Reducing Code Size Using Outlining

Jessica Paquette
Apple

Outline

- Code size
- Outlining
- Results
- Future work

Motivating Example

```
callq  _printf
movl $2, %edx
movl -8(%rbp), %esi
addl $1, %esi
movl %esi, -8(%rbp)
…
movl -20(%rbp), %ecx
movl $2, %edx
movl -8(%rbp), %esi
addl $1, %esi
movl %esi, -8(%rbp)
…
movl $2, %edx
movl -8(%rbp), %esi
addl $1, %esi
movl %esi, -8(%rbp)
```

```
callq  _printf
movl $2, %edx
movl -8(%rbp), %esi
addl $1, %esi
movl %esi, -8(%rbp)
…
movl -20(%rbp), %ecx
movl $2, %edx
movl -8(%rbp), %esi
addl $1, %esi
movl %esi, -8(%rbp)
…
movl $2, %edx
movl -8(%rbp), %esi
addl $1, %esi
movl %esi, -8(%rbp)
```

```
callq  _printf
movl $2, %edx
movl -8(%rbp), %esi
addl $1, %esi
movl %esi, -8(%rbp)
…
movl -20(%rbp), %ecx
movl $2, %edx
movl -8(%rbp), %esi
addl $1, %esi
movl %esi, -8(%rbp)
…
movl $2, %edx
movl -8(%rbp), %esi
addl $1, %esi
movl %esi, -8(%rbp)
```

```
NEW_FUNC:
movl $2, %edx
movl -8(%rbp), %esi
addl $1, %esi
movl %esi, -8(%rbp)
retq
```

```
callq  _printf
callq NEW_FUNC
…
movl -20(%rbp), %ecx
callq NEW_FUNC
…
callq NEW_FUNC
```

```
NEW_FUNC:
movl $2, %edx
movl -8(%rbp), %esi
addl $1, %esi
movl %esi, -8(%rbp)
retq
```

Outlining

Replacing repeated sequences of instructions with calls to equivalent functions

Outliner

A pass that finds repeated instruction sequences and outlines them.

```
callq  _printf
movl $2, %edx
movl -8(%rbp), %esi
addl $1, %esi
movl %esi, -8(%rbp)
…
movl -20(%rbp), %ecx
movl $2, %edx
movl -8(%rbp), %esi
addl $1, %esi
movl %esi, -8(%rbp)
…
movl $2, %edx
movl -8(%rbp), %esi
addl $1, %esi
movl %esi, -8(%rbp)
```

```
A  callq  _printf
B  movl $2, %edx
C  movl -8(%rbp), %esi
D  addl $1, %esi
E  movl %esi, -8(%rbp)
   …
F  movl -20(%rbp), %ecx
B  movl $2, %edx
C  movl -8(%rbp), %esi
D  addl $1, %esi
E  movl %esi, -8(%rbp)
   …
B  movl $2, %edx
C  movl -8(%rbp), %esi
D  addl $1, %esi
E  movl %esi, -8(%rbp)
```

