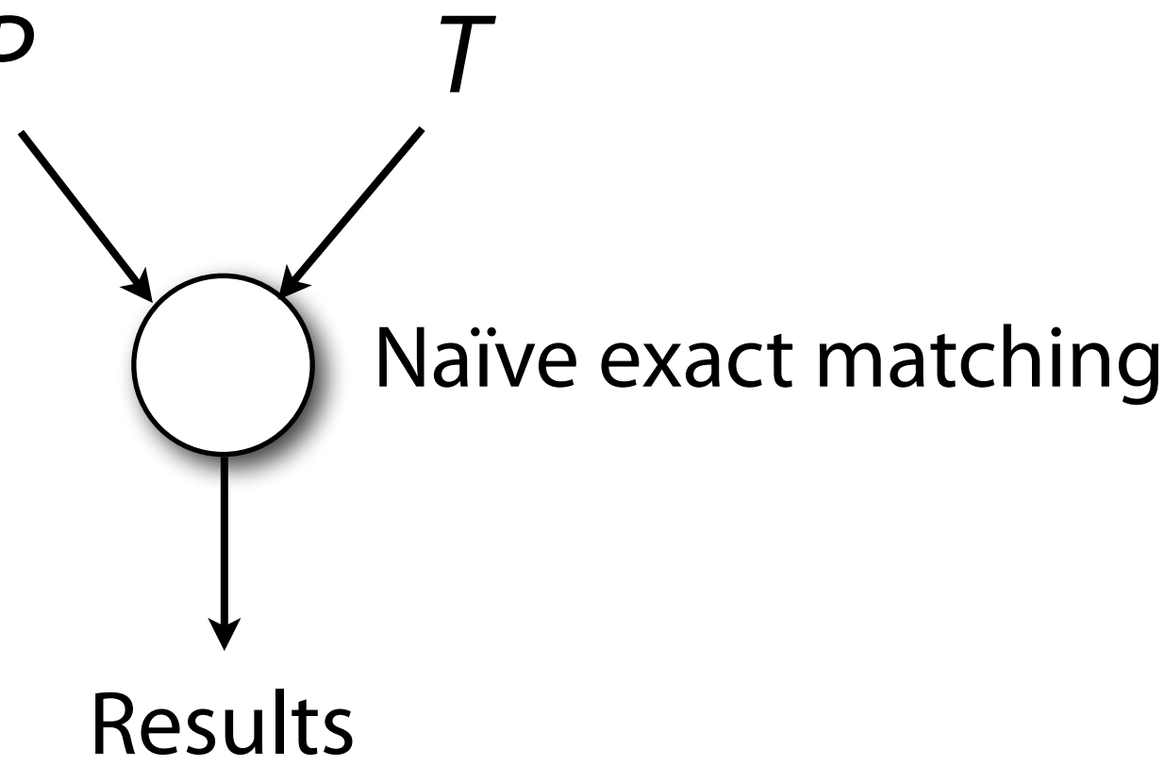
Indexing

Ben Langmead

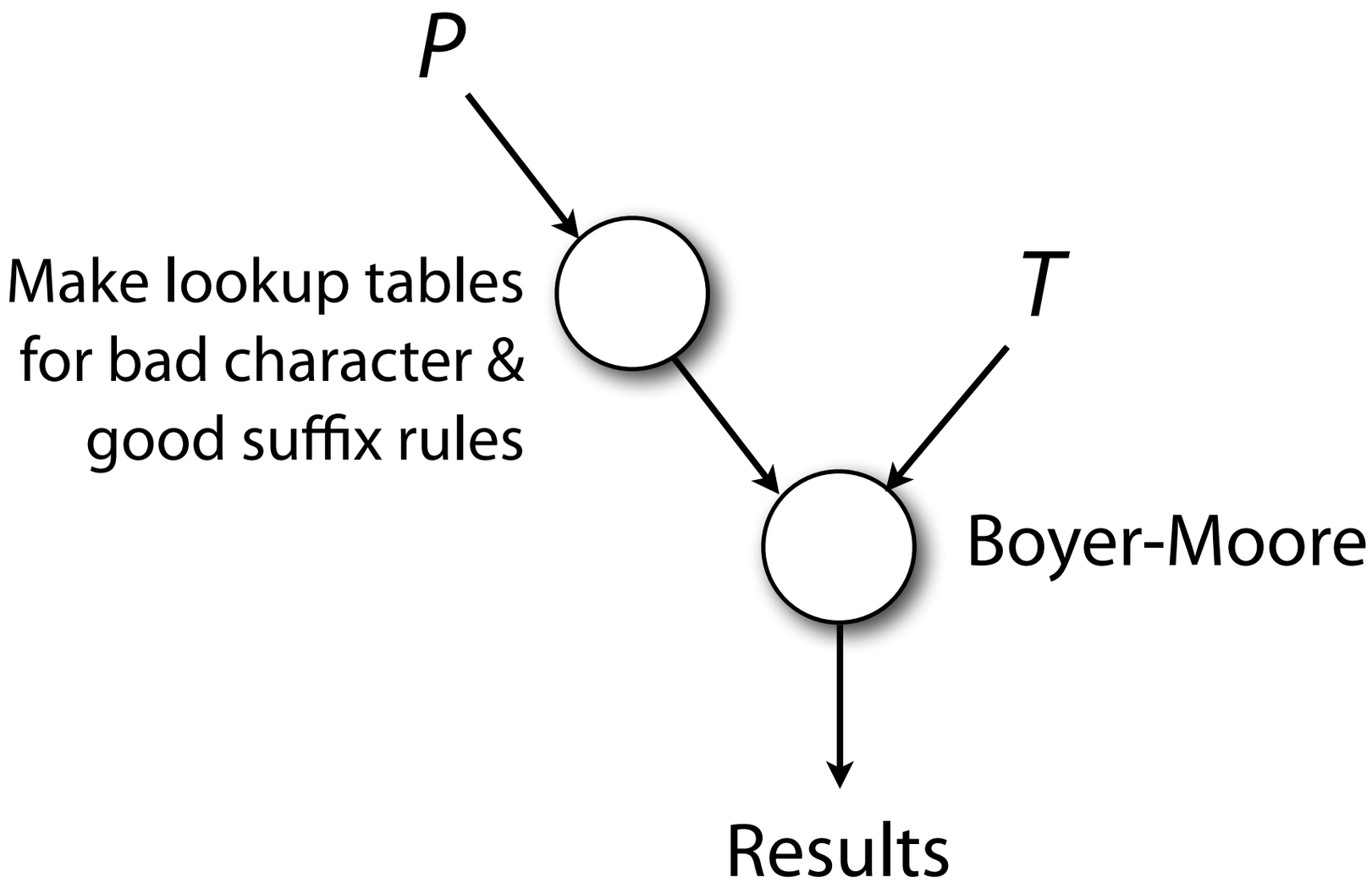
JOHNS HOPKINS
WHITING SCHOOL
of ENGINEERING

Department of Computer Science

Preprocessing: Naïve algorithm



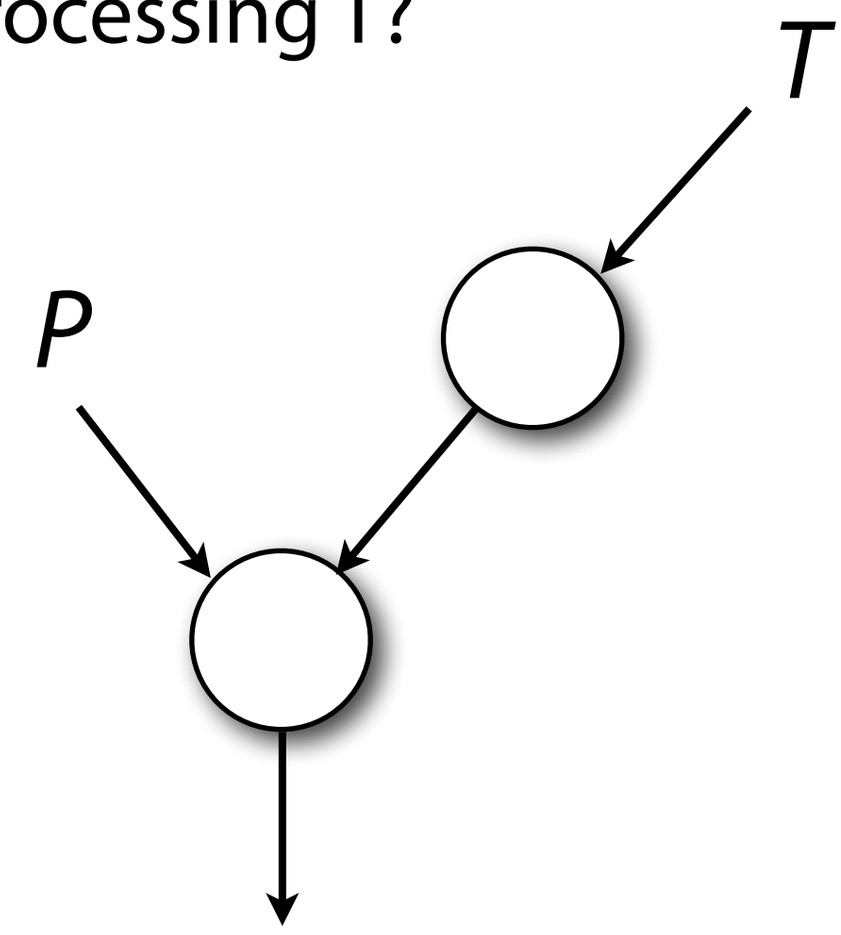
Preprocessing: Boyer-Moore



Preprocessing

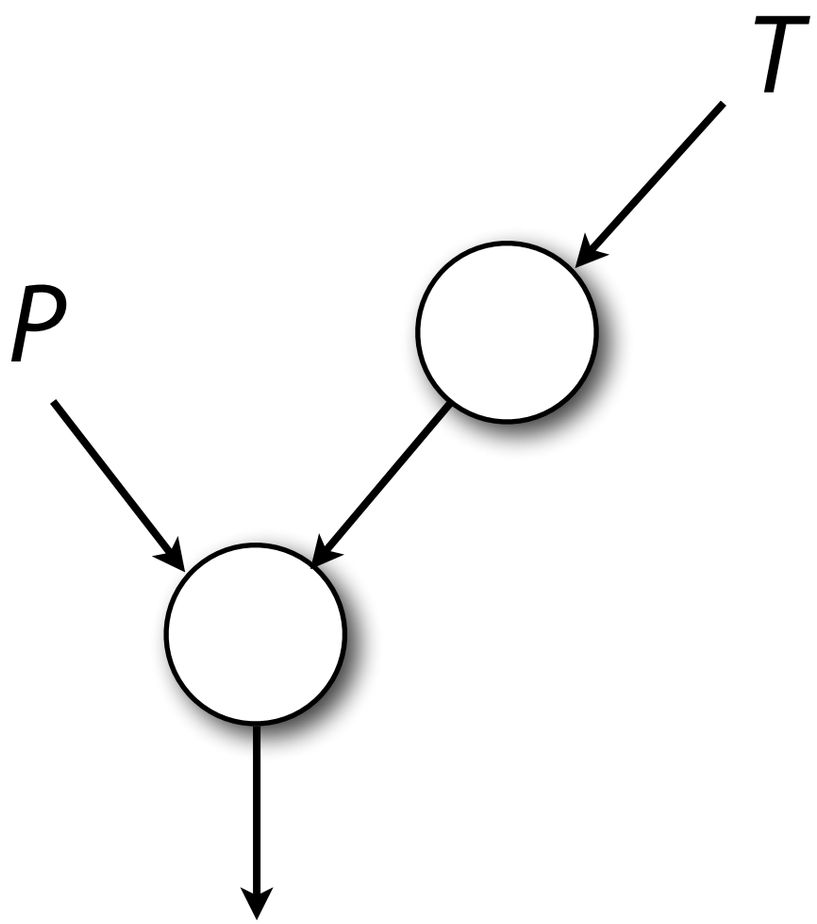
Boyer-Moore preprocessed P

What about preprocessing T?



Preprocessing

Algorithm that preprocesses *T* is *offline*.
Otherwise, algorithm is *online*.

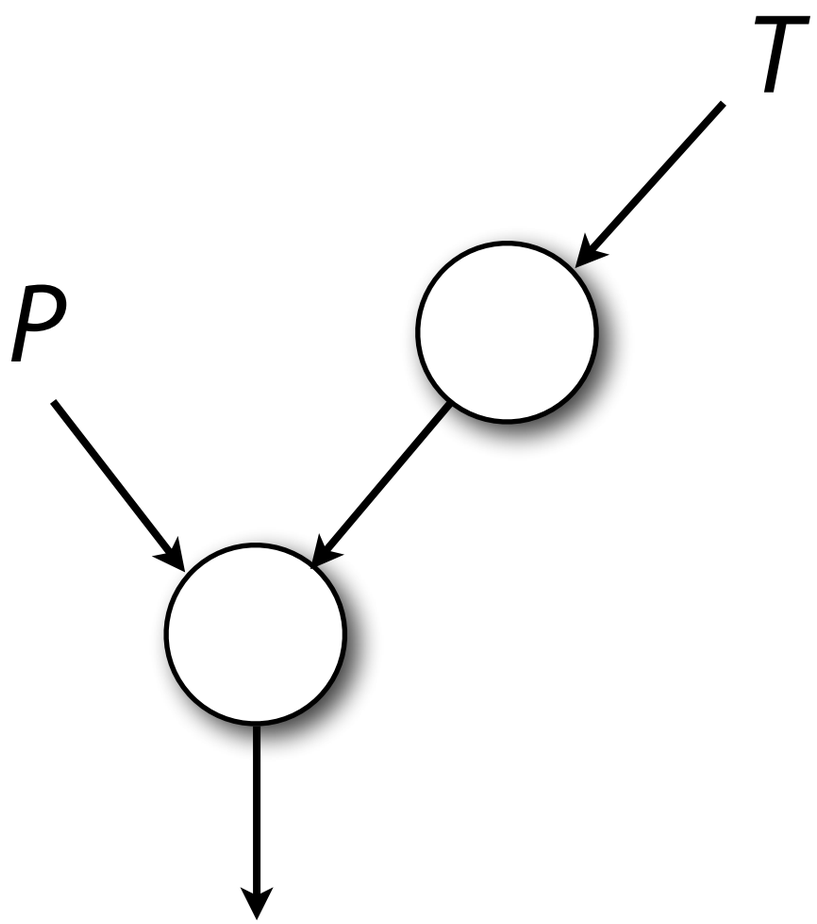


Online or offline?

- Naïve algorithm
- Boyer-Moore
- Web search engine
- Read alignment

Preprocessing

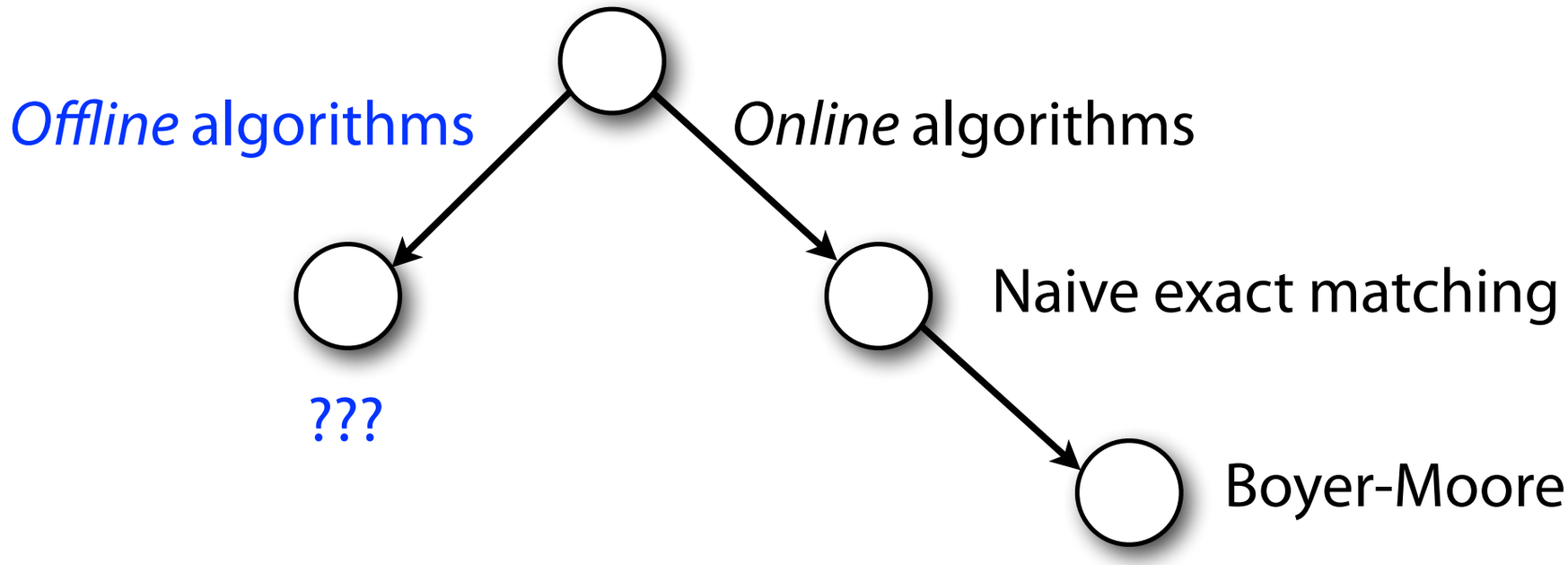
Algorithm that preprocesses *T* is *offline*.
Otherwise, algorithm is *online*.



Online or offline?

- Naïve algorithm
- Boyer-Moore
- Web search engine
- Read alignment

Offline algorithms



Still focusing on exact matching problem: find all places where pattern *P* exactly matches a substring of text *T*

Index

nest site hunting, 482–87
honeypot ants, *see Myrmecocystus*
hormones, 106–9
see also exocrine glands
house (nest site) hunting, 482–92
Hymenoptera (general), xvi
haplodiploid sex determination, 20–22
Hypoponera (ants), 194, 262, 324, 388

inclusive fitness, 20–23, 29–42
information measurement, 251–52
intercastes, 388–89
see also ergatogynes; ergatoid queens; gamergates
Iridomyrmex (ants), 266, 280, 288, 321
Isoptera, *see* termites

juvenile hormone, caste, 106–9, 372

kin recognition, 293–98
kin selection, 18–19, 23–24, 28–42, 299, 386

Macrotermes (termites), 59–60
male recognition, 298
mass communication, 62–63, 214–18
mating, multiple, 155
maze following, 119
Megalomyrmex (ants), 457
Megaponera (ants), *see Pachycondyla*
Melipona (stingless bees), 129
Melophorus (ants), repletes, 257
memory, 117–19, 213
Messor (harvester ants), 212, 232
mind, 117–19
Monomorium, 127, 212, 214, 216–17, 292
motor displays, 235–47
mound-building ants, 2
multilevel selection, 7, 7–13, 24–29
mutilation, ritual, 366–73
mutualism, *see* symbioses, ants
Myanmyrma (fossil ants), 318
Myopias (ants), 326

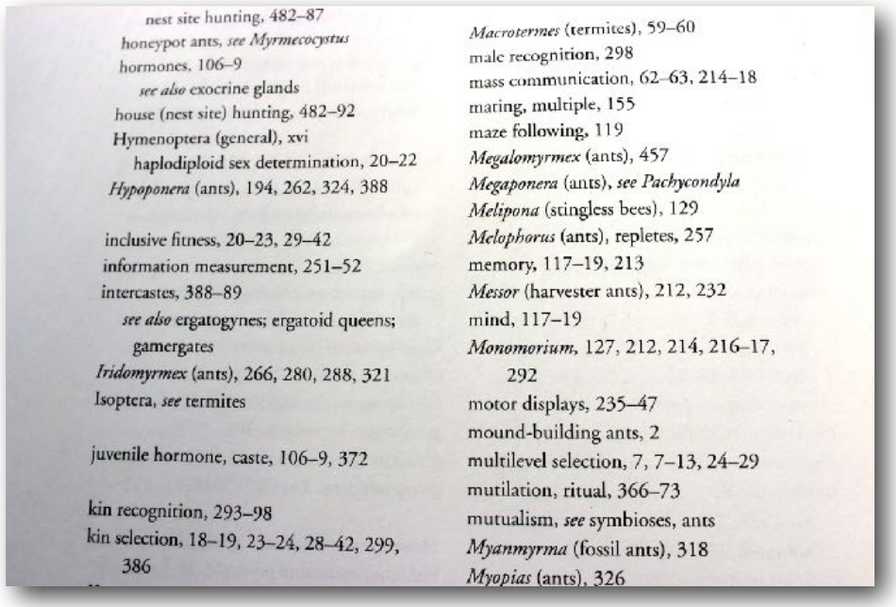
Key terms ordered alphabetically, with associated page #s

