcal{O}(k)$
$indegree(v)$	Return number of incoming edges to node v.	$\mathcal{O}(1)$
$incoming(v, c)$	Return predecessor node starting with symbol c, that has an edge to node v.	$\mathcal{O}(k \log \sigma)$

Outdegree



v	i	L	Node	W
		select(6)	*- select(5)*	
0	0	**1**	$ $ $	T
1	1	**1**	C G A	C
2	2	**1**	$ T A	C
3	3	**0**	G A C	G
	4	**1**	G A C	T
4	5	**1**	T A C	G-
5	**6**	**1**	G T C	G
6	7	**0**	A C G	A
	8	**1**	A C G	T
7	9	1	T C G	A-
8	10	1	$ $ T	A
9	11	1	A C T	$
10	12	1	C G T	C

Study the details here:

http://alexbowe.com/succinct-debruijn-graphs/