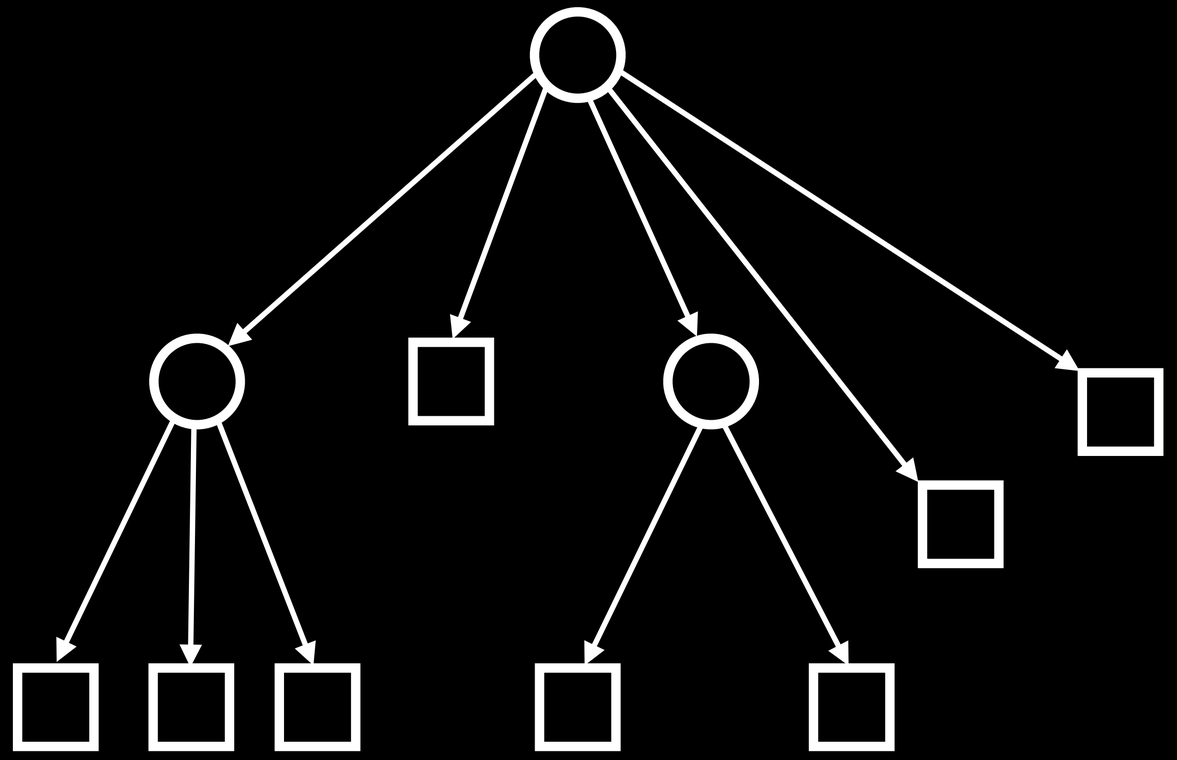
Programs are like **strings**

Find repeated substrings

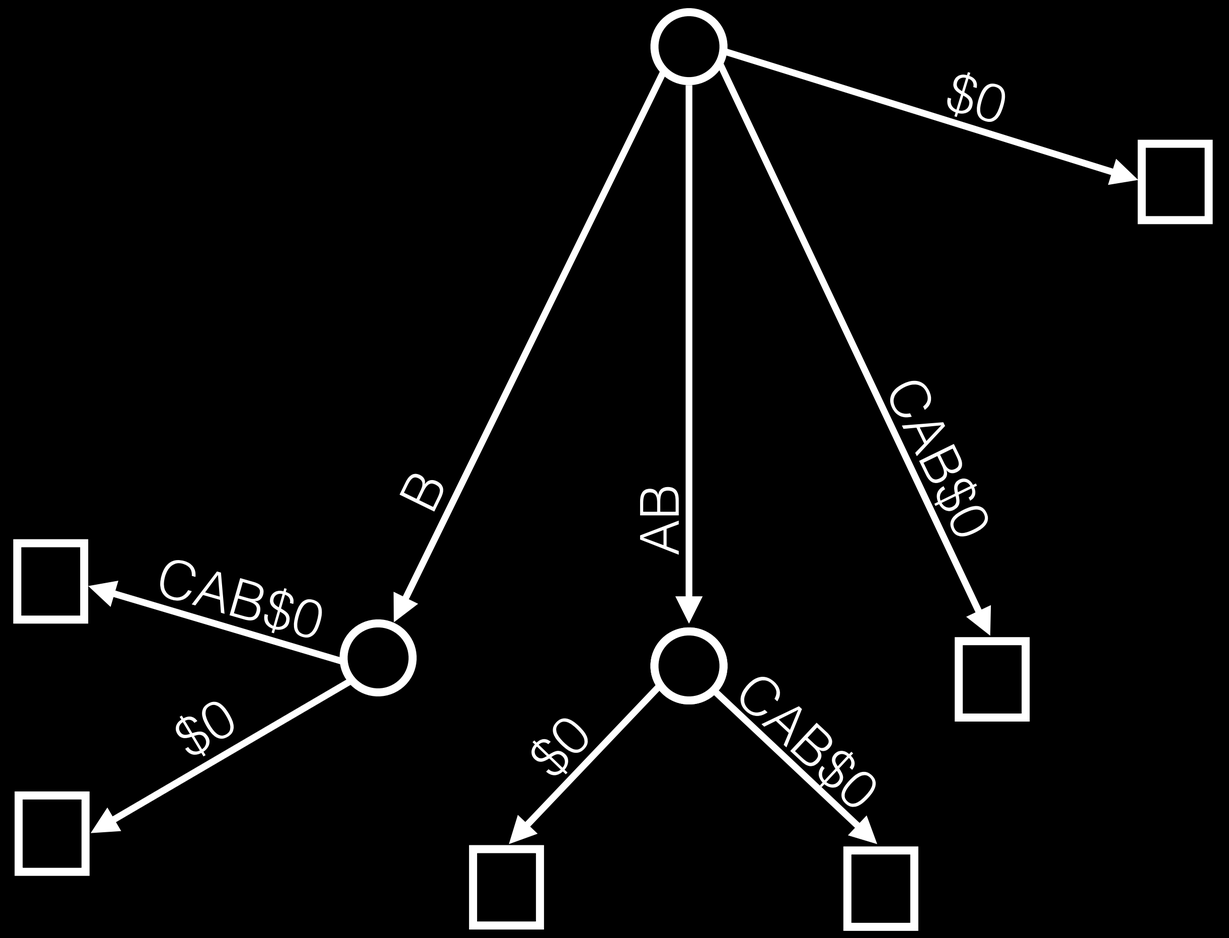