Index

Grocery store items grouped into aisles

Index

Indexes use *ordering* and *grouping* to make it easy to jump to relevant portions of the data



nest site hunting, 482–87
honeypot ants, *see Myrmecocystus*
hormones, 106–9
see also exocrine glands
house (nest site) hunting, 482–92
Hymenoptera (general), xvi
haplodiploid sex determination, 20–22
Hypoponera (ants), 194, 262, 324, 388

inclusive fitness, 20–23, 29–42
information measurement, 251–52
intercastes, 388–89
see also ergatogynes; ergatoid queens; gamergates
Iridomyrmex (ants), 266, 280, 288, 321
Isoptera, *see* termites

juvenile hormone, caste, 106–9, 372

kin recognition, 293–98
kin selection, 18–19, 23–24, 28–42, 299, 386

Macrotermes (termites), 59–60
male recognition, 298
mass communication, 62–63, 214–18
mating, multiple, 155
maze following, 119
Megalomyrmex (ants), 457
Megaponera (ants), *see Pachycondyla*
Melipona (stingless bees), 129
Melophorus (ants), repletes, 257
memory, 117–19, 213
Messor (harvester ants), 212, 232
mind, 117–19
Monomorium, 127, 212, 214, 216–17, 292
motor displays, 235–47
mound-building ants, 2
multilevel selection, 7, 7–13, 24–29
mutilation, ritual, 366–73
mutualism, *see* symbioses, ants
Myanmyrma (fossil ants), 318
Myopias (ants), 326

Indexing DNA

Index of T

T: C G T G C G T G C T T

Indexing DNA

Index of T

`CGTGC: 0`

T: `C G T G C G T G C T T`

Indexing DNA

Index of T

```
CGTGC:  0
GTGCG:  1
```

T: C G T G C G T G C T T

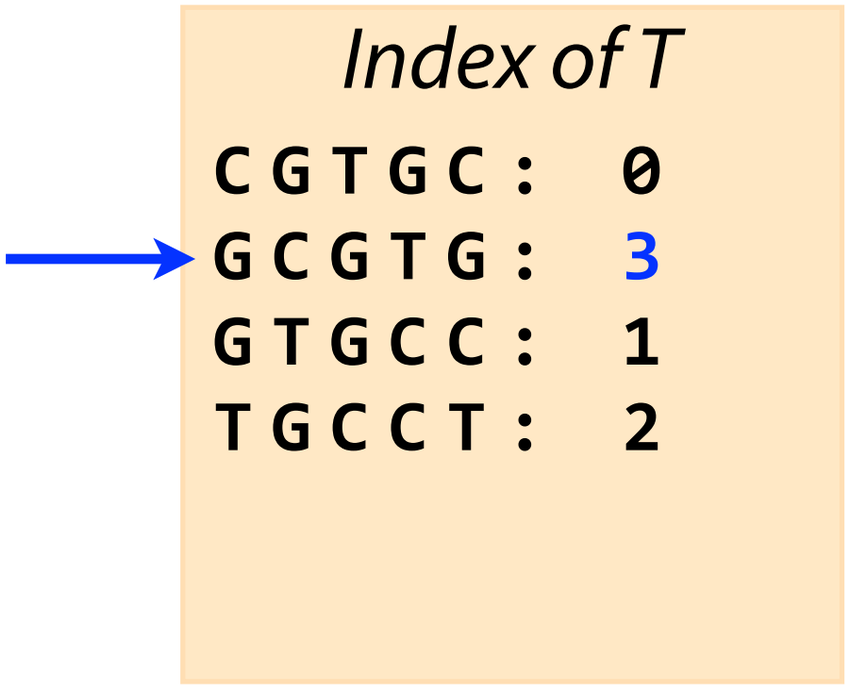
Indexing DNA

Index of T

```
CGTGC:  0
GTGCG:  1
TGCGT:  2
```

T: C G T G C G T G C T T

Indexing DNA



T: C G T G C G T G C T T

Indexing DNA

Index of T

T: C G T G C G T G C T T

Indexing DNA

Index of T

```
CGTGC:  0,4
GCGTG:  3
GTGCC:  1
GTGCT:  5
TGCCT:  2
```

T: C G T G C G T G C T T

Indexing DNA

Index of T

```
CGTGC:  0,4
GCGTG:  3
GTGCC:  1
GTGCT:  5
TGCCT:  2
TGCTT:  6
```

T: **C G T G C G T G C T T**

Indexing DNA

k-mer: substring of length k

Index of T

```
CGTGC:  0,4
GCGTG:  3
GTGCC:  1
GTGCT:  5
TGCCT:  2
TGCTT:  6
```

5-mer index

T: **C G T G C G T G C T T**

Querying the index

Index of T

```
CGTGC:  0,4
GCGTG:  3
GTGCC:  1
GTGCT:  5
TGCCT:  2
TGCTT:  6
```

T: **C G T G C G T G C T T**

P: **G C G T G C**

Querying the index

Index of T

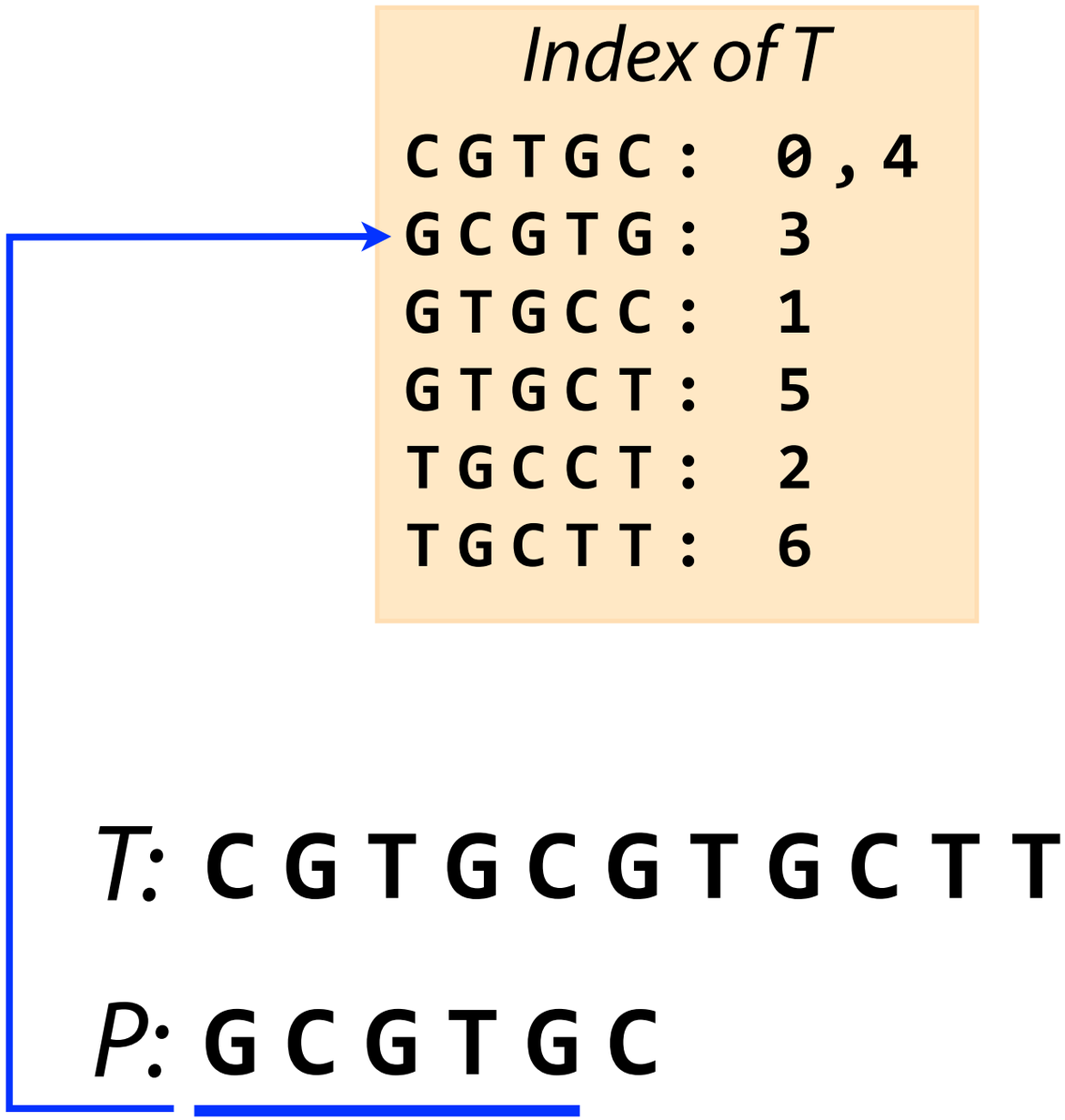
```
CGTGC:  0,4
GCGTG:  3
GTGCC:  1
GTGCT:  5
TGCCT:  2
TGCTT:  6
```

?

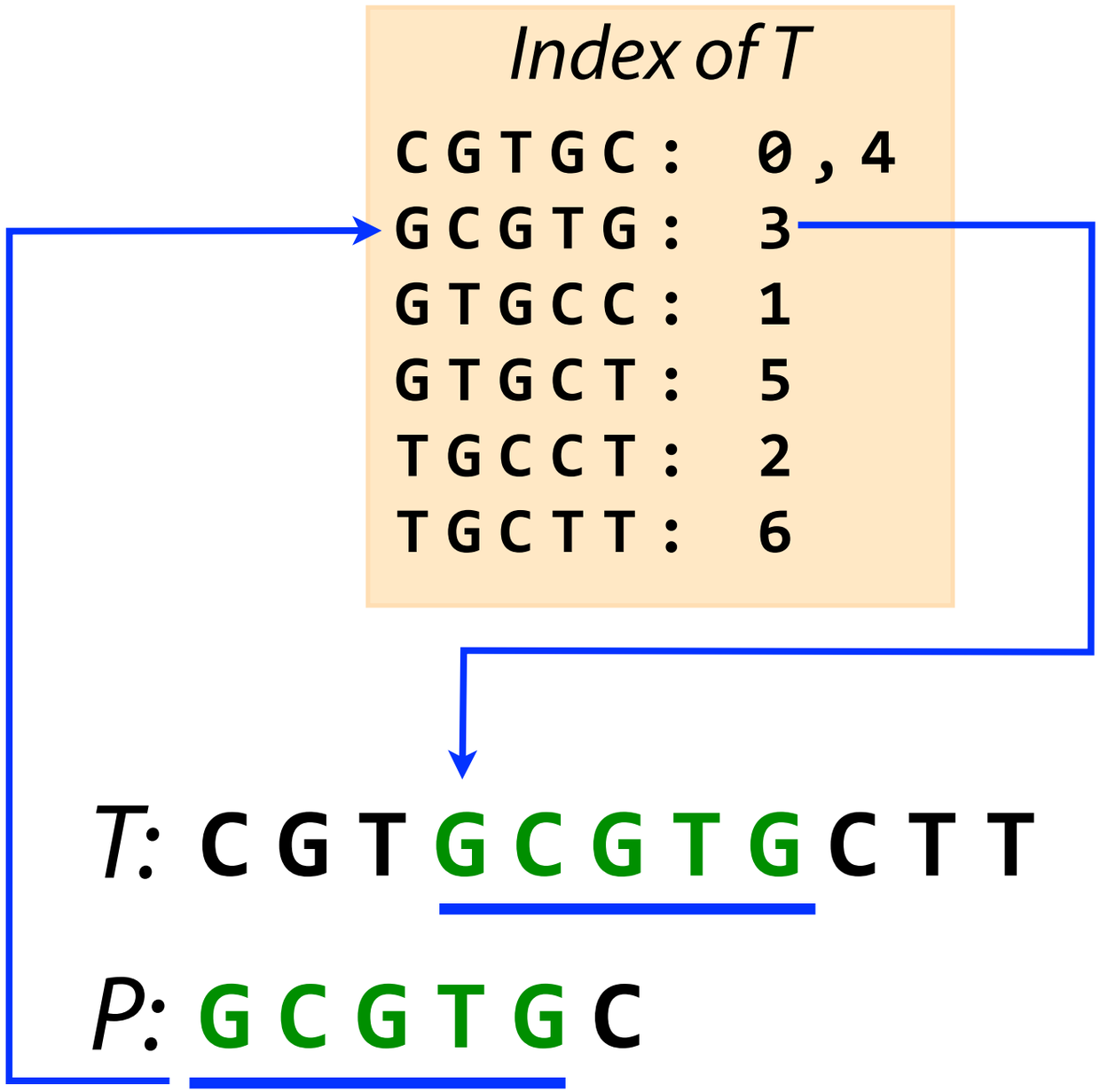
T: C G T G C G T G C T T

P: G C G T G C

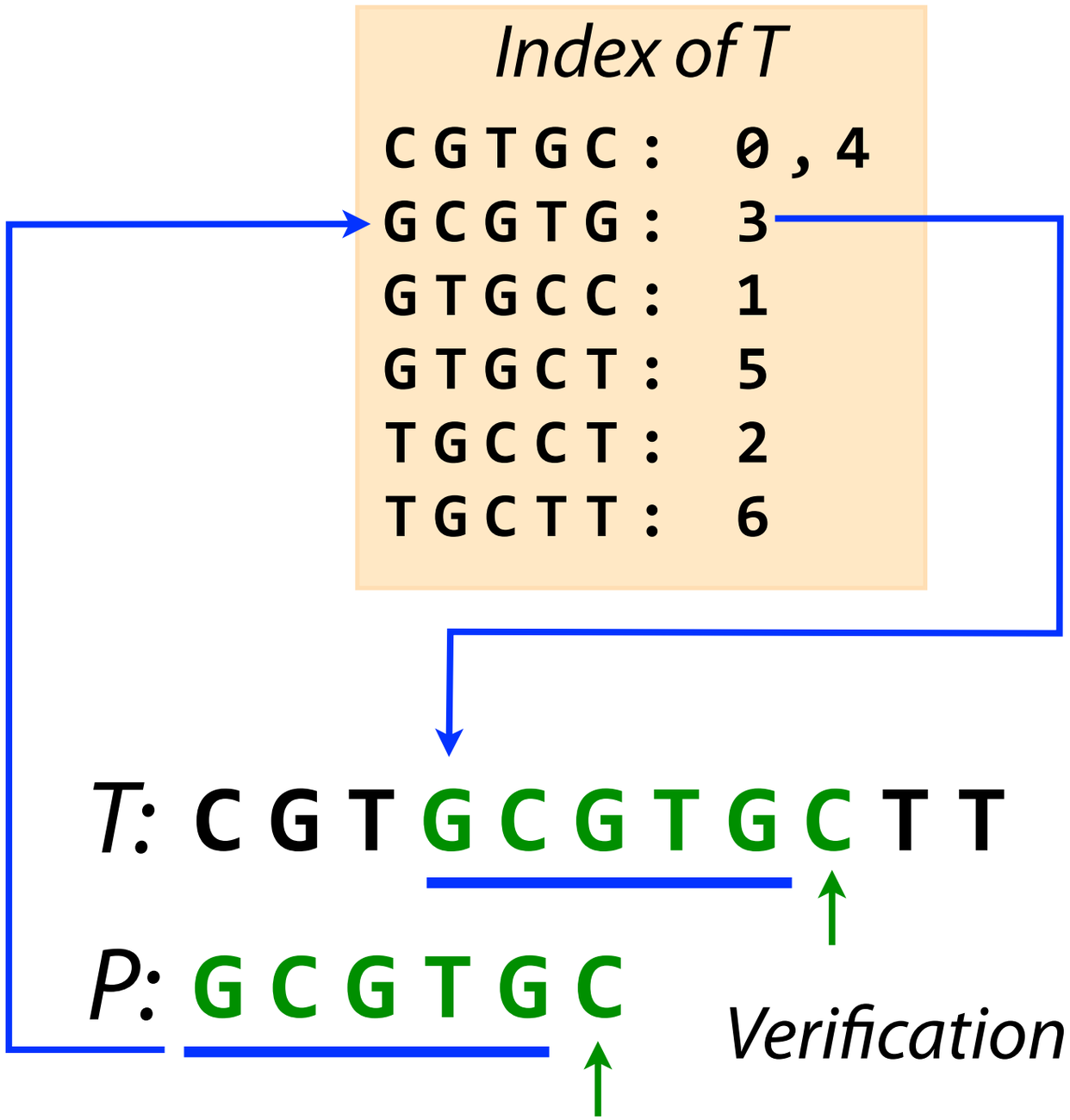
Querying the index



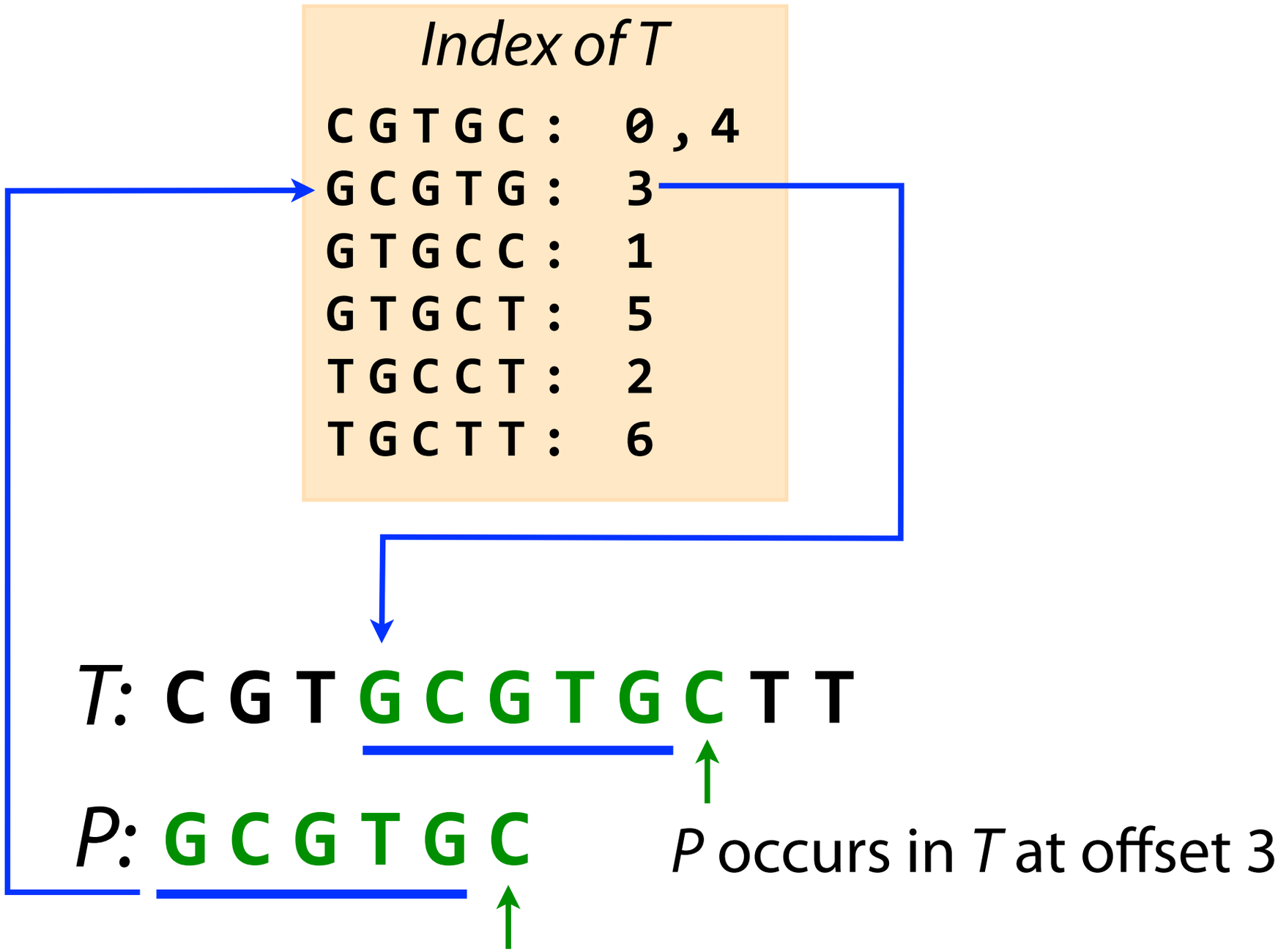
Querying the index



Querying the index



Querying the index



Querying the index

Index of T

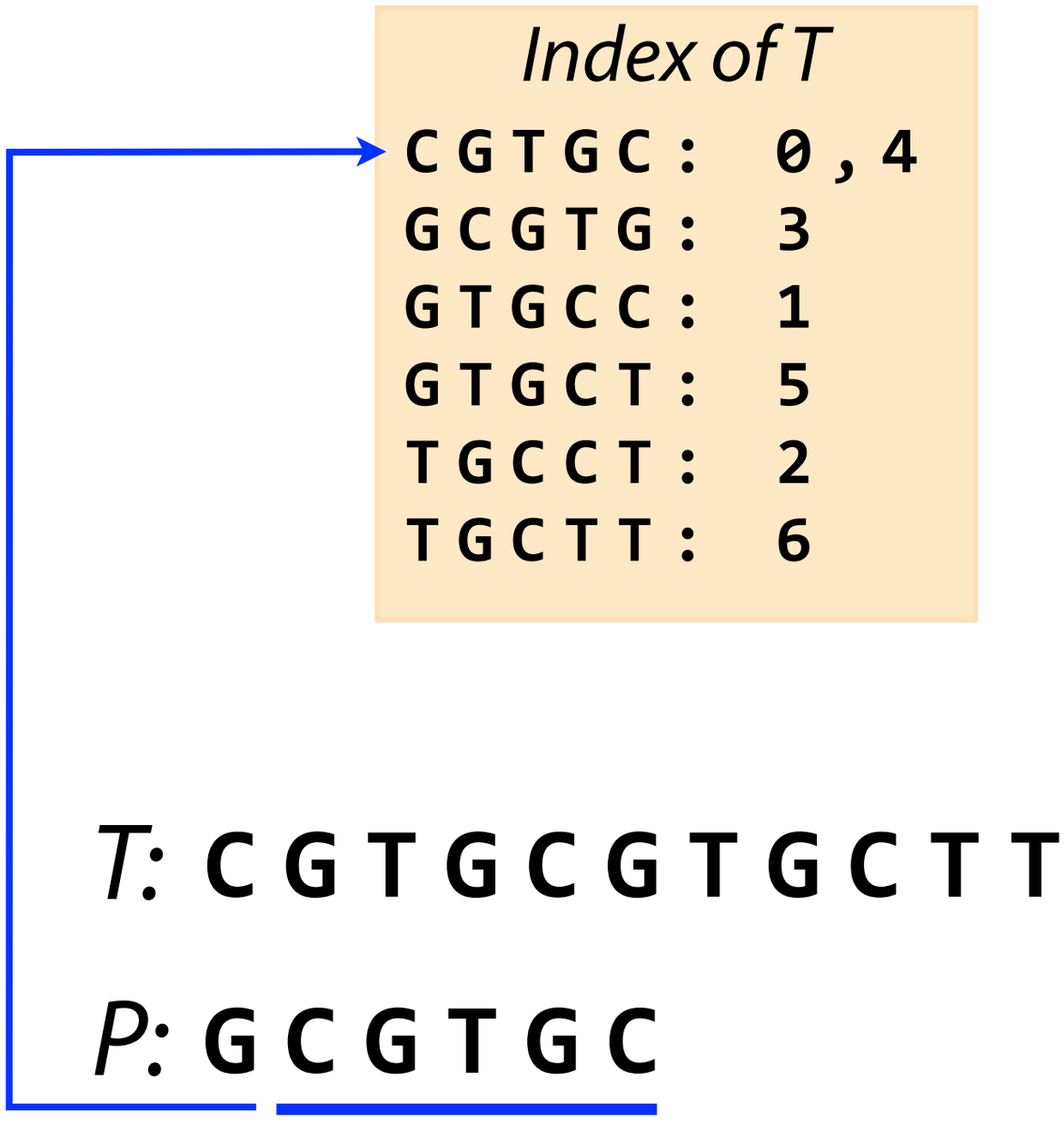
```
CGTGC:  0,4
GCGTG:  3
GTGCC:  1
GTGCT:  5
TGCCT:  2
TGCTT:  6
```

?

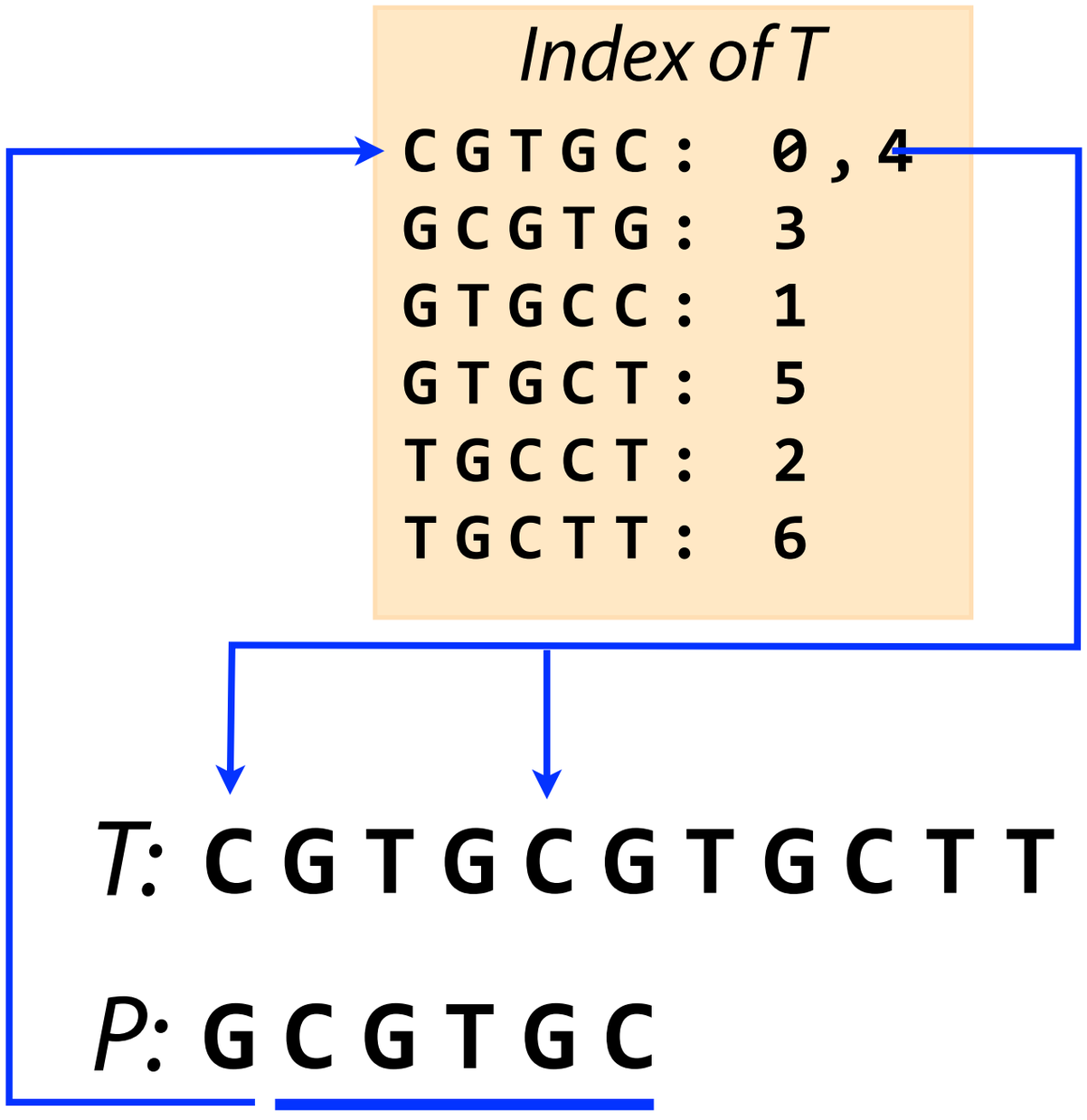
T: **C G T G C G T G C T T**

P: **G C G T G C**

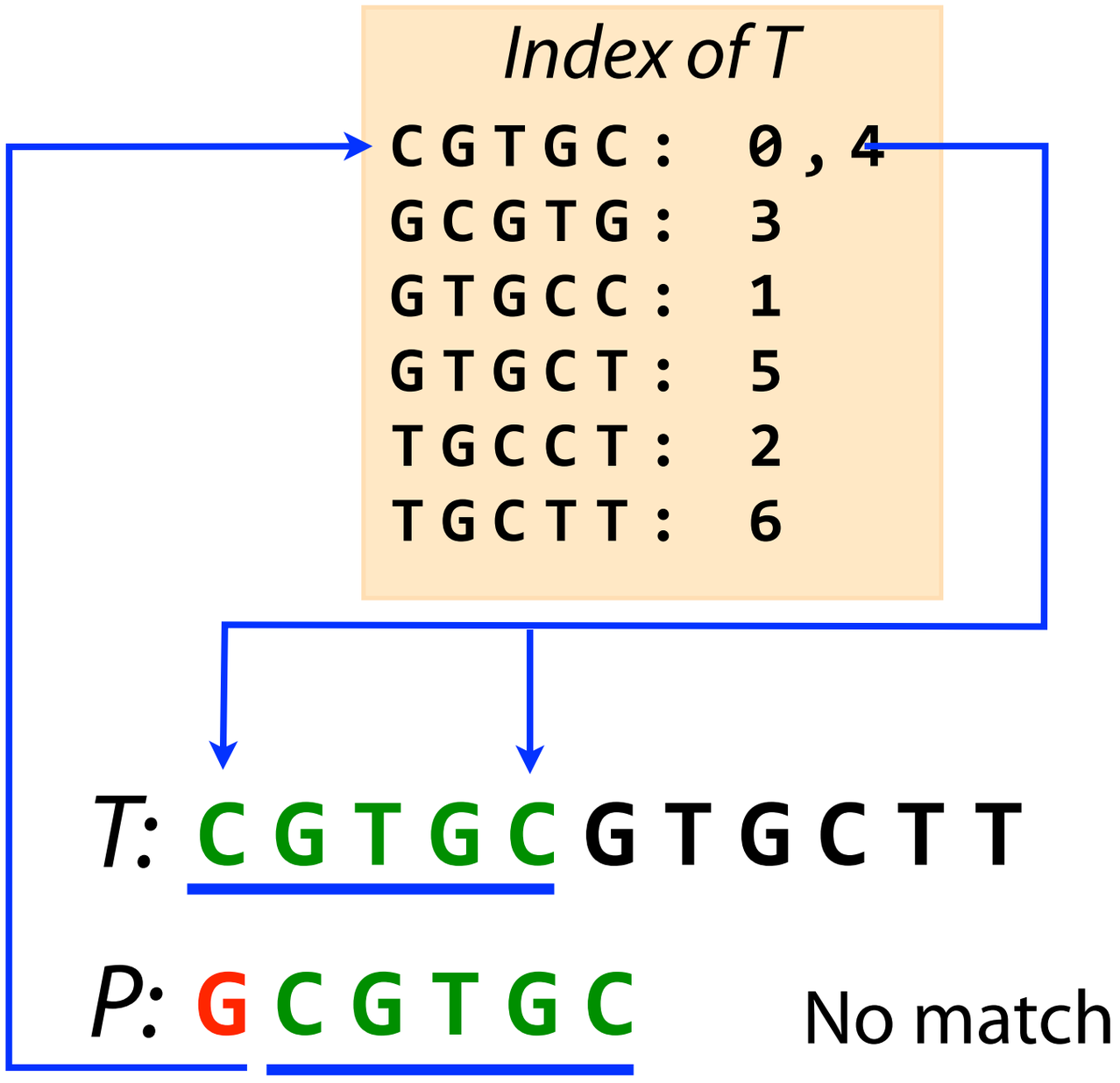
Querying the index



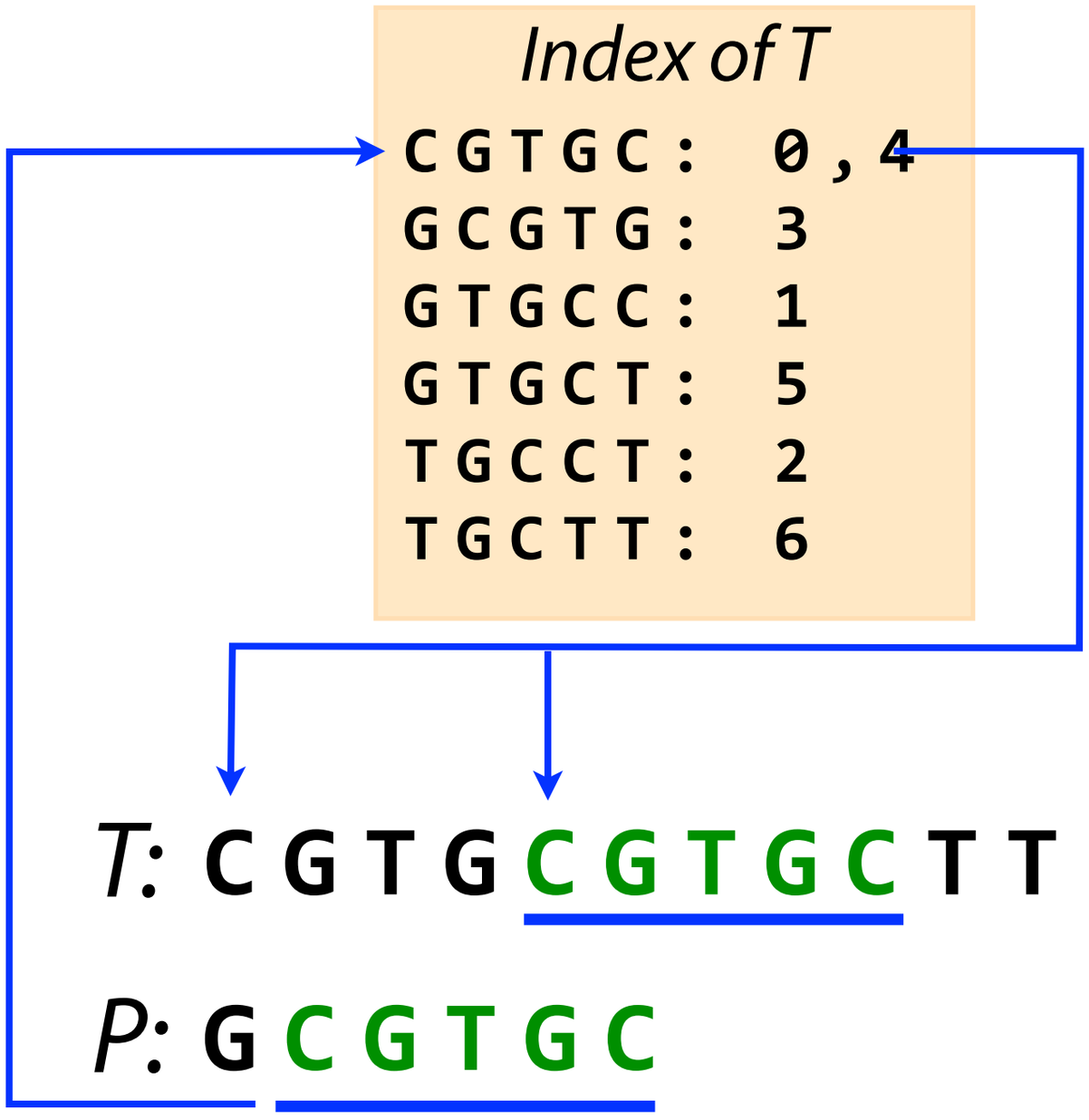
Querying the index



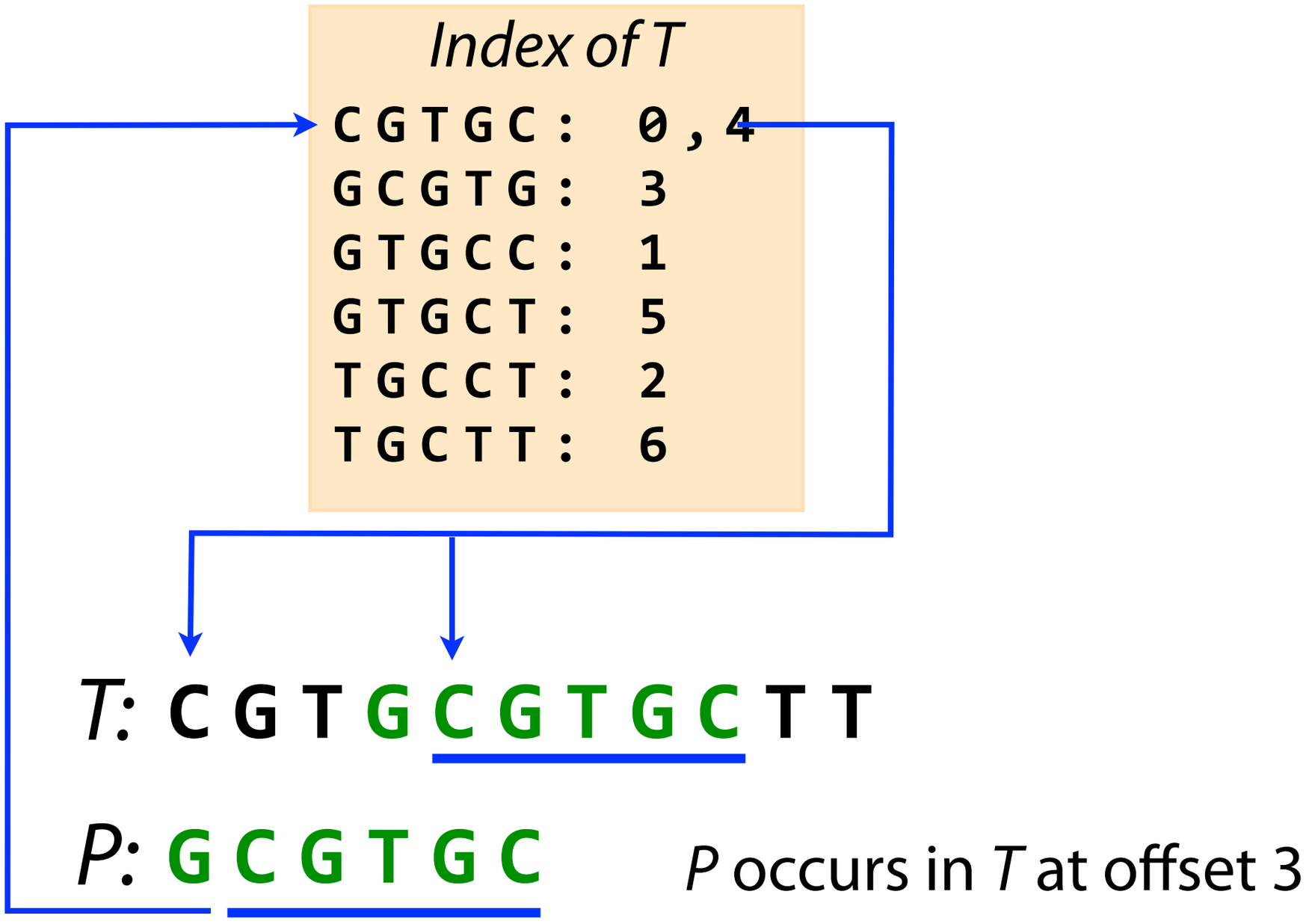
Querying the index



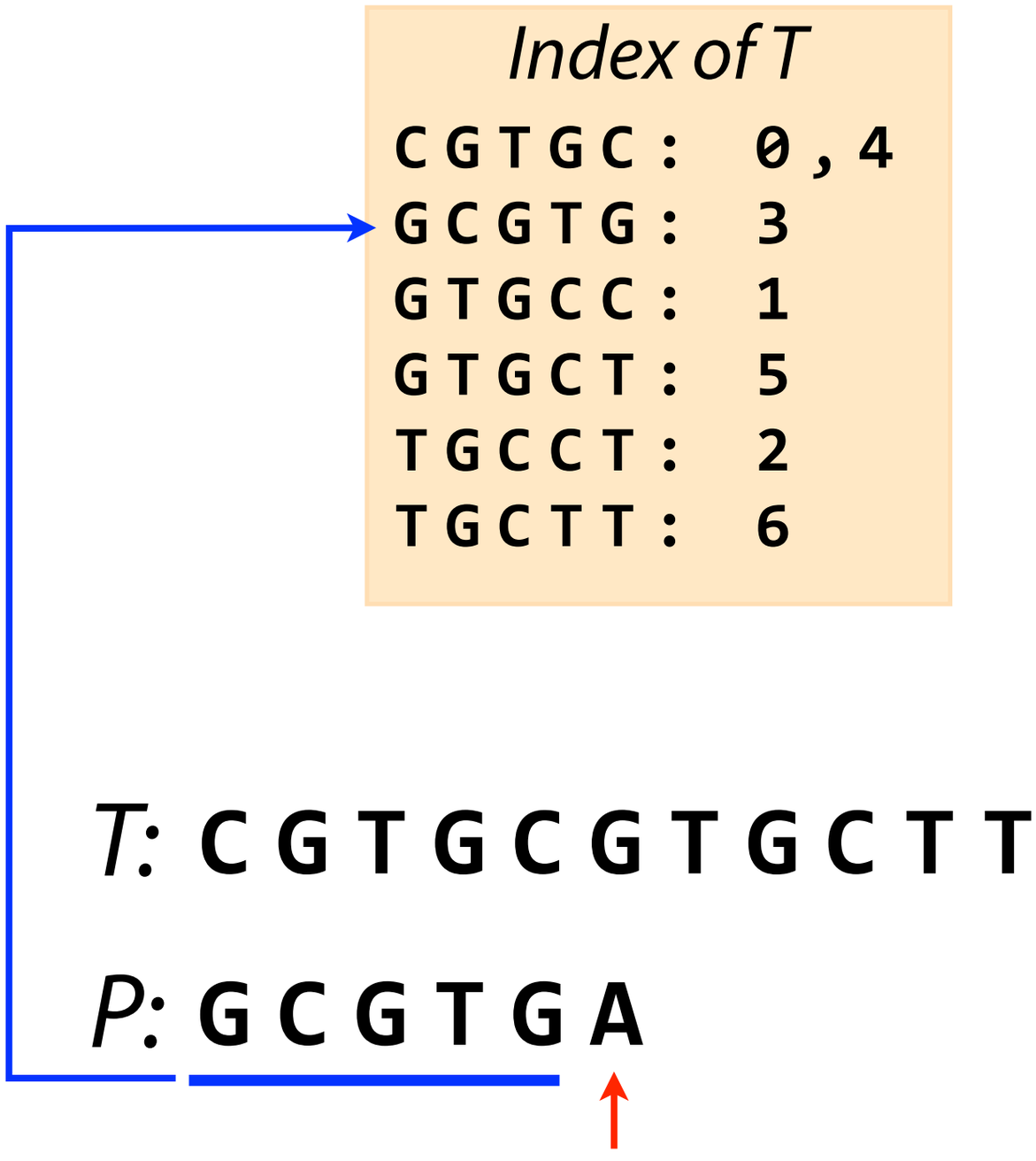
Querying the index



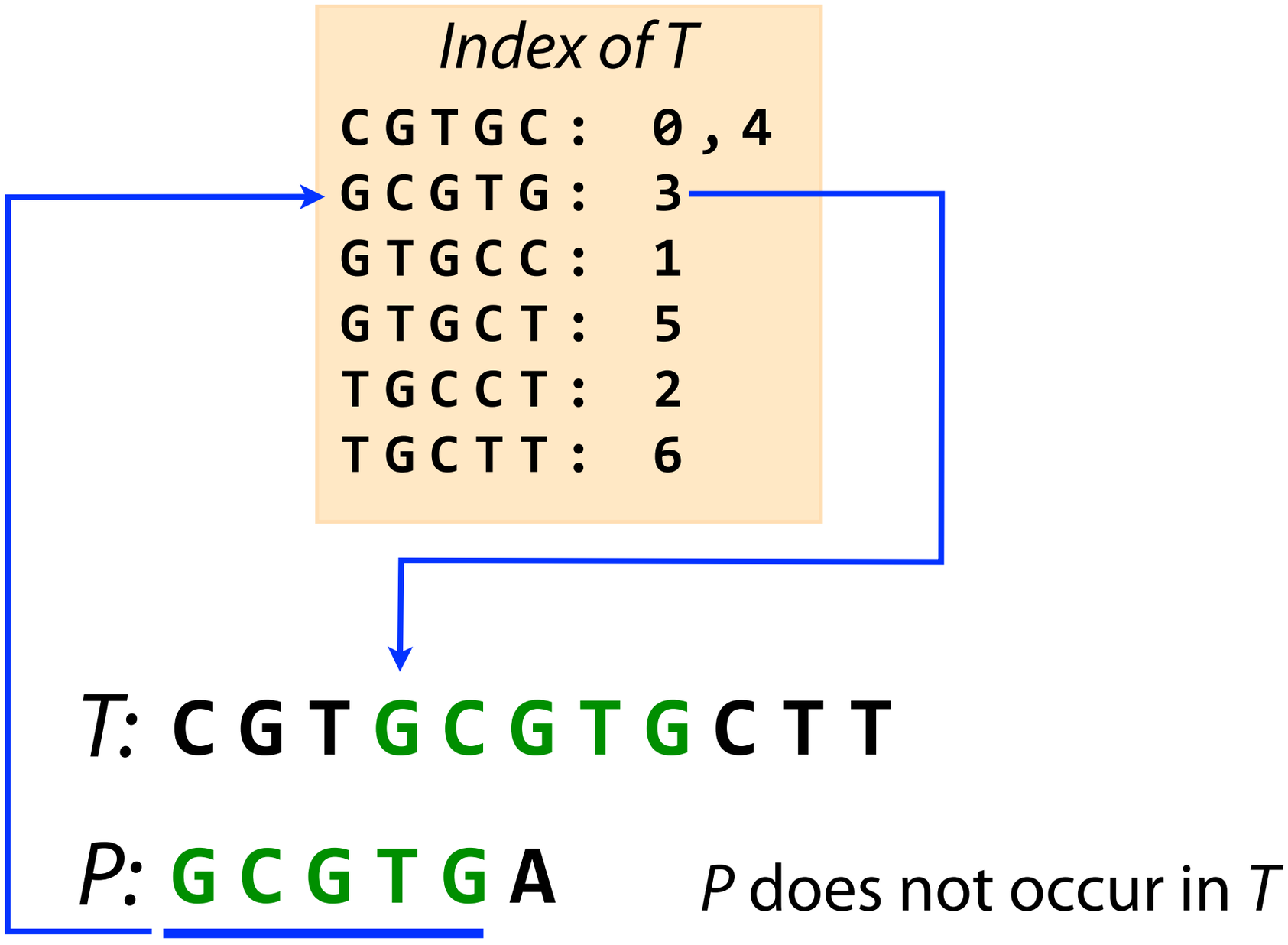
Querying the index



Querying the index



Querying the index



Querying the index

Index of T

```
CGTGC:  0,4
GCGTG:  3
GTGCC:  1
GTGCT:  5
TGCCT:  2
TGCTT:  6
```

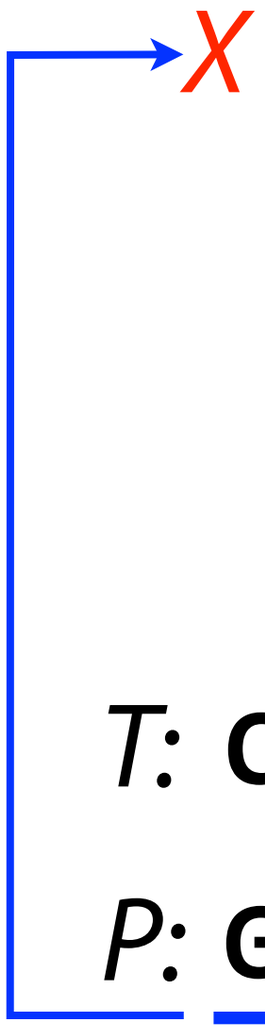
T: C G T G C G T G C T T

P: G C G T A C

Querying the index

Index of T

```
CGTGC:  0,4
GCGTG:  3
GTGCC:  1
GTGCT:  5
TGCCT:  2
TGCTT:  6
```



T: **C G T G C G T G C T T**

P: **G C G T A C**

P does not occur in *T*

Querying the index

Index of T

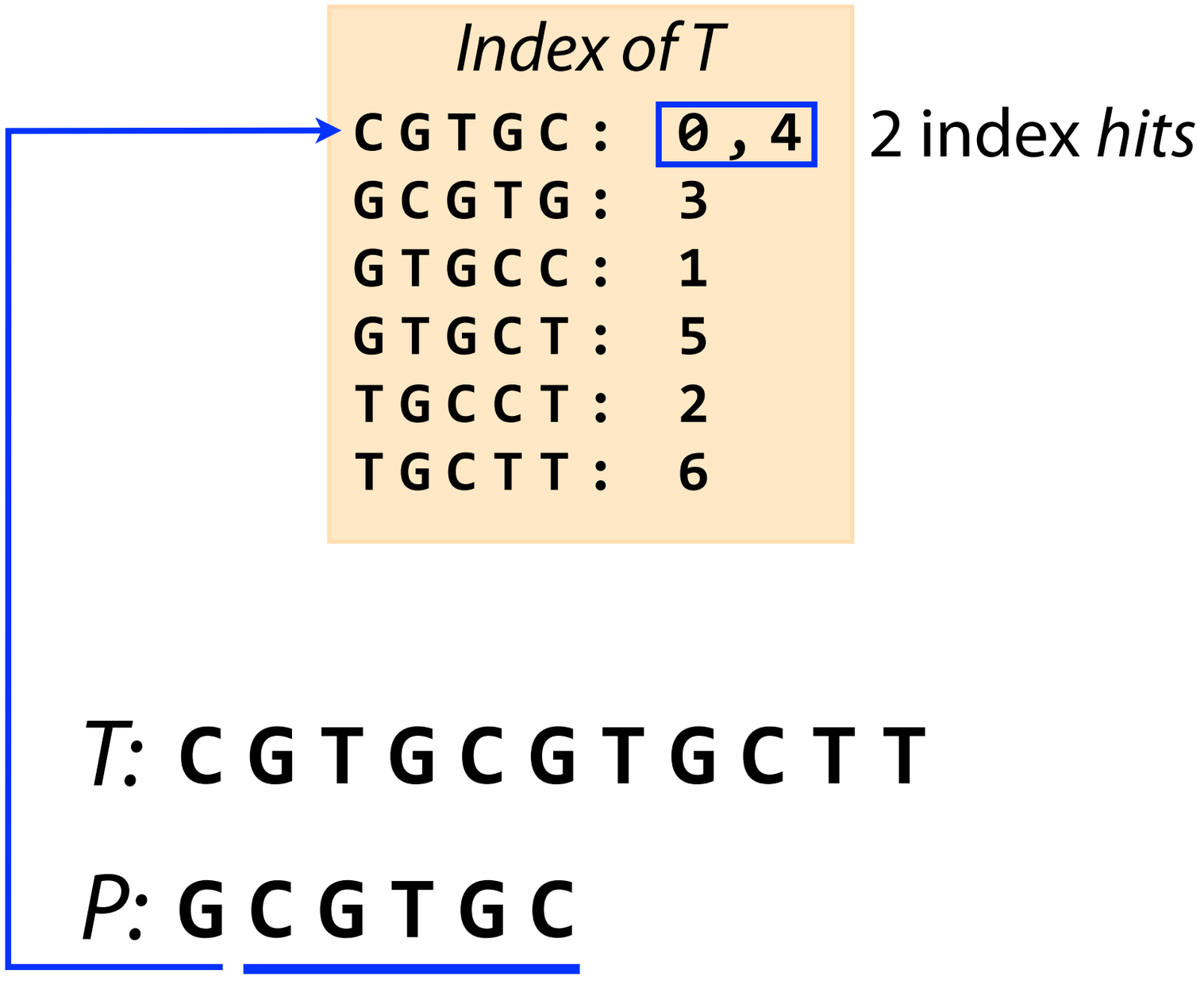
```
CGTGC:  0,4
GCGTG:  3
GTGCC:  1
GTGCT:  5
TGCCT:  2
TGCTT:  6
```

1 index *hit*

T: C G T G C G T G C T T

P: G C G T G C

Querying the index



Data structures

Index of T

```
CGTGC:  0,4
GCGTG:  3
GTGCC:  1
GTGCT:  5
TGCCT:  2
TGCTT:  6
```

Abstractly, index is a *multimap* associating keys (k-mers) with one or more values (offsets)

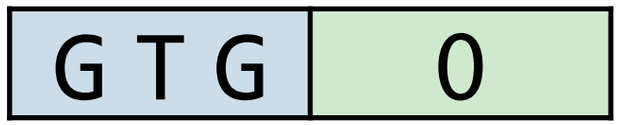