Suffix Tree

A data structure for string searching

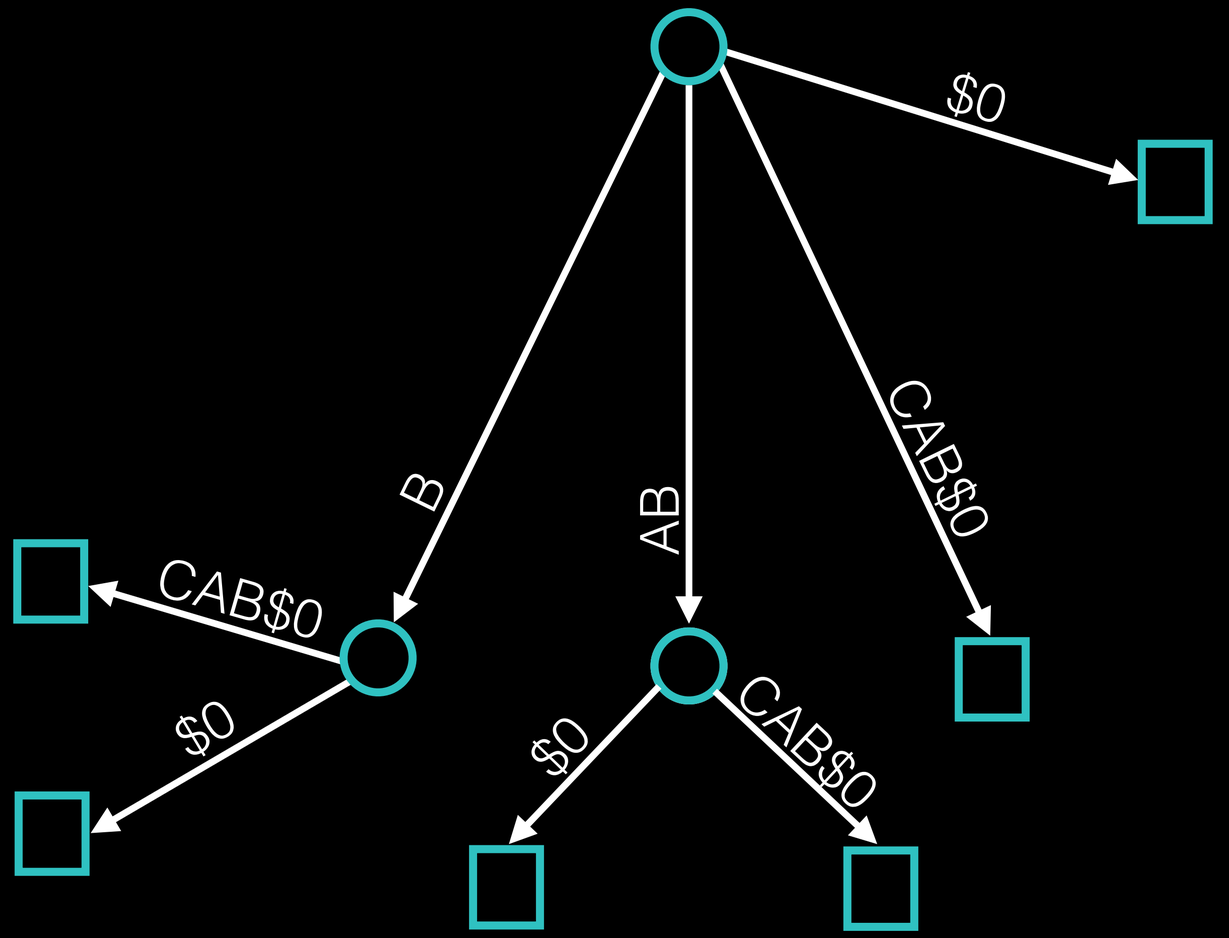
(Not a suffix trie!)



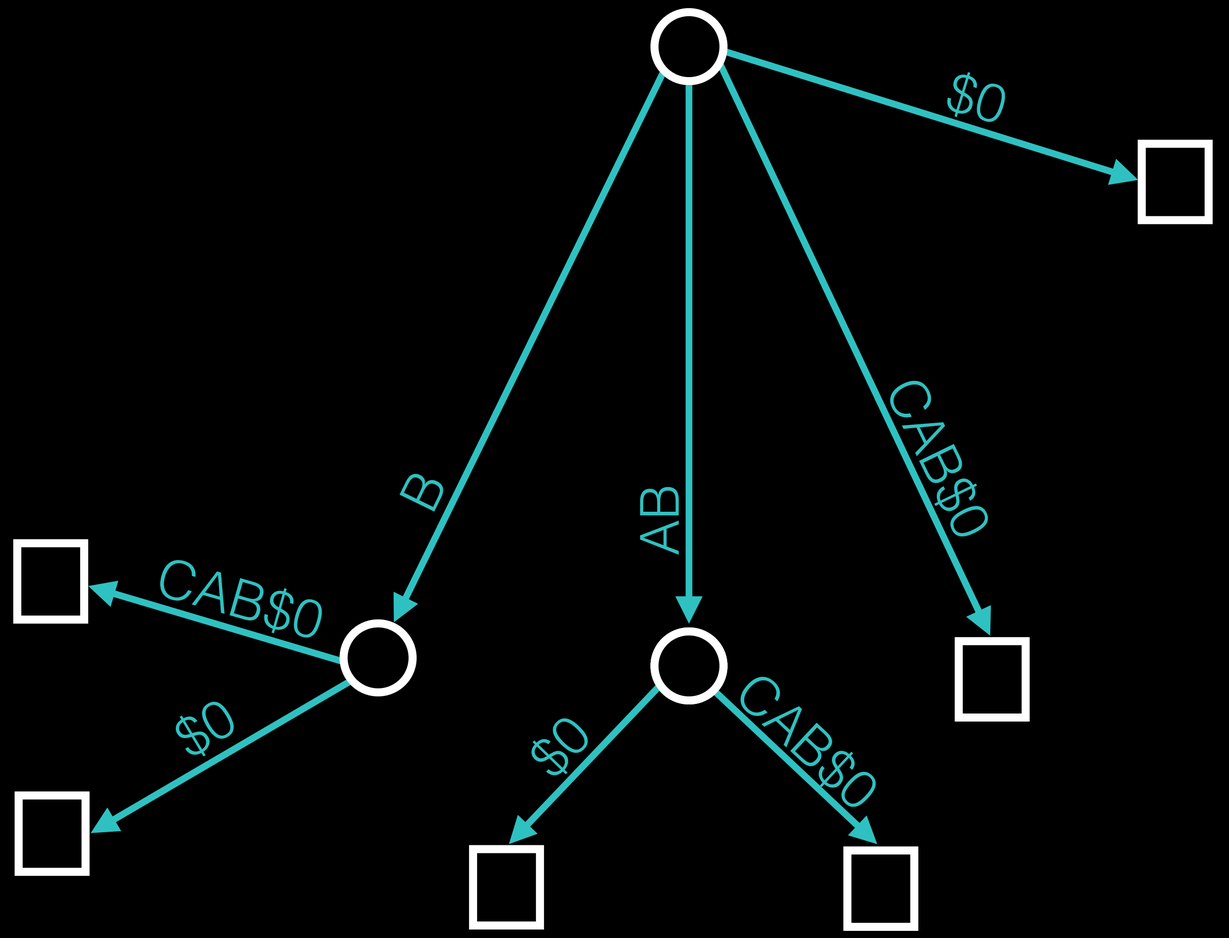
A B C A B $0



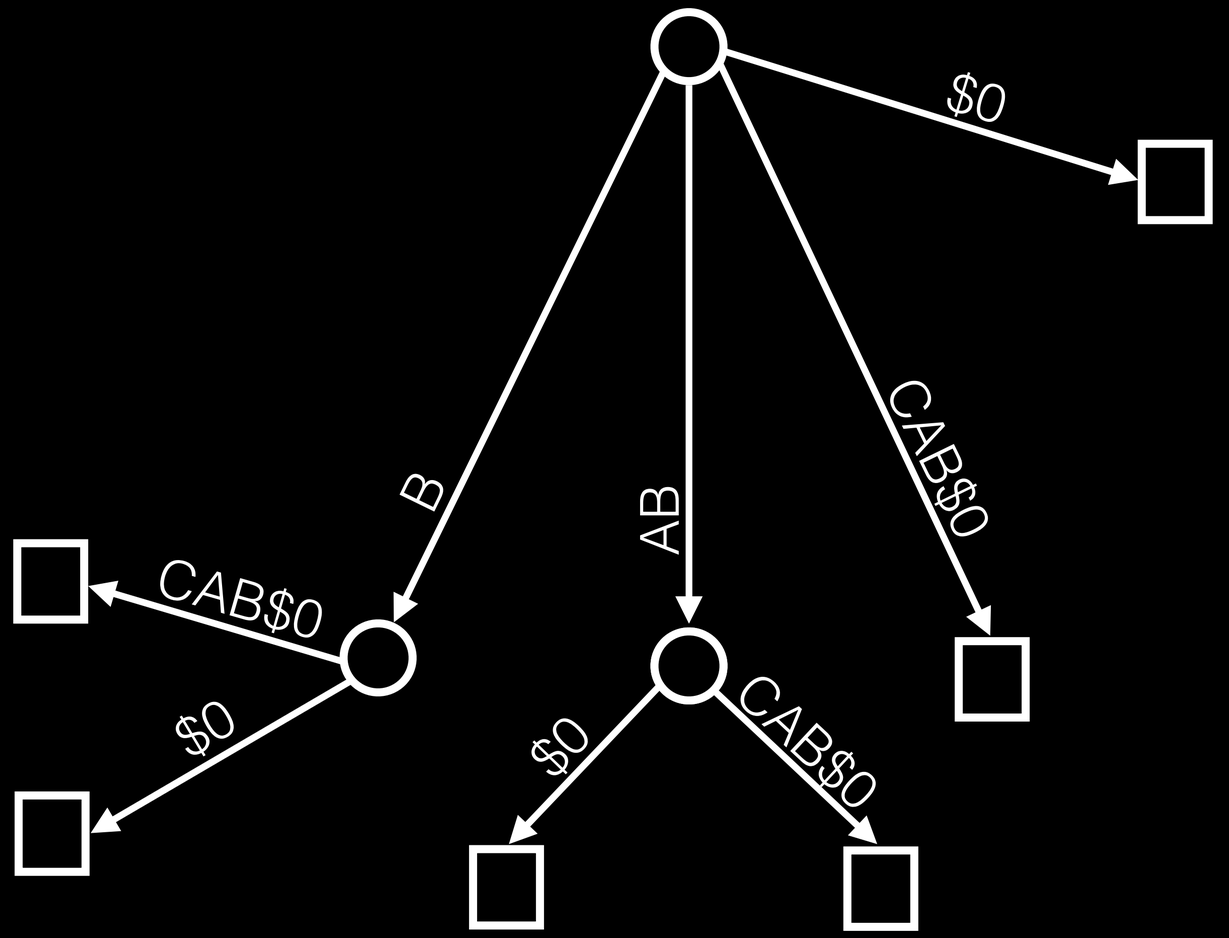
A B C A B $0



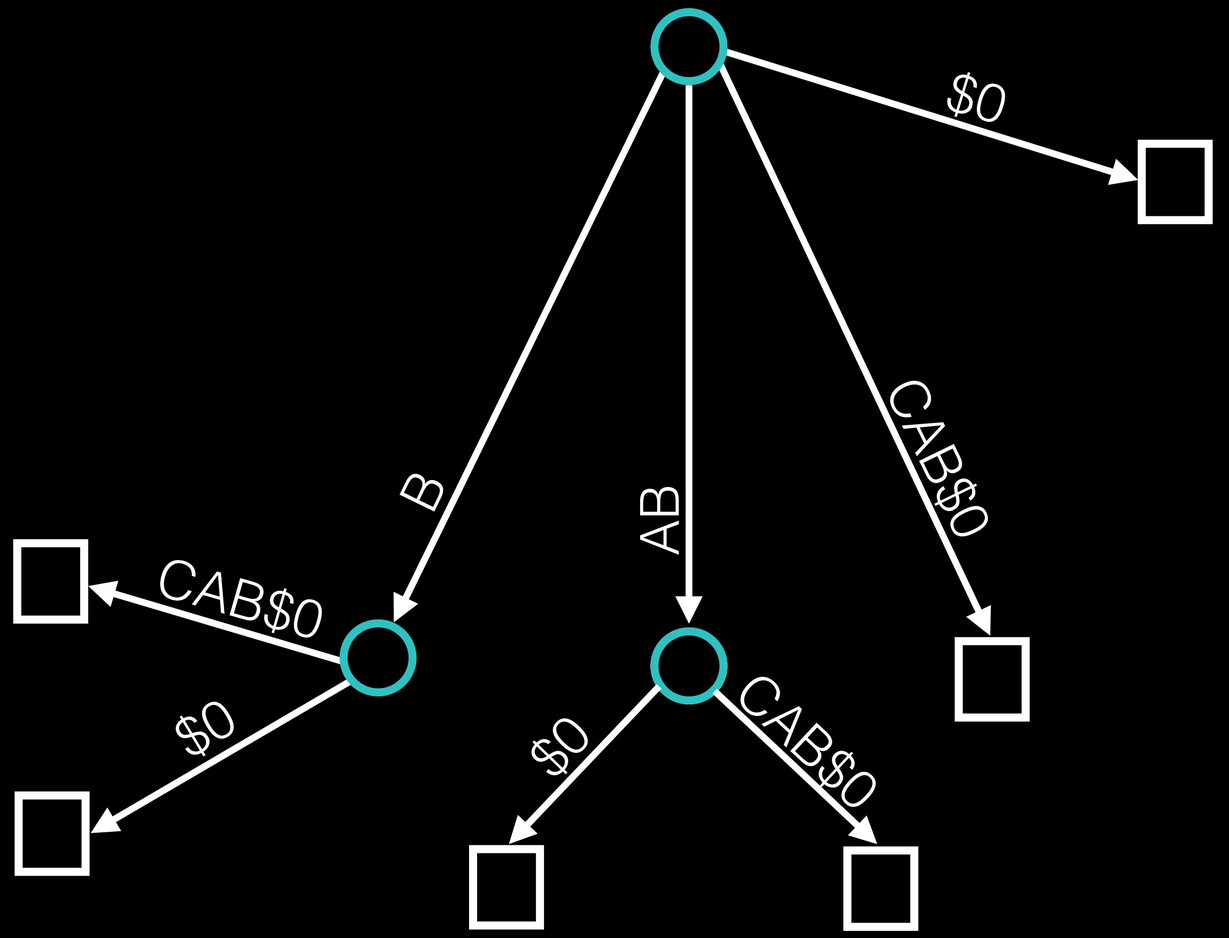
A B C A B $0



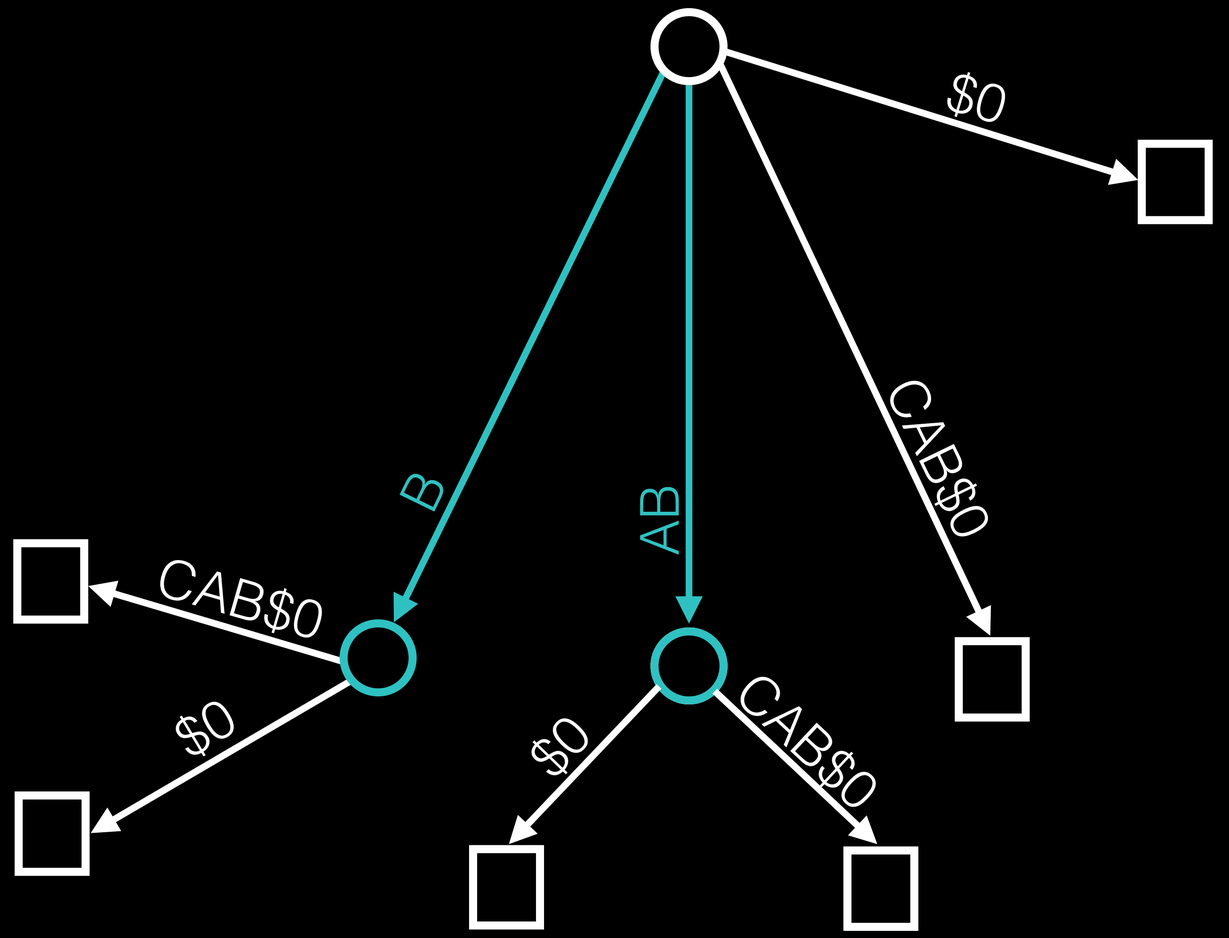
A B C A B $0



A B C A B $0



A B C A B $0



Advantages

Given a string of length L…

- O(L) construction
- O(L) longest repeated substring
- O(L) time most frequent substring