What data structures allow us to represent and query a multimap?

Data structures

First idea: add key-value pairs to an array & sort the array

T: **G T G C G T G T G G G G**

Data structures



T: G T G C G T G T G G G G

Data structures

G T G	0
T G C	1

T: G T G C G T G T G G G G

Data structures

G T G	0
T G C	1
G C G	2

T: G T G C G T G T G G G G

Data structures

G T G	0
T G C	1
G C G	2
C G T	3
G T G	4
T G T	5
G T G	6
T G G	7
G G G	8
G G G	9
G G G	10

T: **G T G C G T G T G G G G**

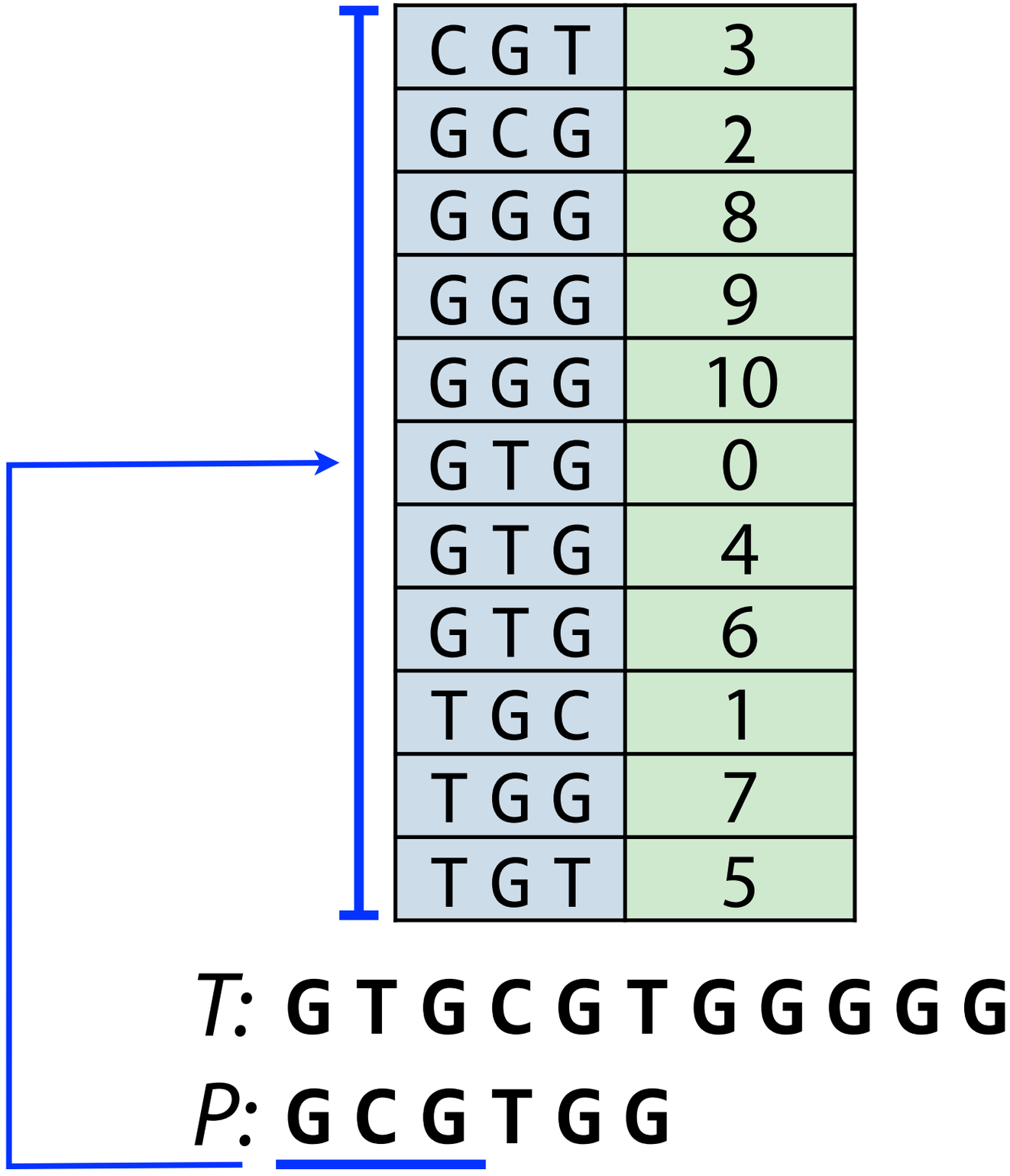
Data structures

Alphabetical by k-mer

C G T	3
G C G	2
G G G	8
G G G	9
G G G	10
G T G	0
G T G	4
G T G	6
T G C	1
T G G	7
T G T	5

T: **G T G C G T G T G G G G**

Binary search



Binary search

C G T	3
G C G	2
G G G	8
G G G	9
G G G	10
G T G	0
G T G	4
G T G	6
T G C	1
T G G	7
T G T	5

G C G < G T G

1st bisection

T: **G T G C G T G G G G G**

P: **G C G T G G**

Binary search

G C G < G G G

C G T	3
G C G	2
G G G	8
G G G	9
G G G	10
G T G	0
G T G	4
G T G	6
T G C	1
T G G	7
T G T	5

2nd bisection

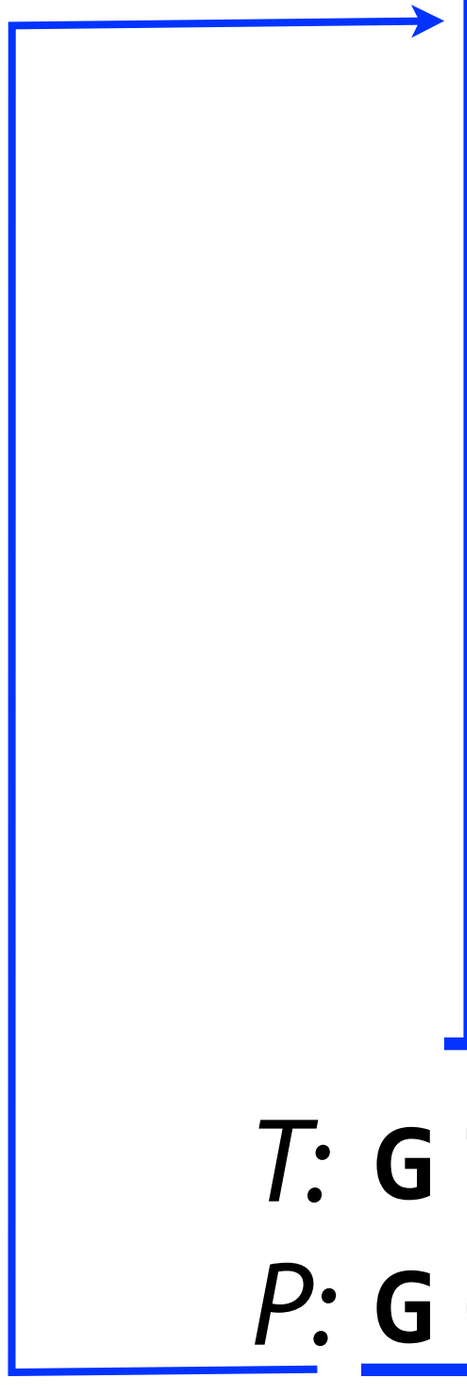
T: **G T G C G T G G G G G**

P: **G C G T G G**

Binary search

G C G = G G G

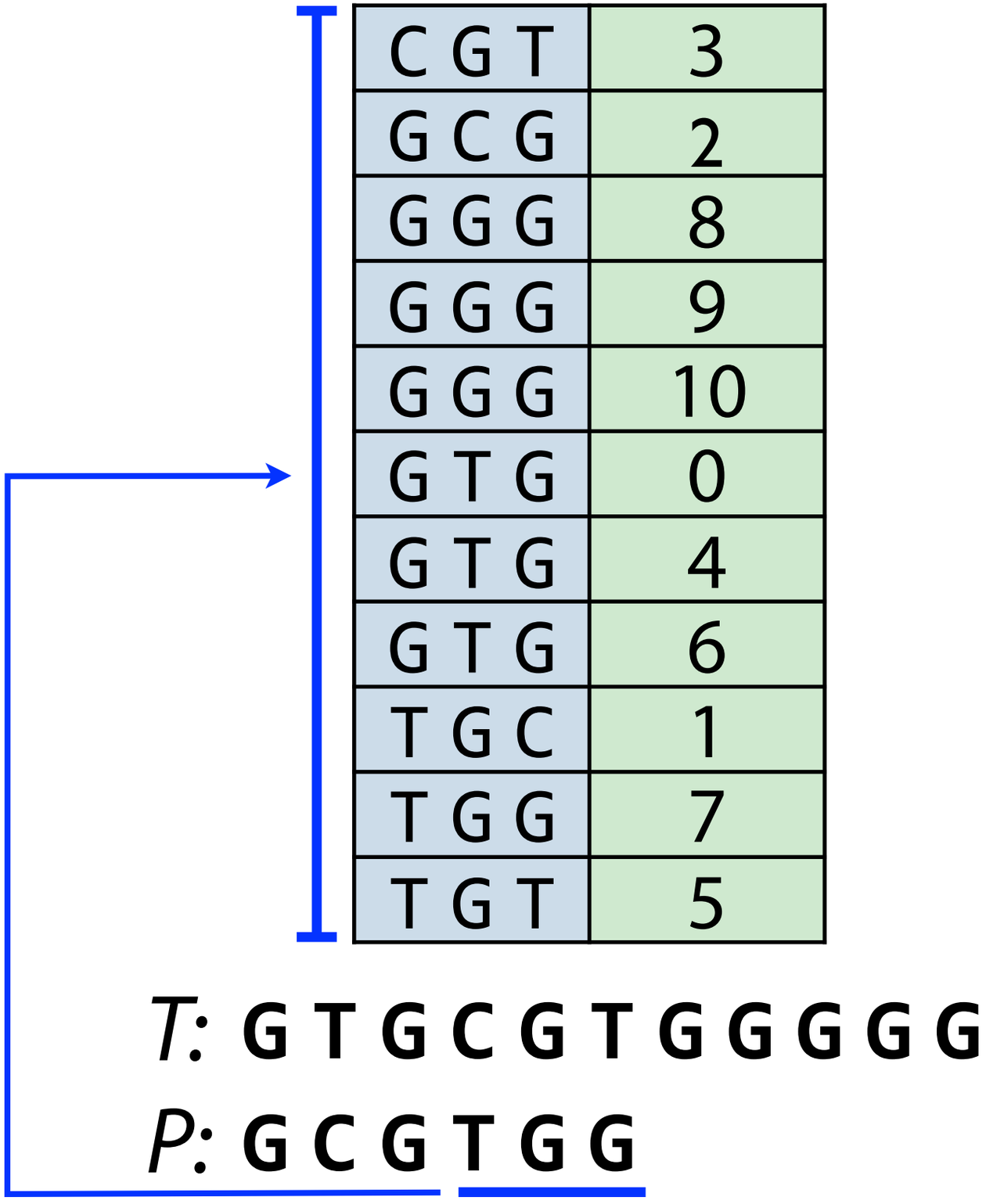
C G T	3
G C G	2
G G G	8
G G G	9
G G G	10
G T G	0
G T G	4
G T G	6
T G C	1
T G G	7
T G T	5



T: **G T G C G T G G G G G**

P: **G C G T G G**

Binary search



C G T	3
G C G	2
G G G	8
G G G	9
G G G	10
G T G	0
G T G	4
G T G	6
T G C	1
T G G	7
T G T	5

T: **G T G C G T G G G G G**

P: **G C G T G G**

Binary search

T G G > G T G

C G T	3
G C G	2
G G G	8
G G G	9
G G G	10
G T G	0
G T G	4
G T G	6
T G C	1
T G G	7
T G T	5

T: **G T G C G T G G G G G**

P: **G C G T G G**

Binary search

After 1st bisection

C G T	3
G C G	2
G G G	8
G G G	9
G G G	10
G T G	0
G T G	4
G T G	6
T G C	1
T G G	7
T G T	5

T G G > T G C

T: **G T G C G T G G G G G**

P: **G C G T G G**

Binary search

After 2nd bisection

C G T	3
G C G	2
G G G	8
G G G	9
G G G	10
G T G	0
G T G	4
G T G	6
T G C	1
T G G	7
T G T	5

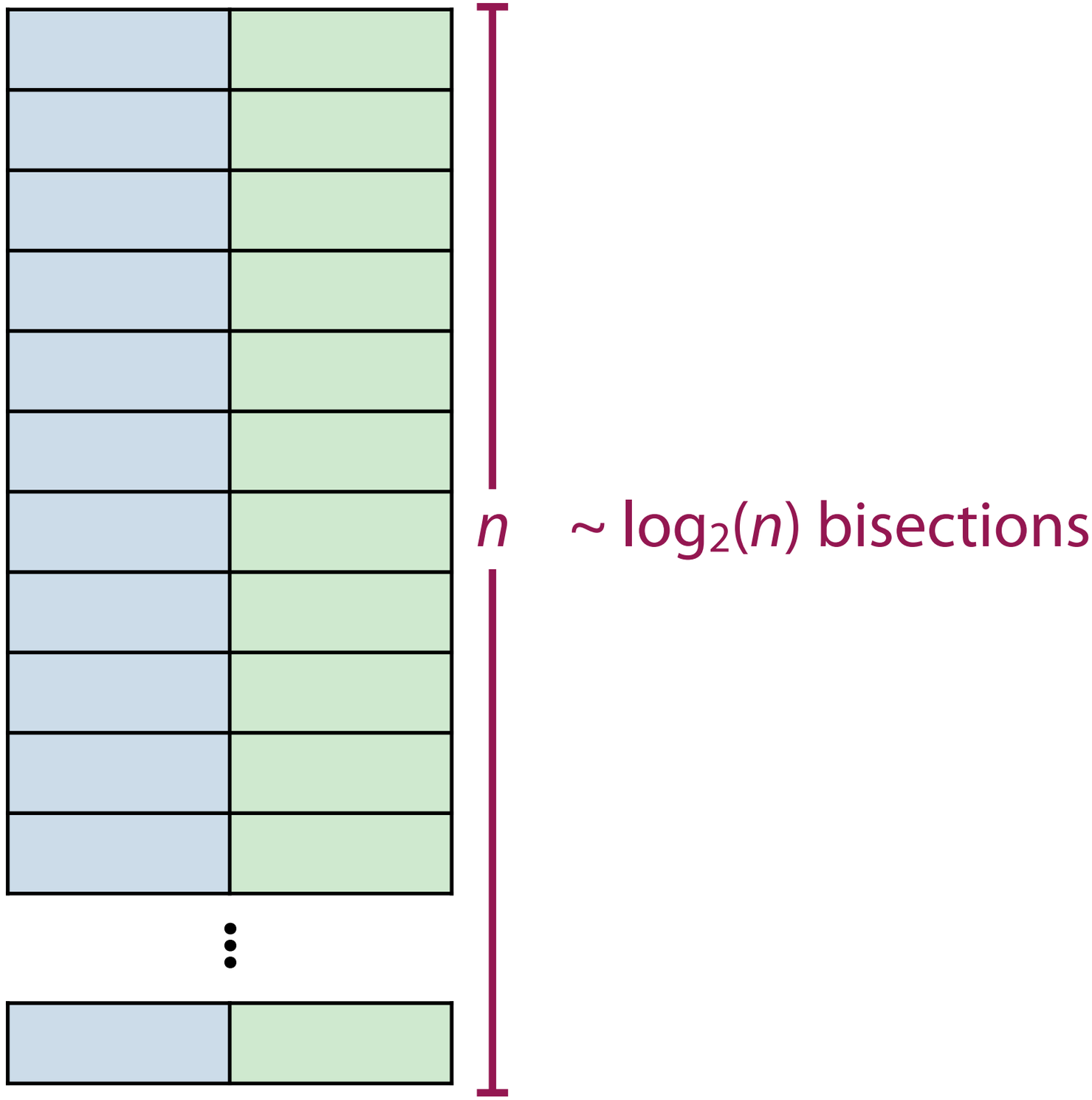
T G G = T G G

T: **G T G C G T G G G G G**

P: **G C G T G G**

Binary search

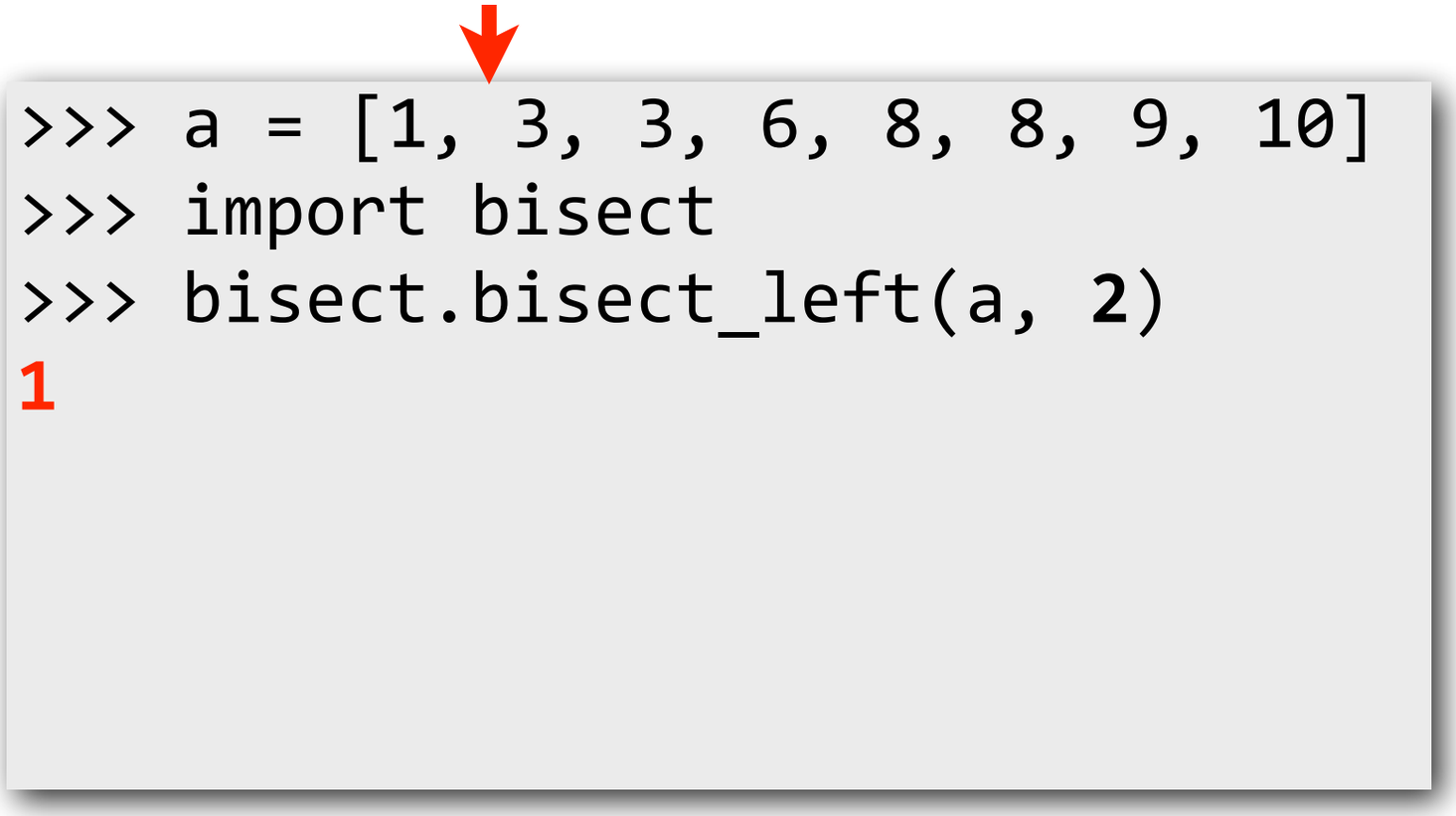
About how many bisections in the worst case, as a function of *n*?



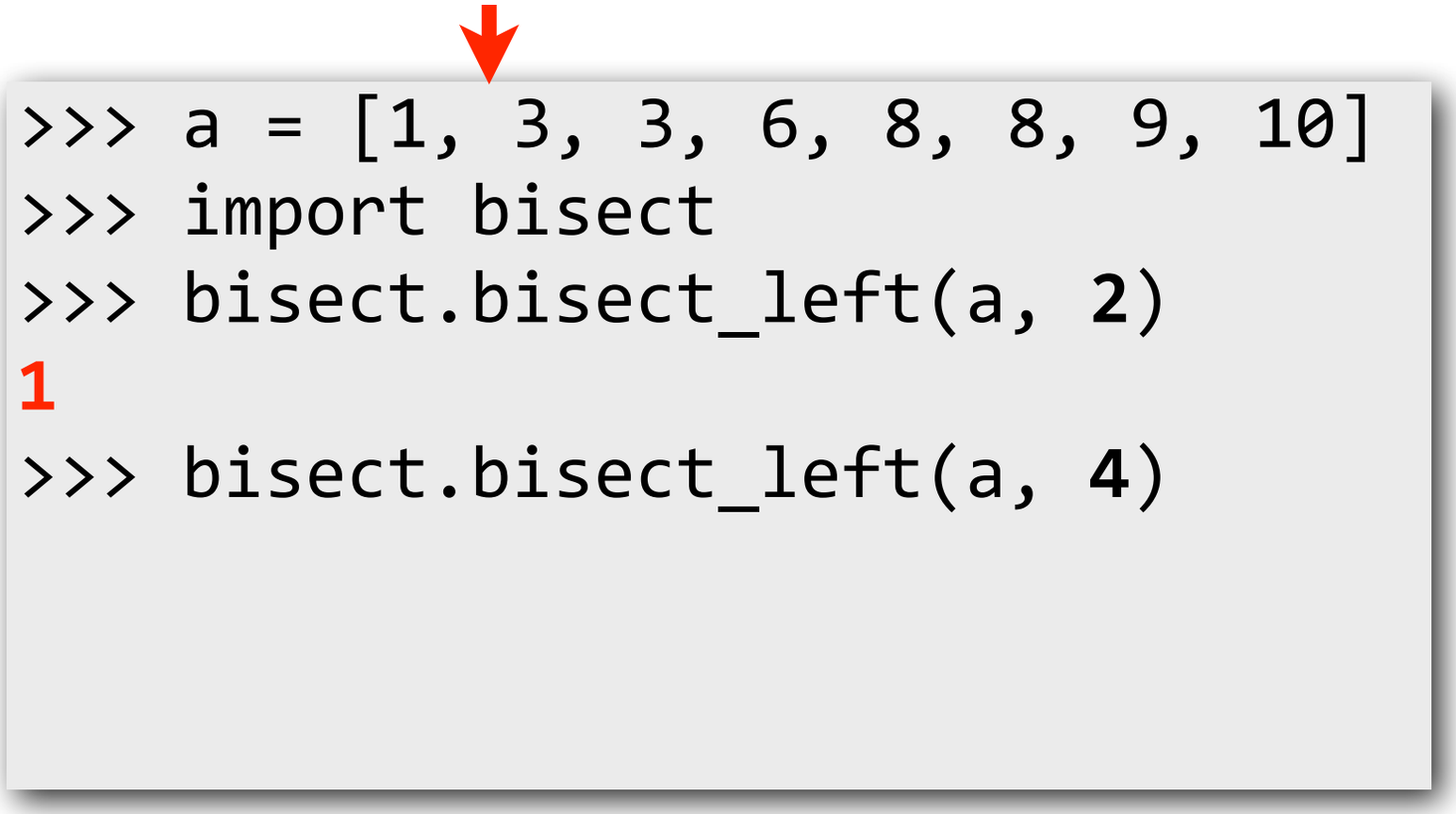
`bisect.bisect_left(a, x)`: Leftmost offset where **`x`** can be inserted into **`a`** to maintain order

```
>>> a = [1, 3, 3, 6, 8, 8, 9, 10]
>>> import bisect
>>> bisect.bisect_left(a, 2)
```

`bisect.bisect_left(a, x)`: Leftmost offset where **`x`** can be inserted into **`a`** to maintain order

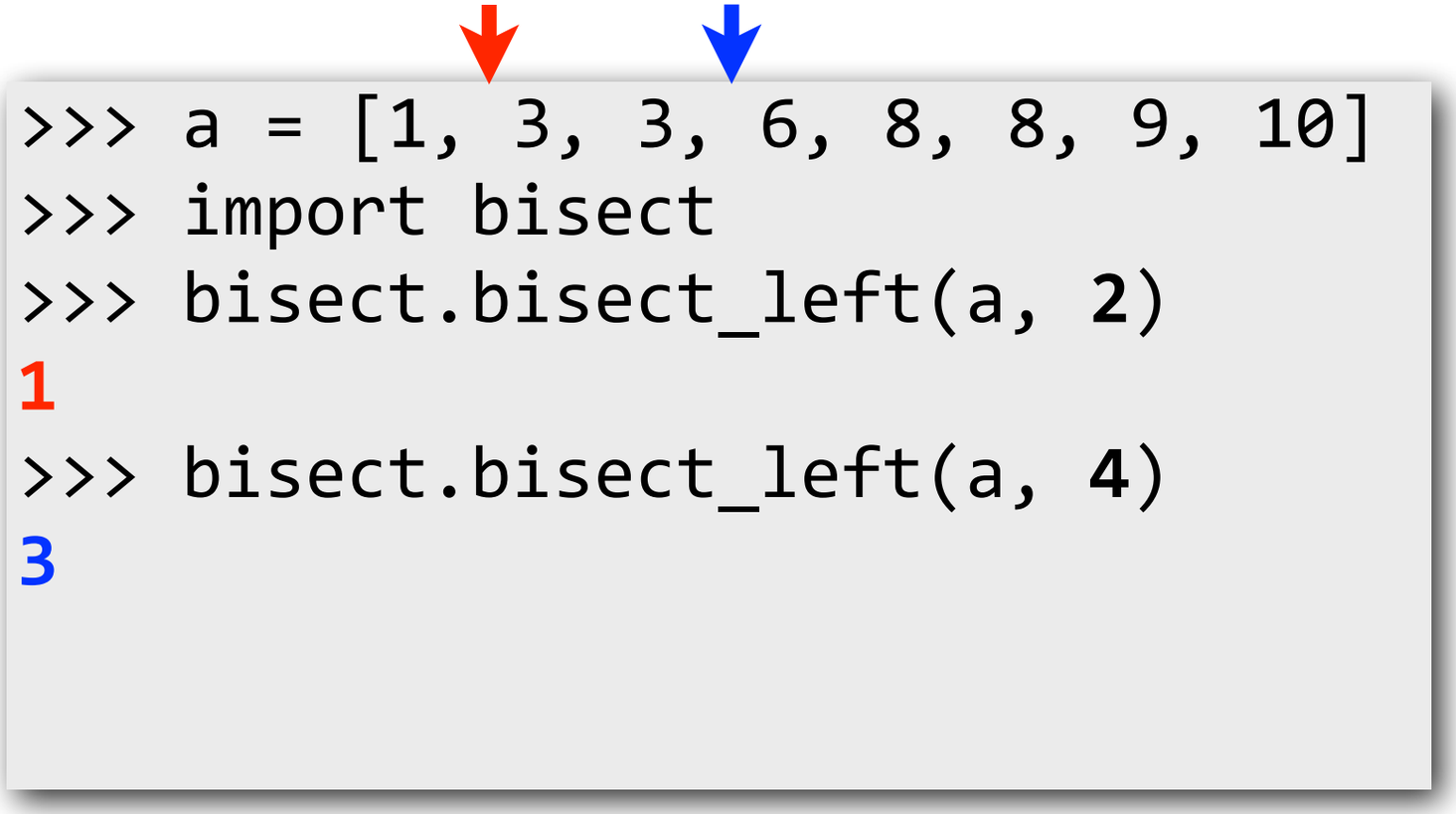


`bisect.bisect_left(a, x)`: Leftmost offset where **`x`** can be inserted into **`a`** to maintain order

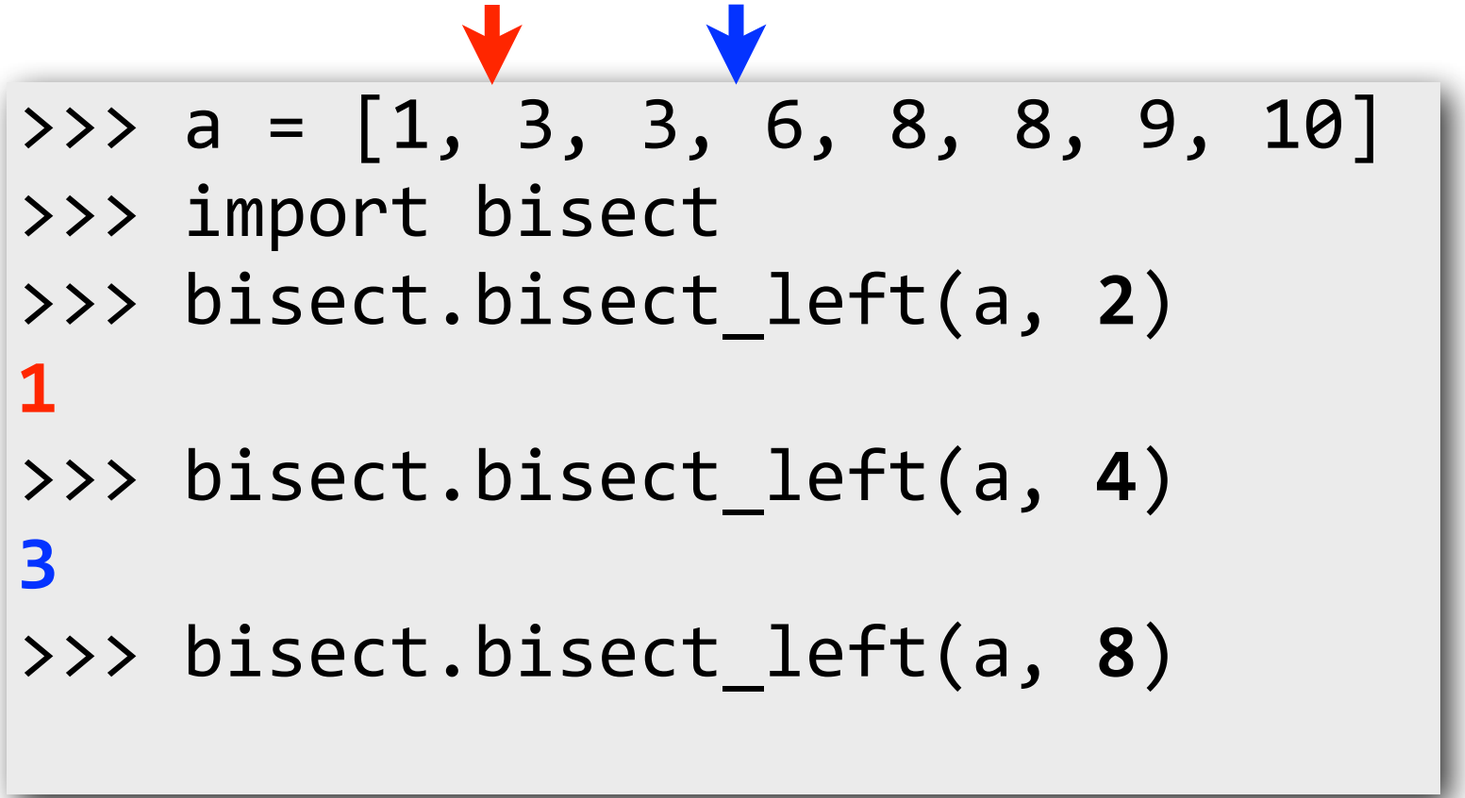


```
>>> a = [1, 3, 3, 6, 8, 8, 9, 10]
>>> import bisect
>>> bisect.bisect_left(a, 2)
1
>>> bisect.bisect_left(a, 4)
```