Simplified Suffix Tree Construction

A B C A B $0

Suffixes

A B C A B $0

Suffixes

A B C A B $0

Suffixes

A

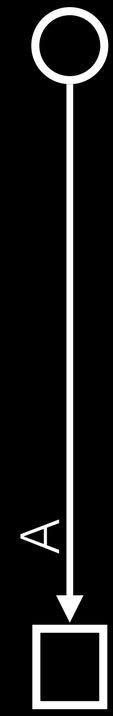
A B C A B $0

Suffixes

A B C A B $0

Suffixes

B, AB



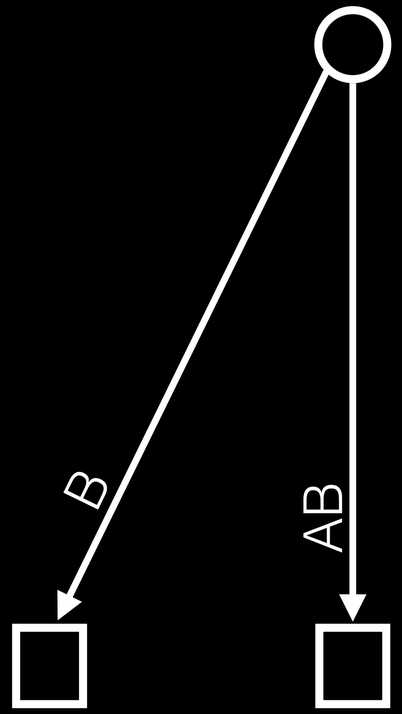
A B C A B $0

Suffixes

B

A B C A B $0

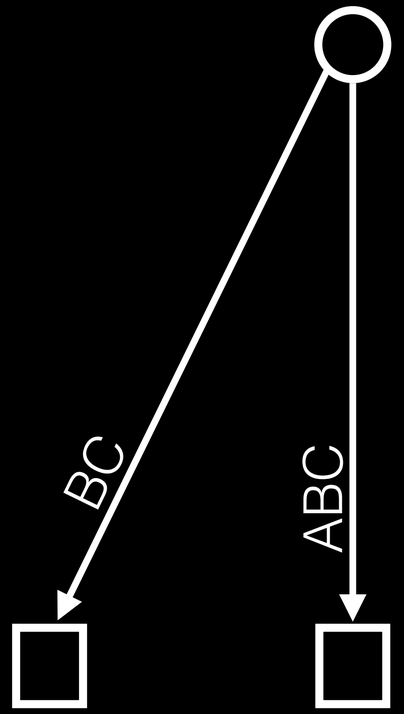
Suffixes