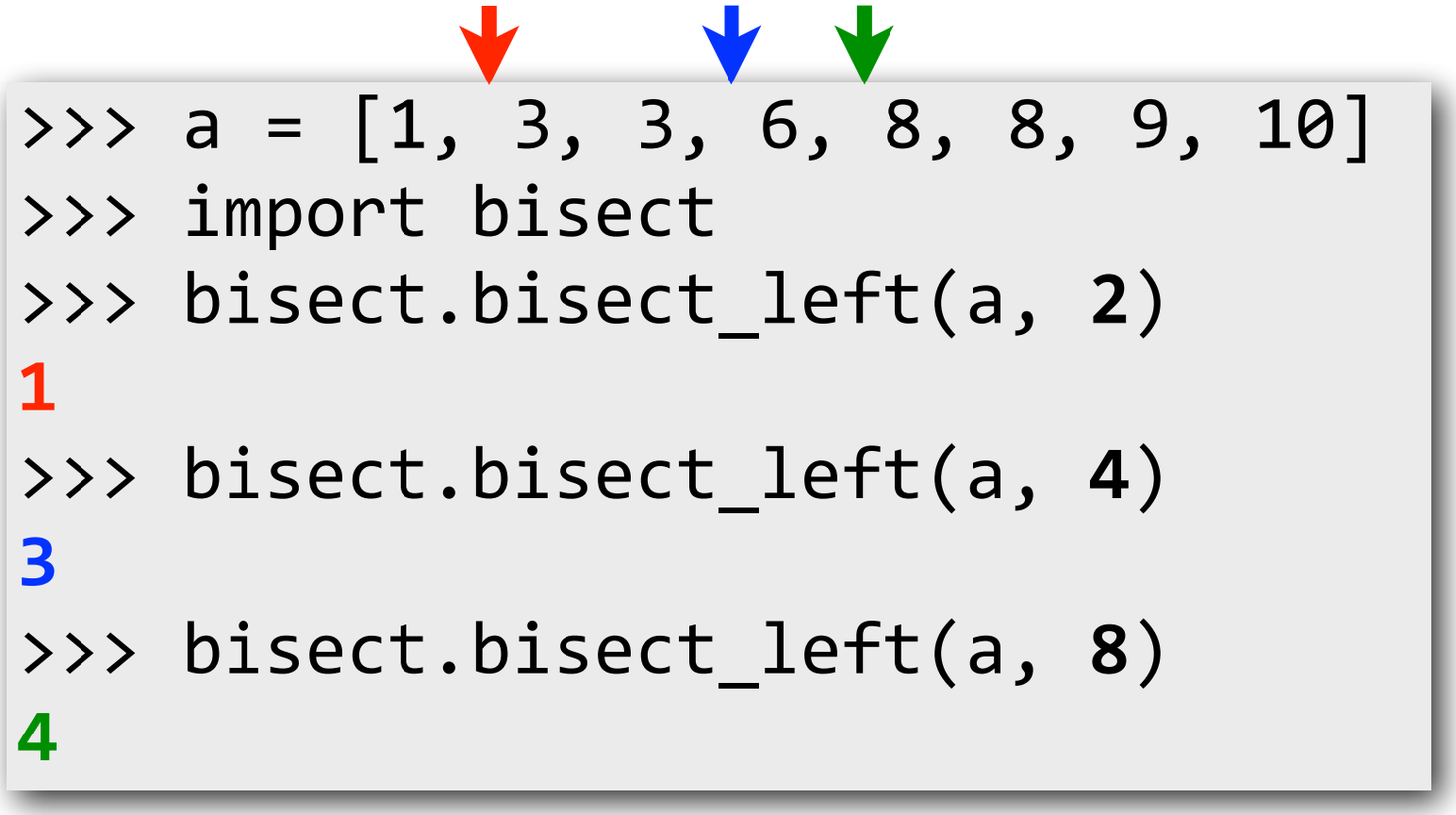
`bisect.bisect_left(a, x)`: Leftmost offset where **`x`** can be inserted into **`a`** to maintain order



`bisect.bisect_left(a, x)`: Leftmost offset where **`x`** can be inserted into **`a`** to maintain order



`bisect.bisect_left(a, x)`: Leftmost offset where **`x`** can be inserted into **`a`** to maintain order



`a` must already be sorted, or answer will be nonsense

```
bisect_left(index, 'GTG')
```