A B C A B $0

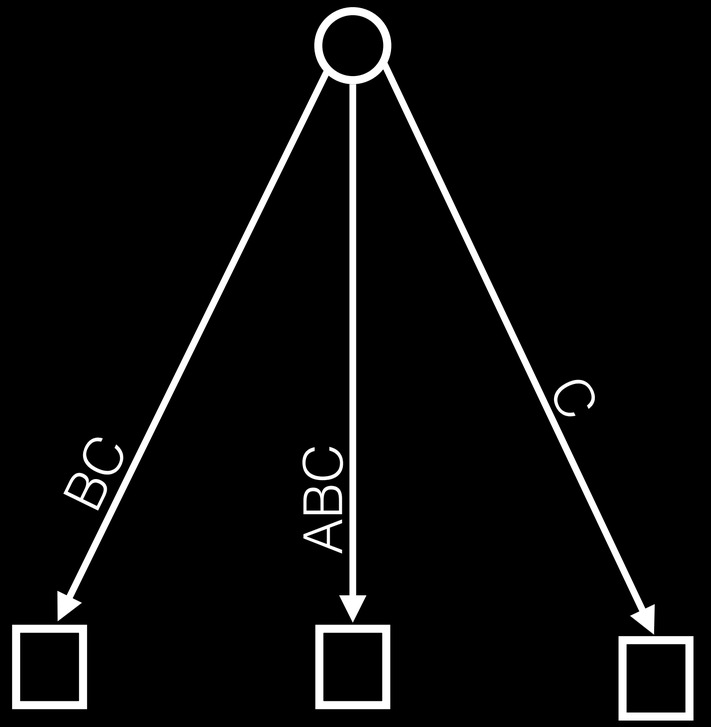
Suffixes

C



A B C A B $0

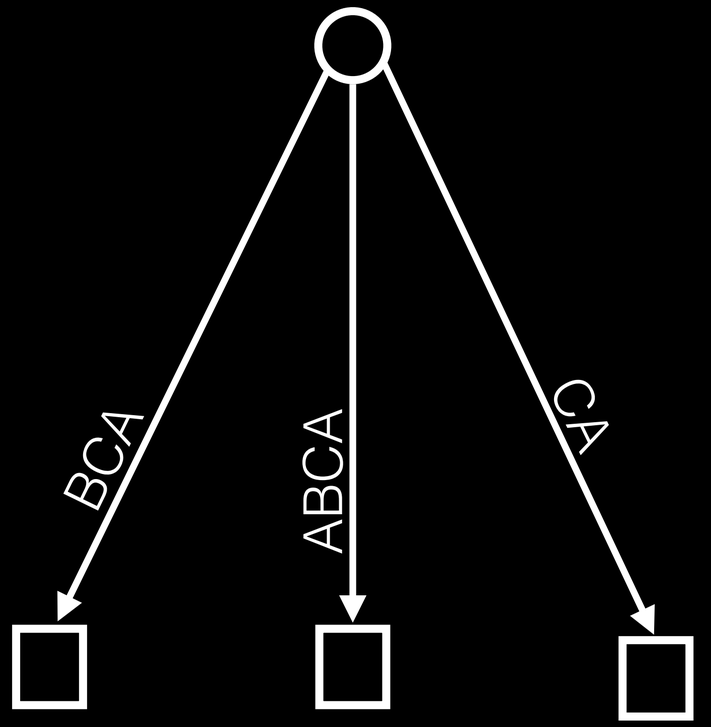
Suffixes



A B C A B $0

Suffixes

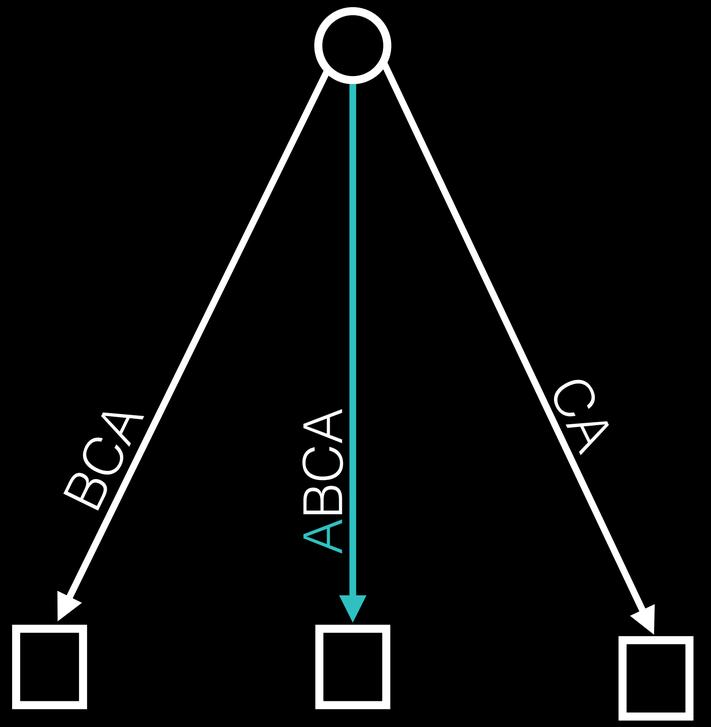
A



A B C A B $0

Suffixes

A

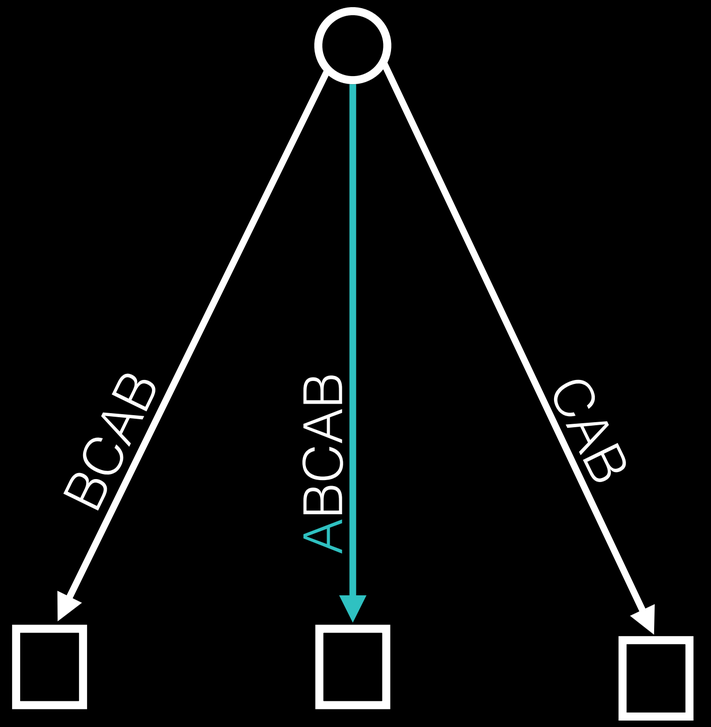


A B C A B $0

Suffixes

A B

B

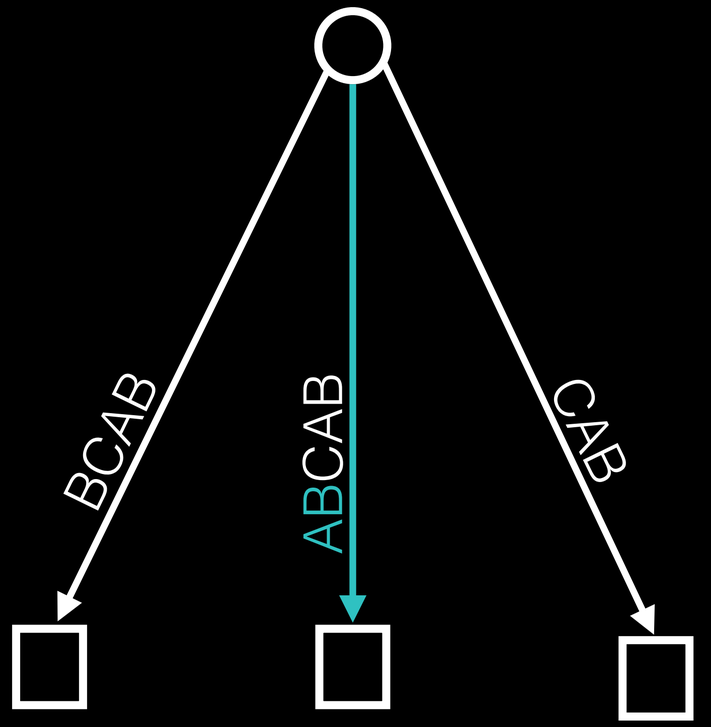