C G T	3
G C G	2
G G G	8
G G G	9
G G G	10
G T G	0
G T G	4
G T G	6
T G C	1
T G G	7
T G T	5

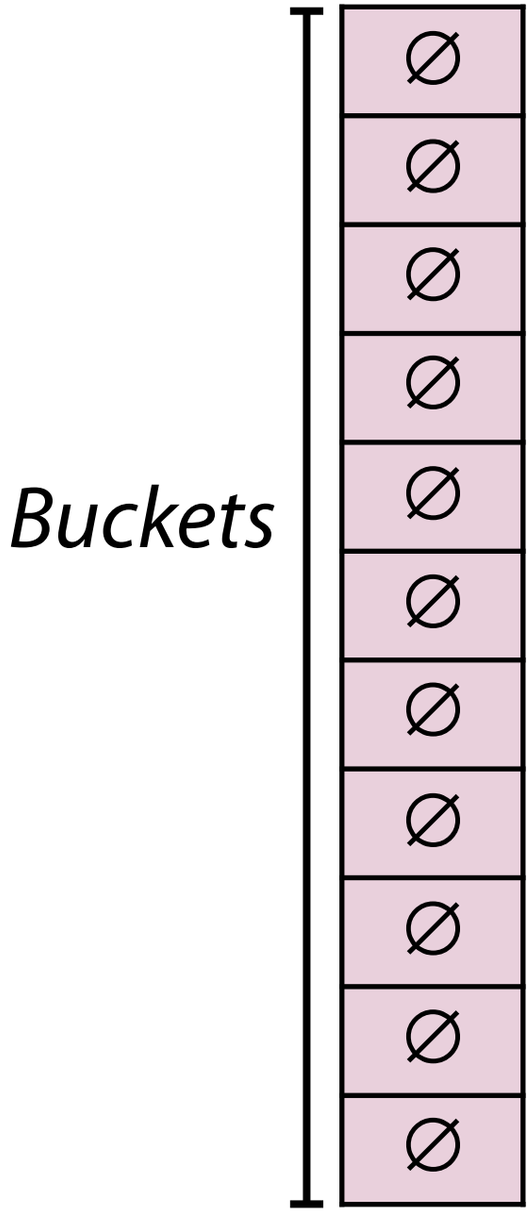
T: **G T G C G T G G G G G**

P: **G C G T G G**

Binary-search k-mer index implementation

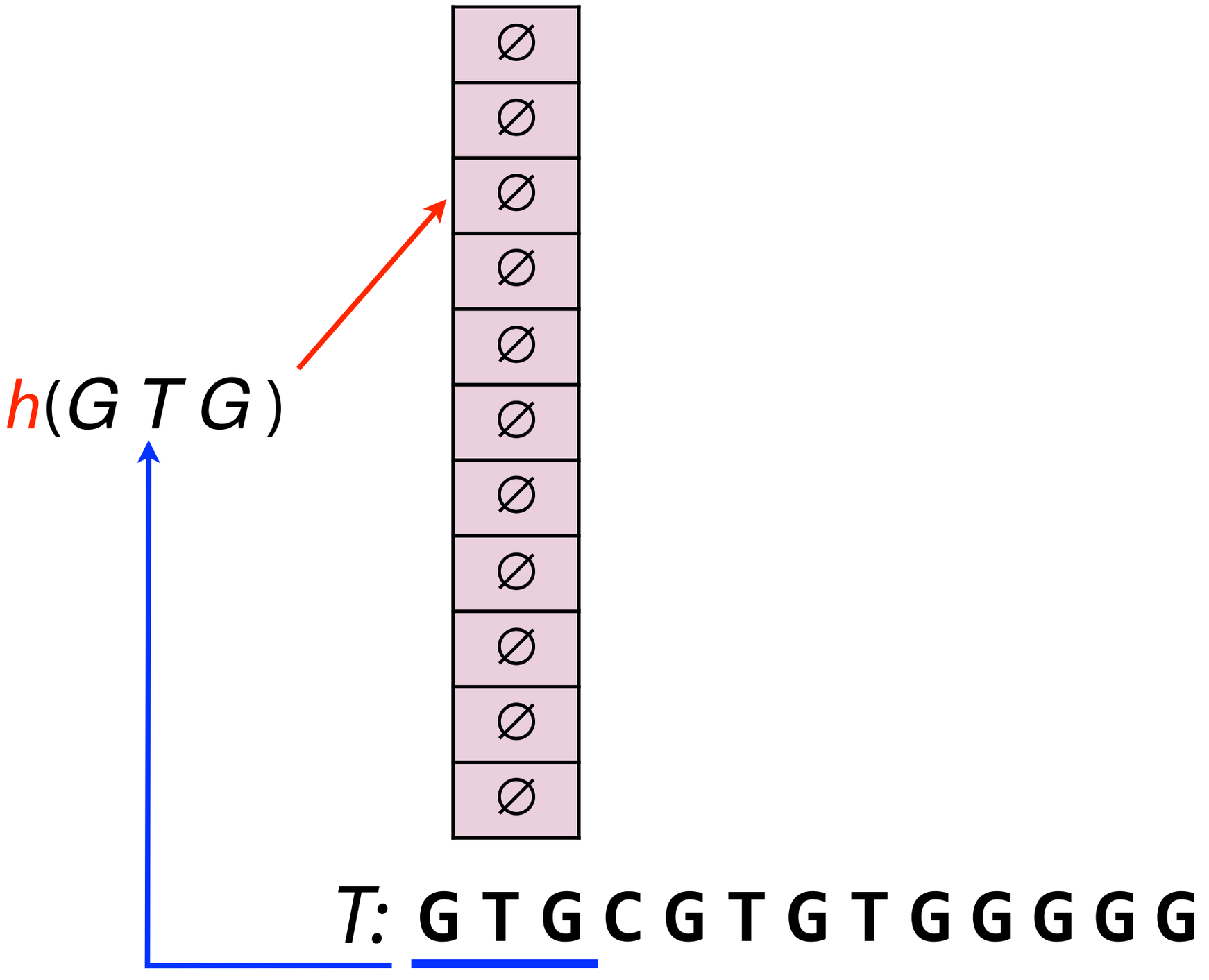
http://j.mp/CG_KmerIndex

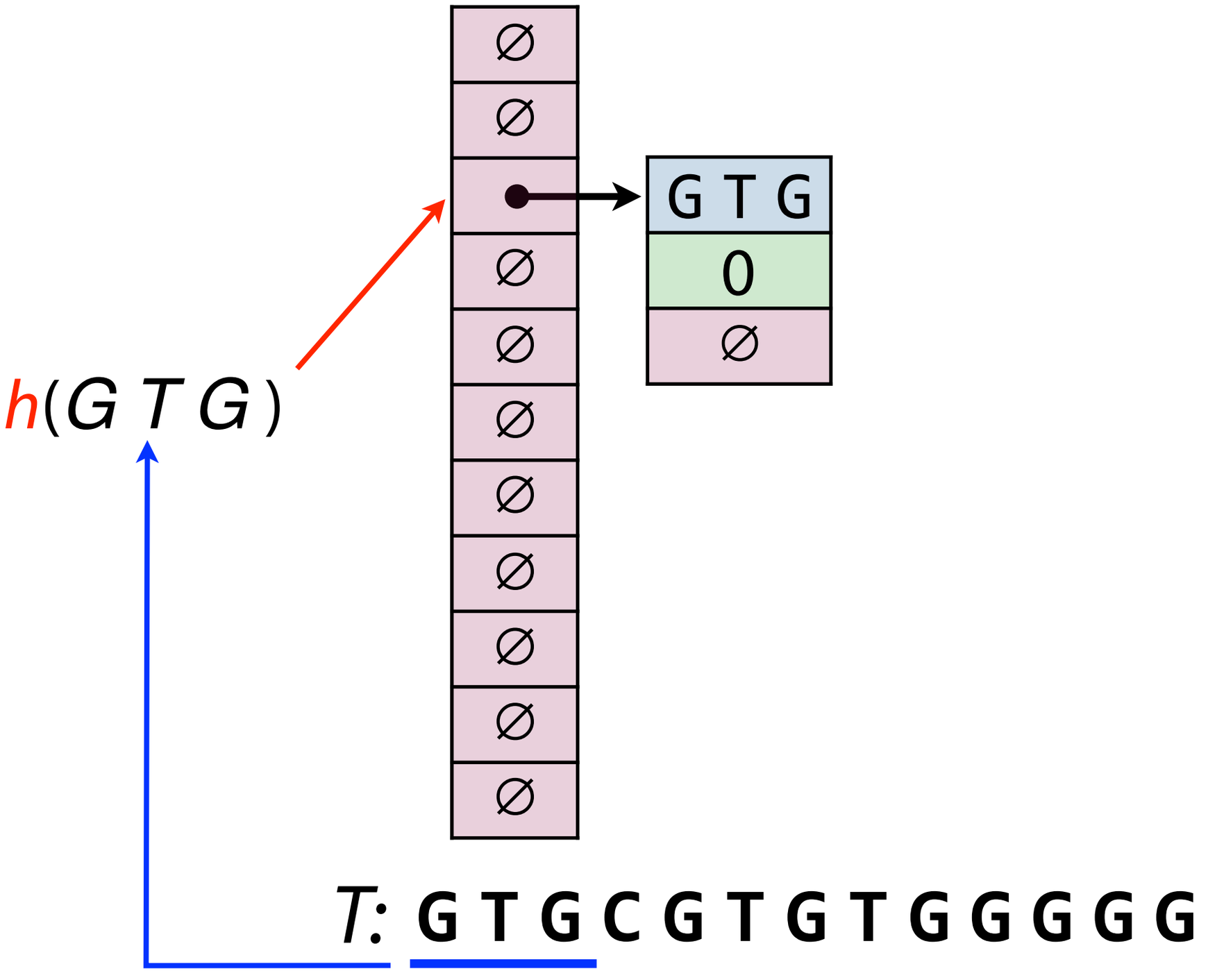
Hash table as multimap

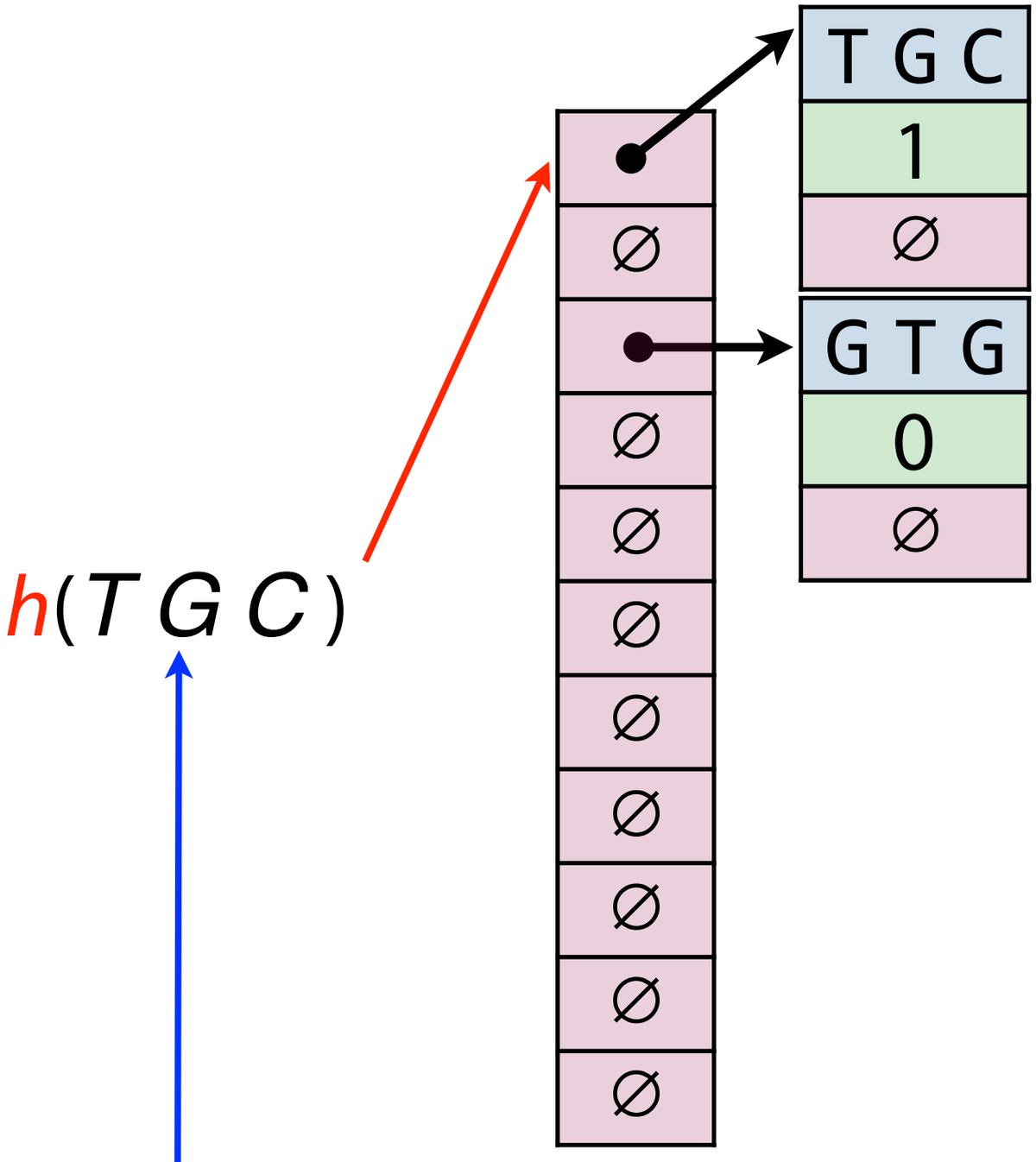


Hash function *h* maps
3-mers to buckets

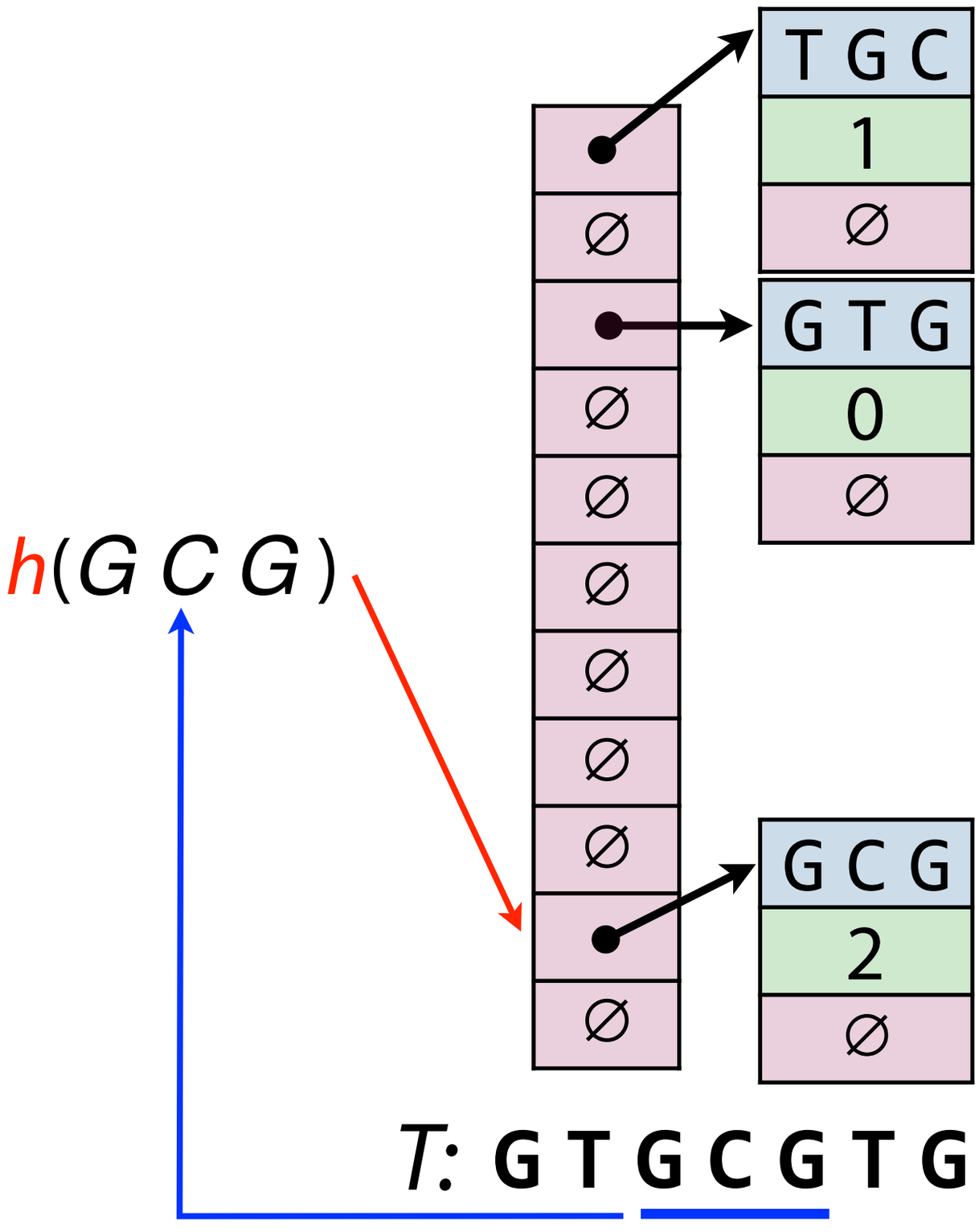
T: **G T G C G T G T G G G G**

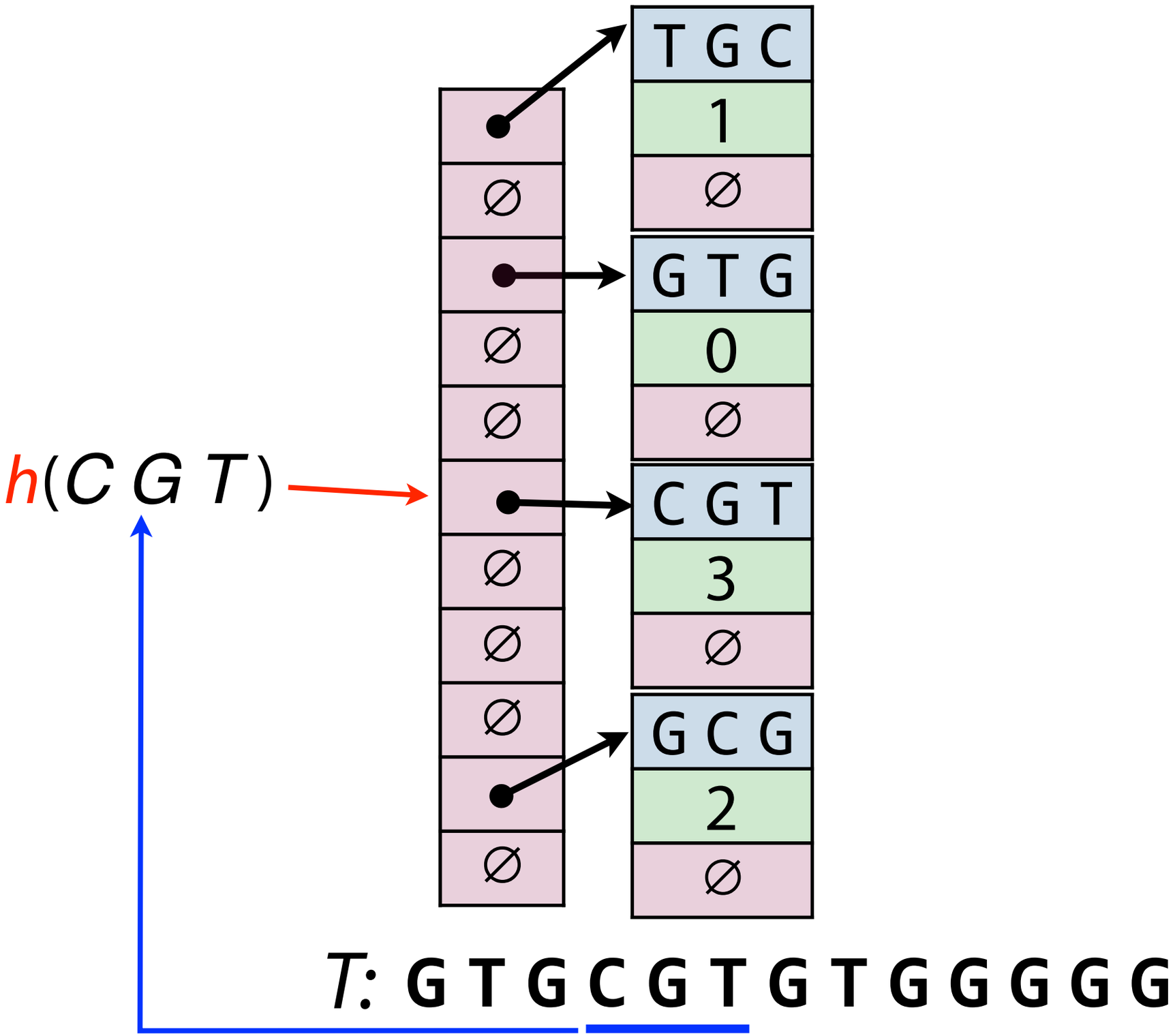

h(G T G)
∅
∅
∅
∅
∅
∅
∅
∅
∅
∅
∅
∅
T: G T G C G T G T G G G G

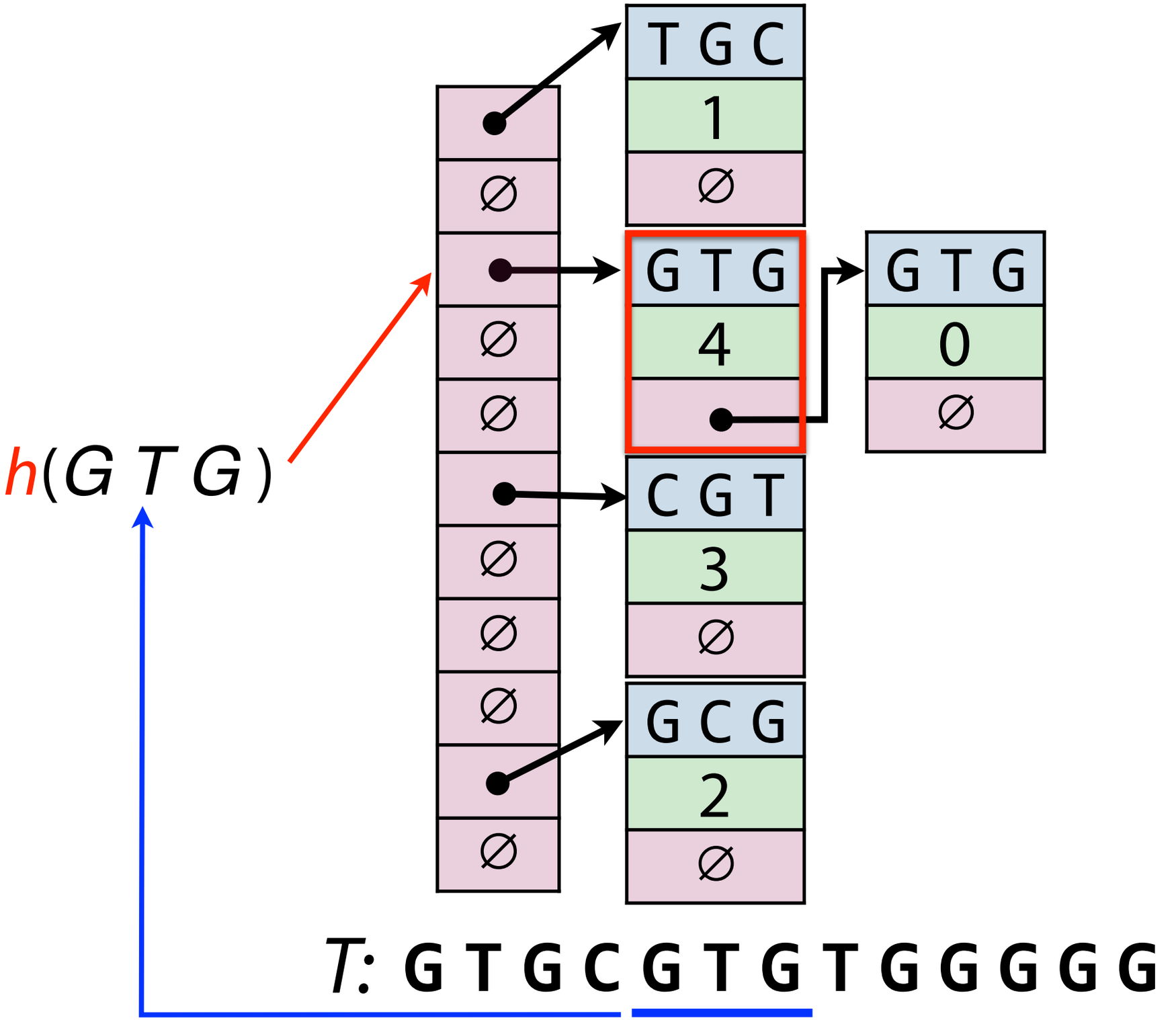

h(G T G)
∅
∅
G T G
0
∅
∅
∅
∅
∅
∅
∅
∅
∅
T: G T G C G T G T G G G G

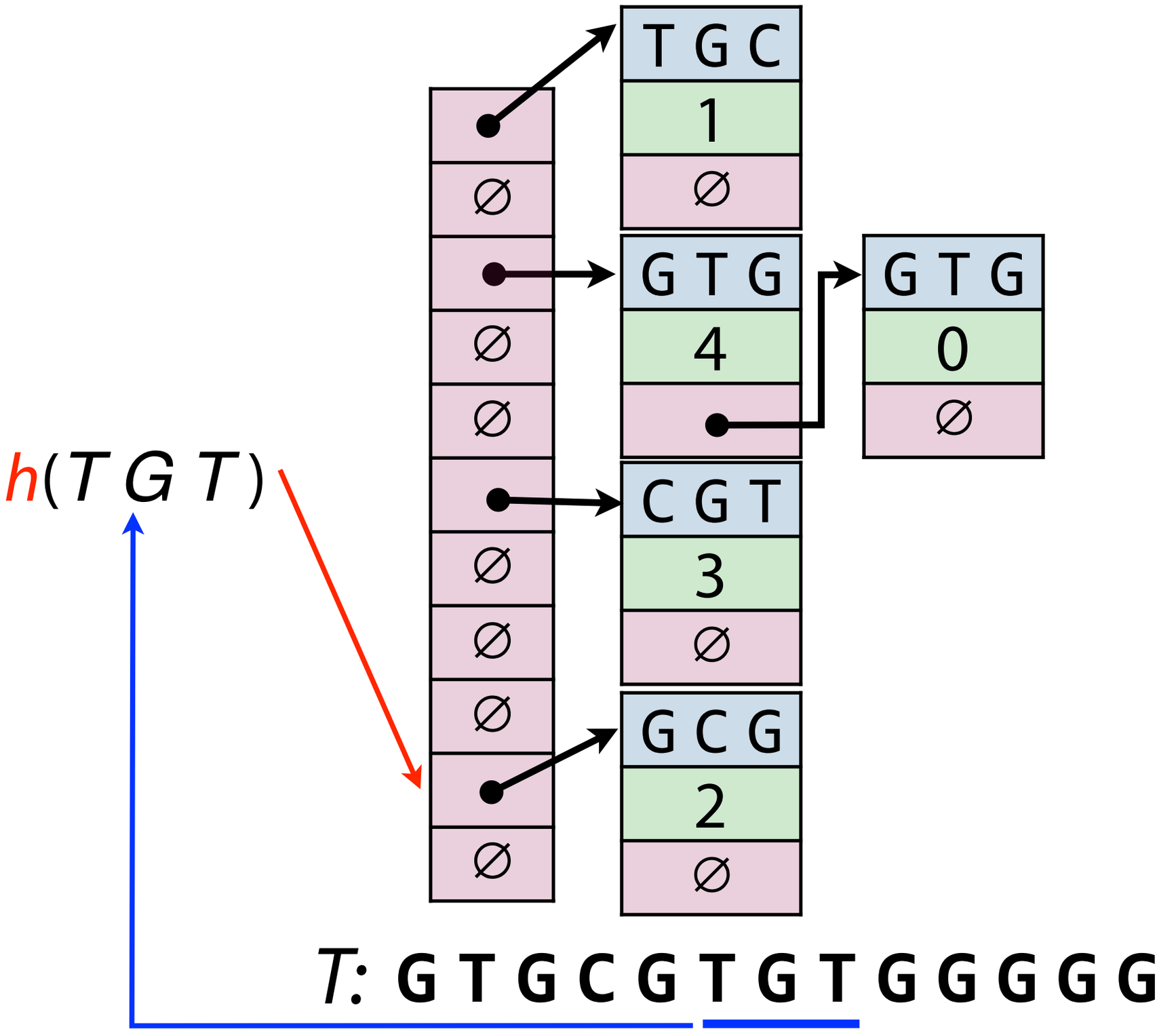


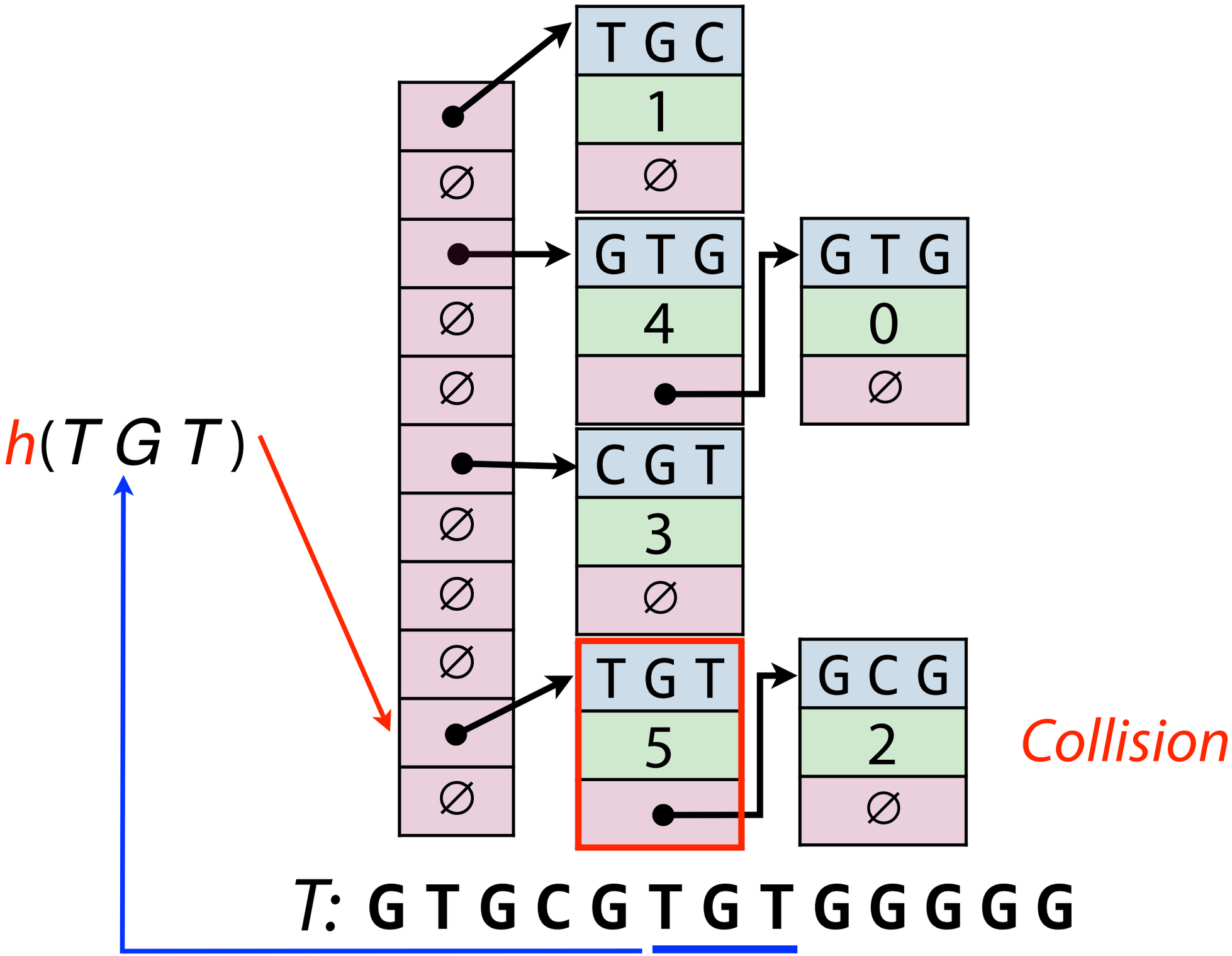
T: **G T G C G T G T G G G G**

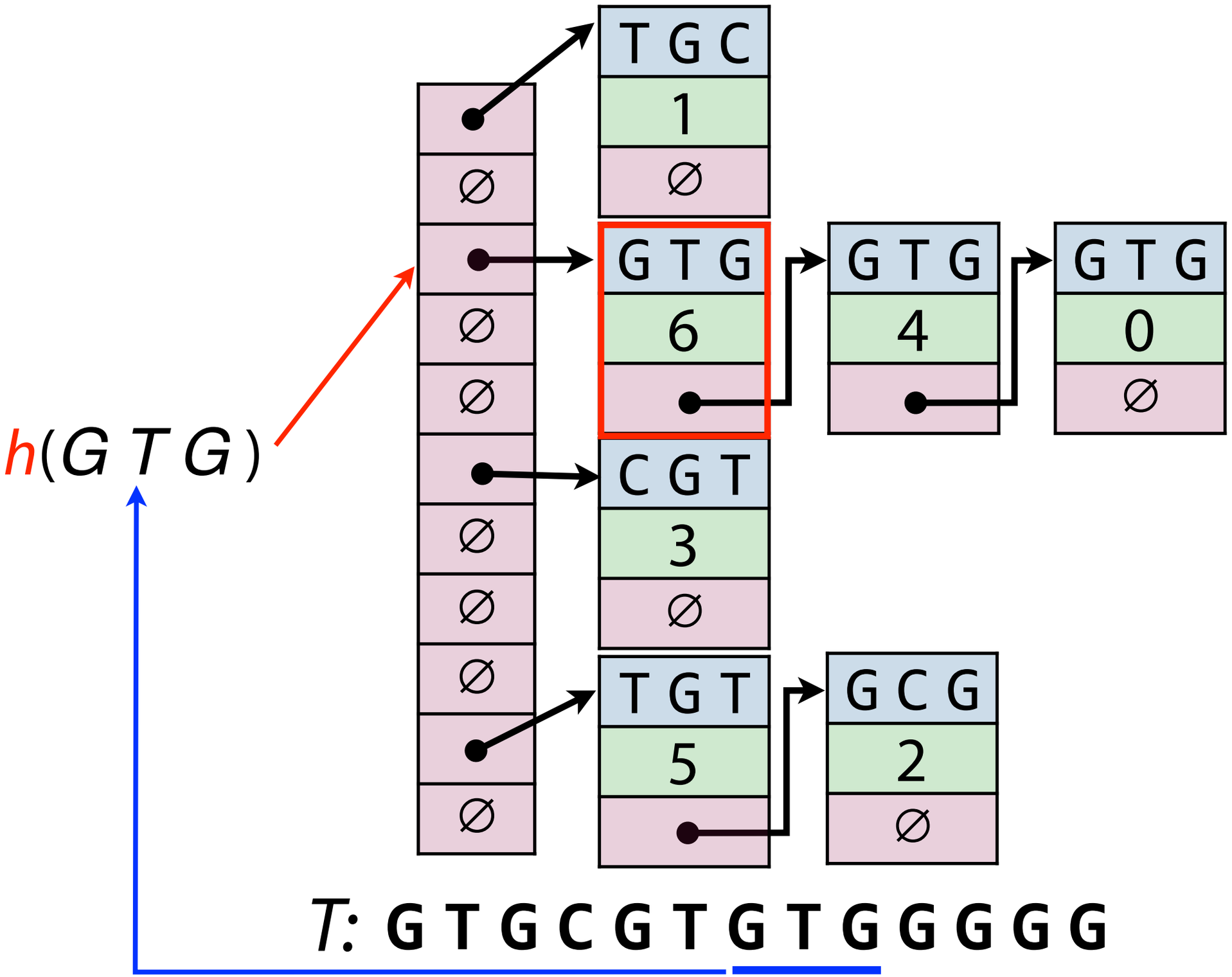

h(G C G)
T G C
1
∅
G T G
0
∅
∅
∅
∅
∅
∅
∅
∅
∅
∅
G C G
2
∅
T: G T G C G T G T G G G G

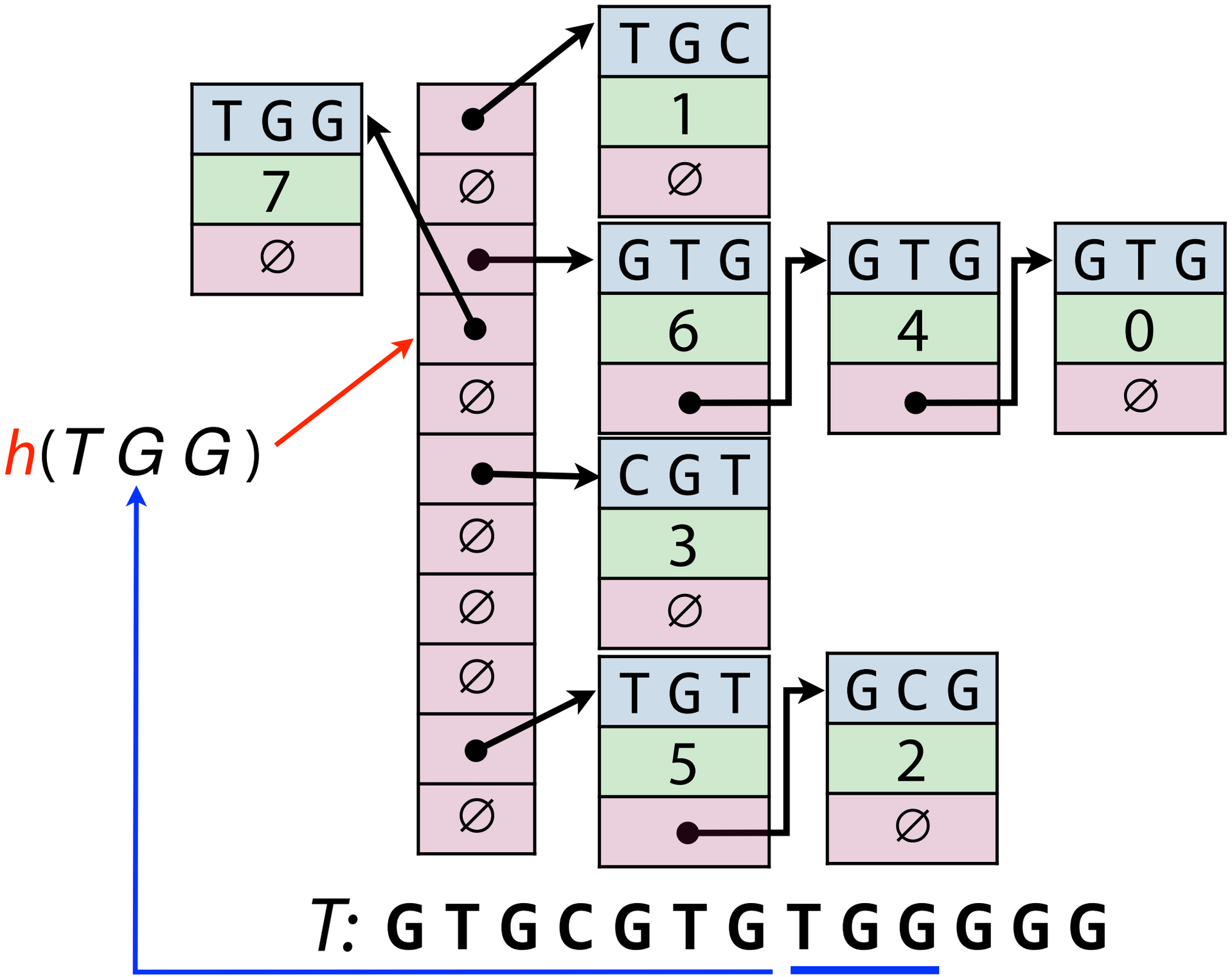

h(C G T)
T G C
1
∅
G T G
0
∅
C G T
3
∅
G C G
2
∅
∅
∅
∅
∅
∅
∅
∅
∅
T: G T G C G T G T G G G G

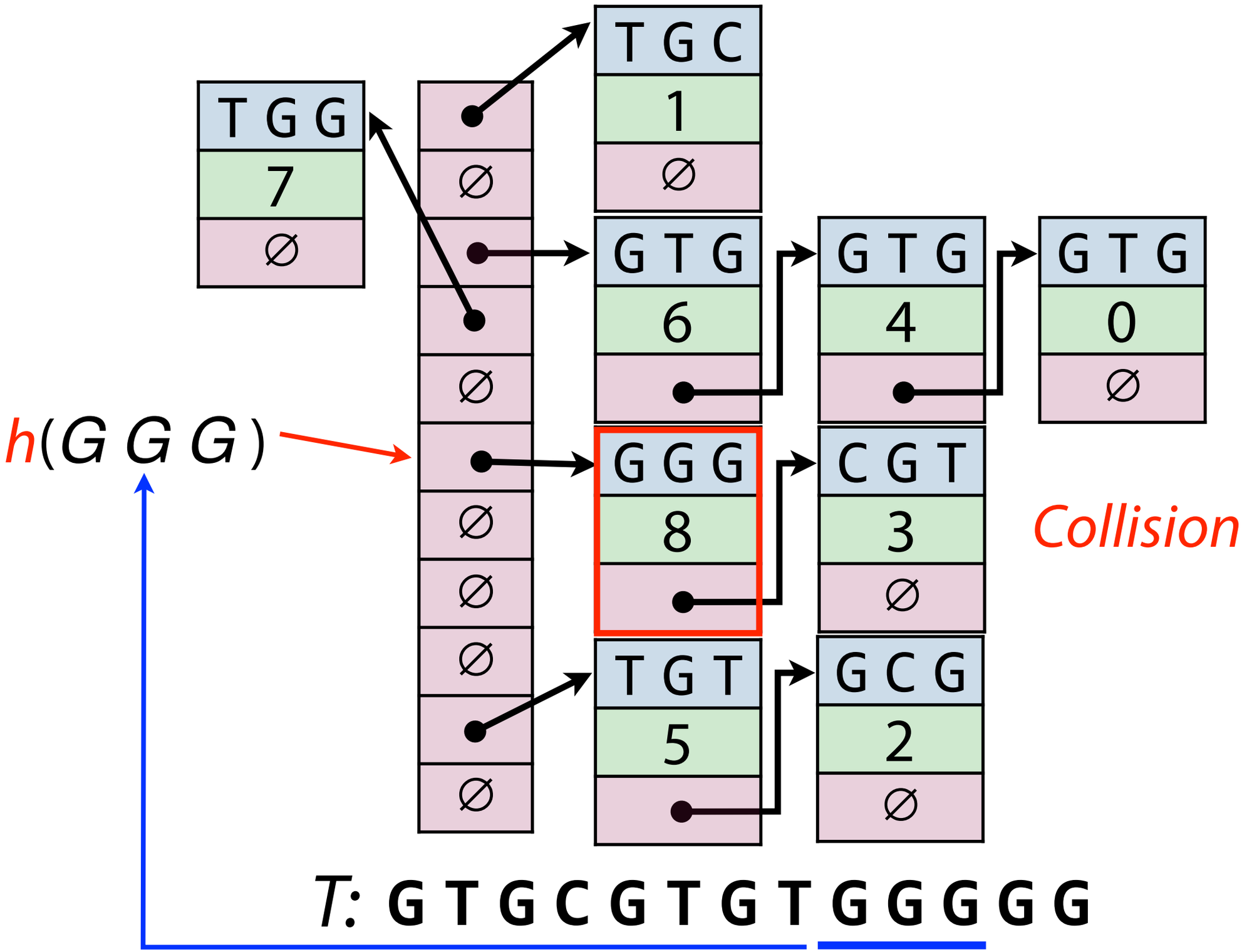

h(G T G)
T G C
1
∅
G T G
4
G T G
0
∅
C G T
3
∅
G C G
2
∅
∅
∅
∅
∅
∅
∅
∅
∅
T: G T G C G T G T G G G G

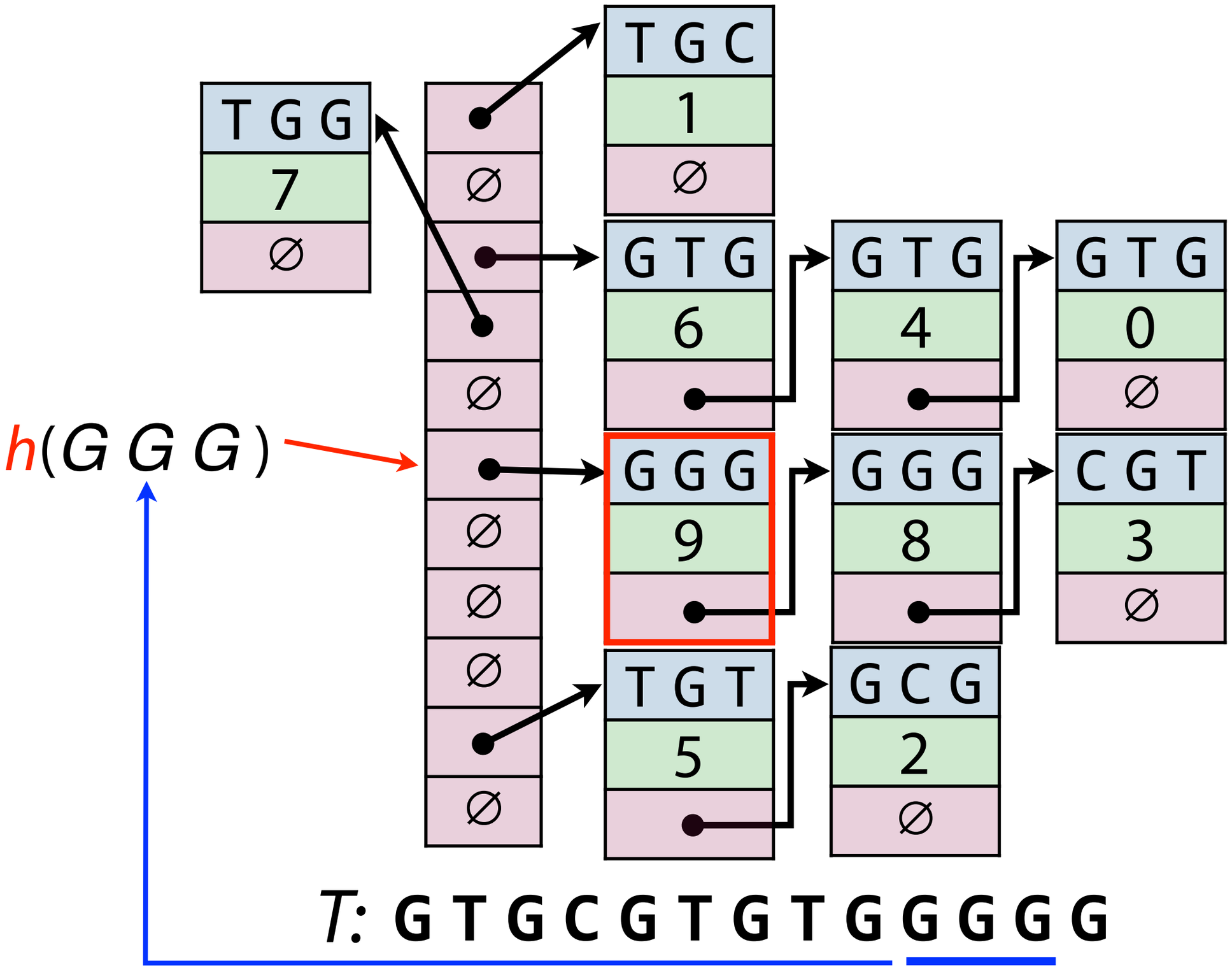

h(T G T)
T G C
1
∅
G T G
4
G T G
0
∅
C G T
3
∅
G C G
2
∅
∅
∅
∅
∅
∅
∅
∅
∅
T: G T G C G T G T G G G G

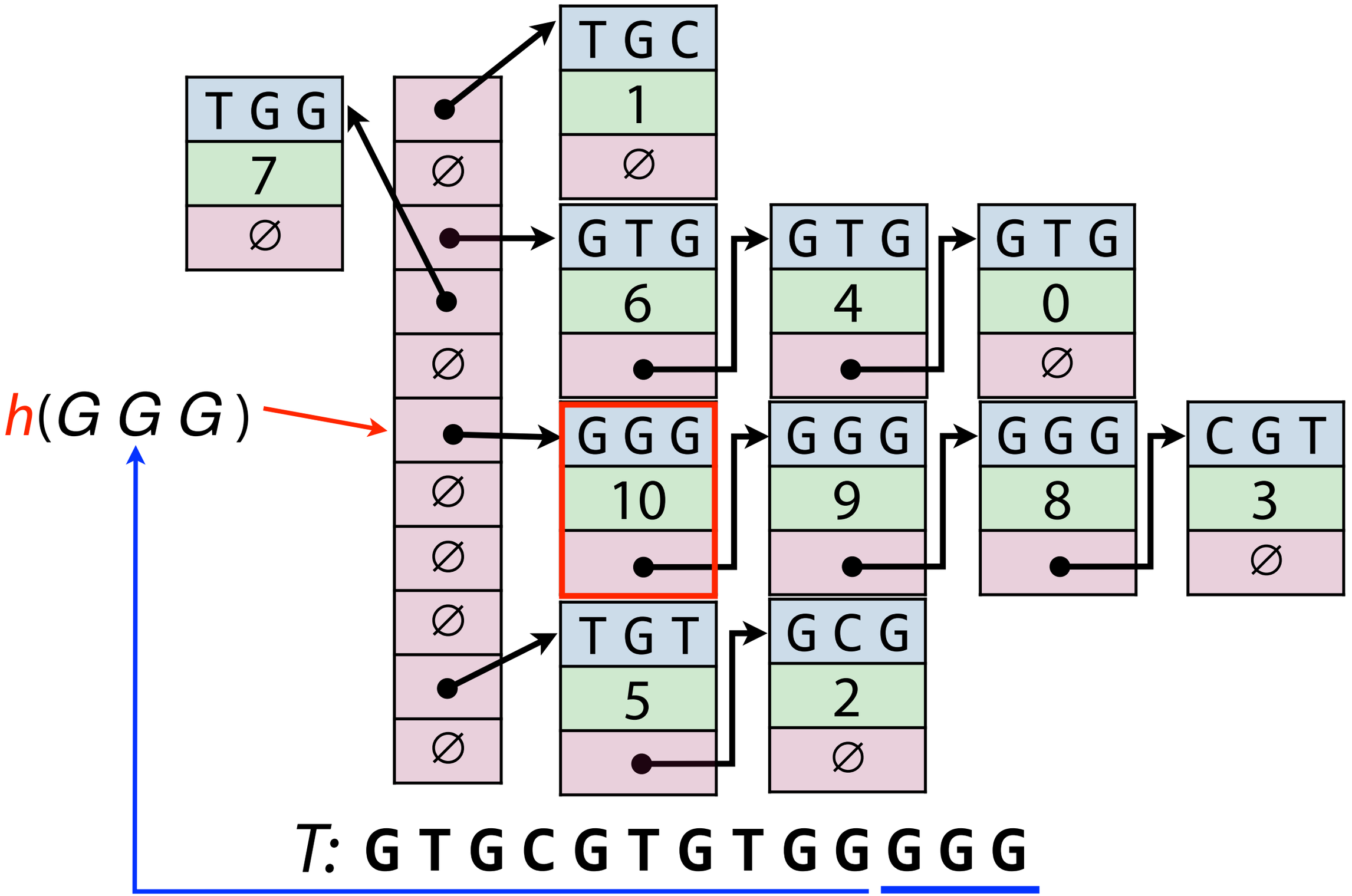

h(T G T)
T G C
1
∅
G T G
4
G T G
0
∅
C G T
3
∅
T G T
5
G C G
2
∅
Collision
T: G T G C G T G T G G G G G

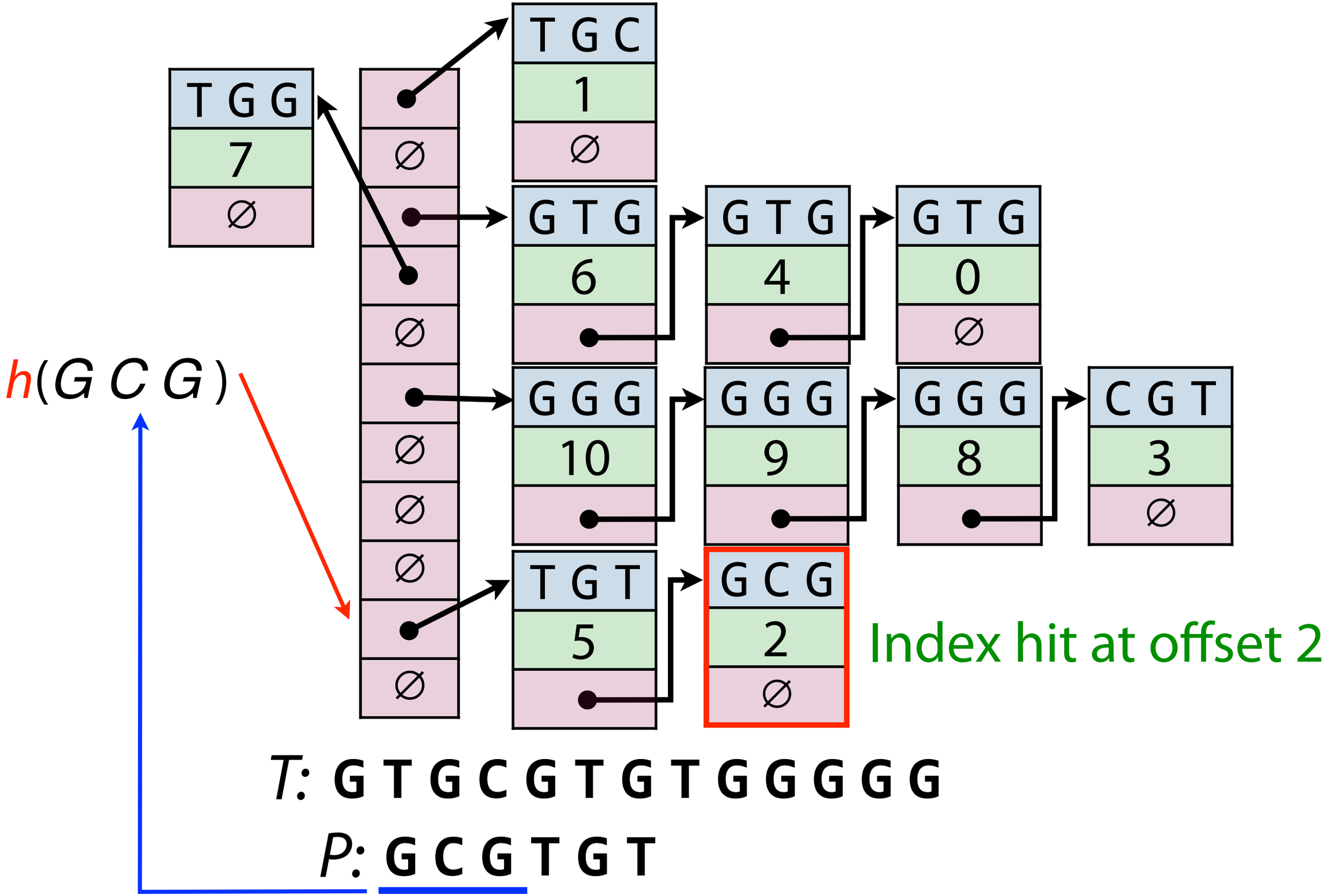

T G C
1
∅
G T G
6
G T G
4
G T G
0
∅
C G T
3
∅
T G T
5
G C G
2
∅
∅
∅
∅
∅
∅
∅
∅
h(G T G)
T: G T G C G T G T G G G G


T G G
7
∅
T G C
1
∅
G T G
6
G T G
4
G T G
0
∅
C G T
3
∅
T G T
5
G C G
2
∅
∅
∅
∅
∅
∅
∅
∅
h(T G G)
T: G T G C G T G T G G G G G


T G C
1
∅
T G G
7
∅
G T G
6
G T G
4
G T G
0
∅
h(G G G)
G G G
8
C G T
3
∅
∅
∅
∅
∅
∅
∅
T G T
5
G C G
2
∅
Collision
T: G T G C G T G T G G G G G


T G G
7
∅
T G C
1
∅
G T G
6
G T G
4
G T G
0
∅
G G G
9
G G G
8
C G T
3
∅
T G T
5
G C G
2
∅
h(G G G)
T: G T G C G T G T G G G G


T G G
7
∅
T G C
1
∅
G T G
6
G T G
4
G T G
0
∅
G G G
10
G G G
9
G G G
8
C G T
3
∅
T G T
5
G C G
2
∅
h(G G G)
T: G T G C G T G T G G G G G


h(G C G)
T G G
7
∅
T G C
1
∅
G T G
6
G T G
4
G T G
0
∅
G G G
10
G G G
9
G G G
8
C G T
3
∅
T G T
5
G C G
2
∅
Index hit at offset 2
T: G T G C G T G T G G G G G
P: G C G T G T

Python dictionary

```
>>> t = 'GTGCGTGTGGGGG'
>>> table = {'GTG':[0, 4, 6], 'TGC':[1],
             'GCG':[2], 'CGT':[3], 'TGT':[5],
             'TGG':[7], 'GGG':[8, 9, 10]}
>>> table['GGG']
[8, 9, 10]
>>> table['CGT']
[3]
```

The built-in *dictionary* type in Python is a building block for a map or (in this case) a multimap

Hash table k-mer index implementation

http://j.mp/CG_KmerIndexHash