A B C A B $0

Suffixes

A B

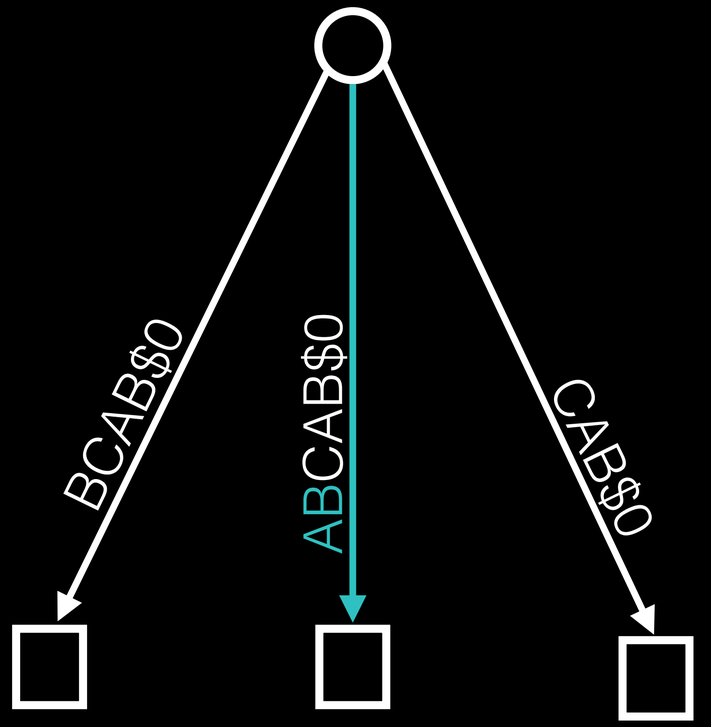
B



A B C A B $0

Suffixes

A B $0
B $0
$0

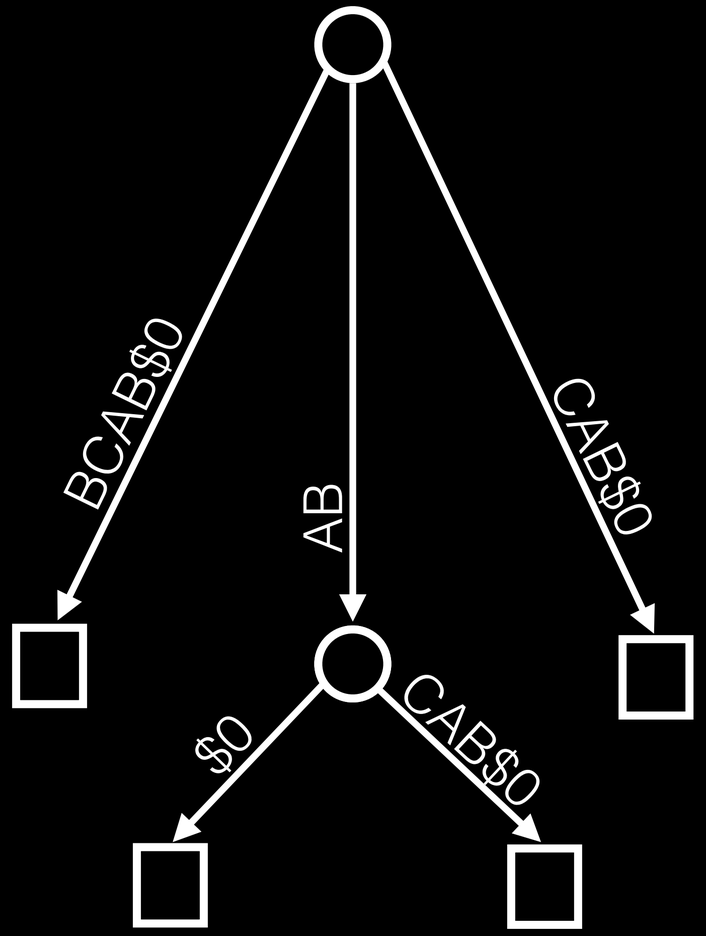


A B C A B $0

Suffixes

B $0

$0

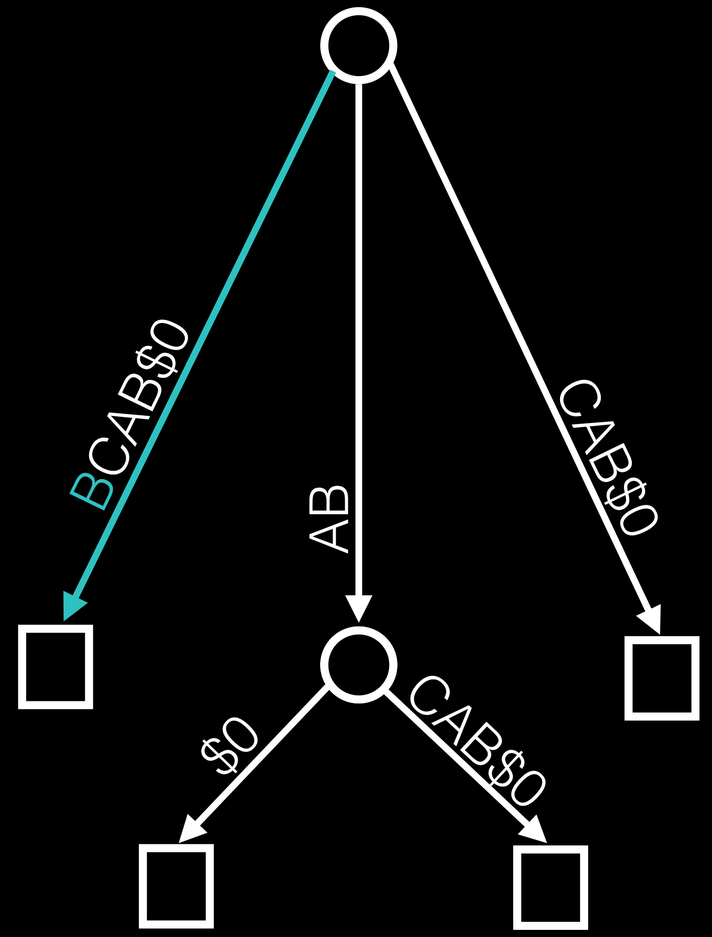


A B C A B $0

Suffixes

B $0

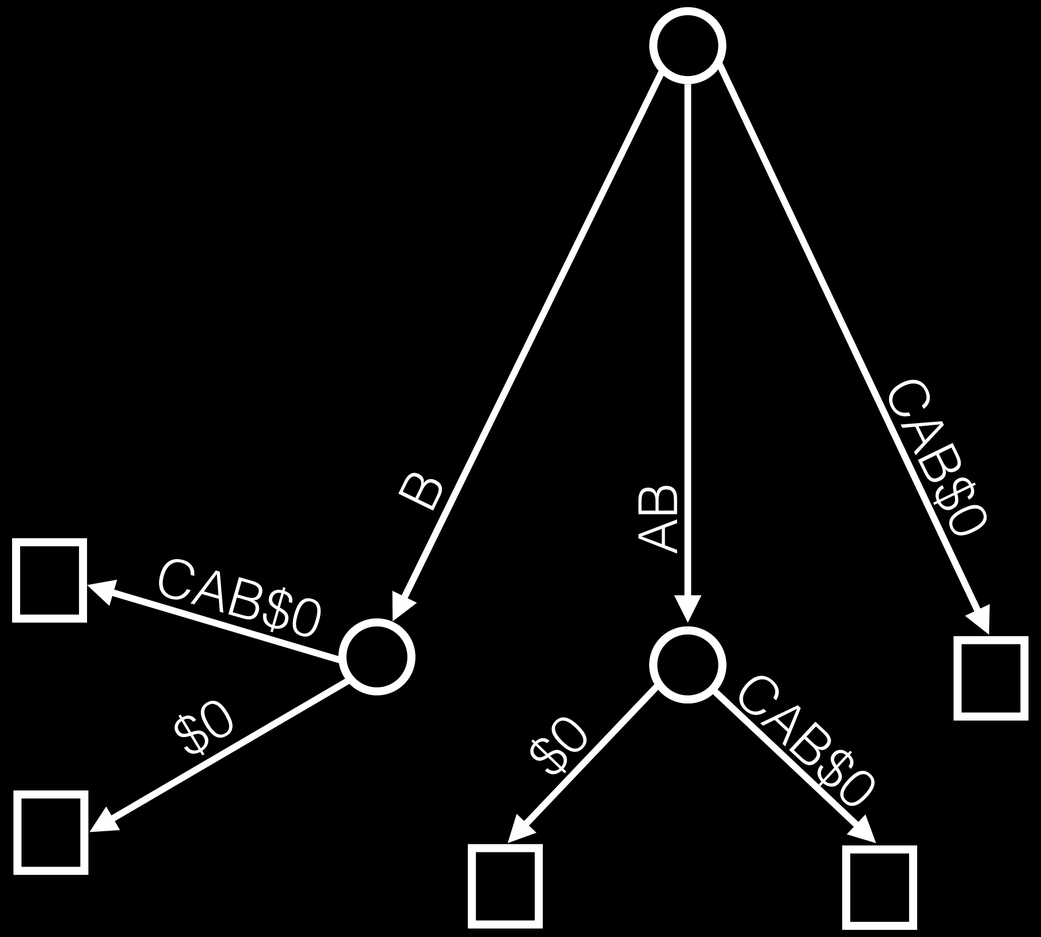
$0



A B C A B $0

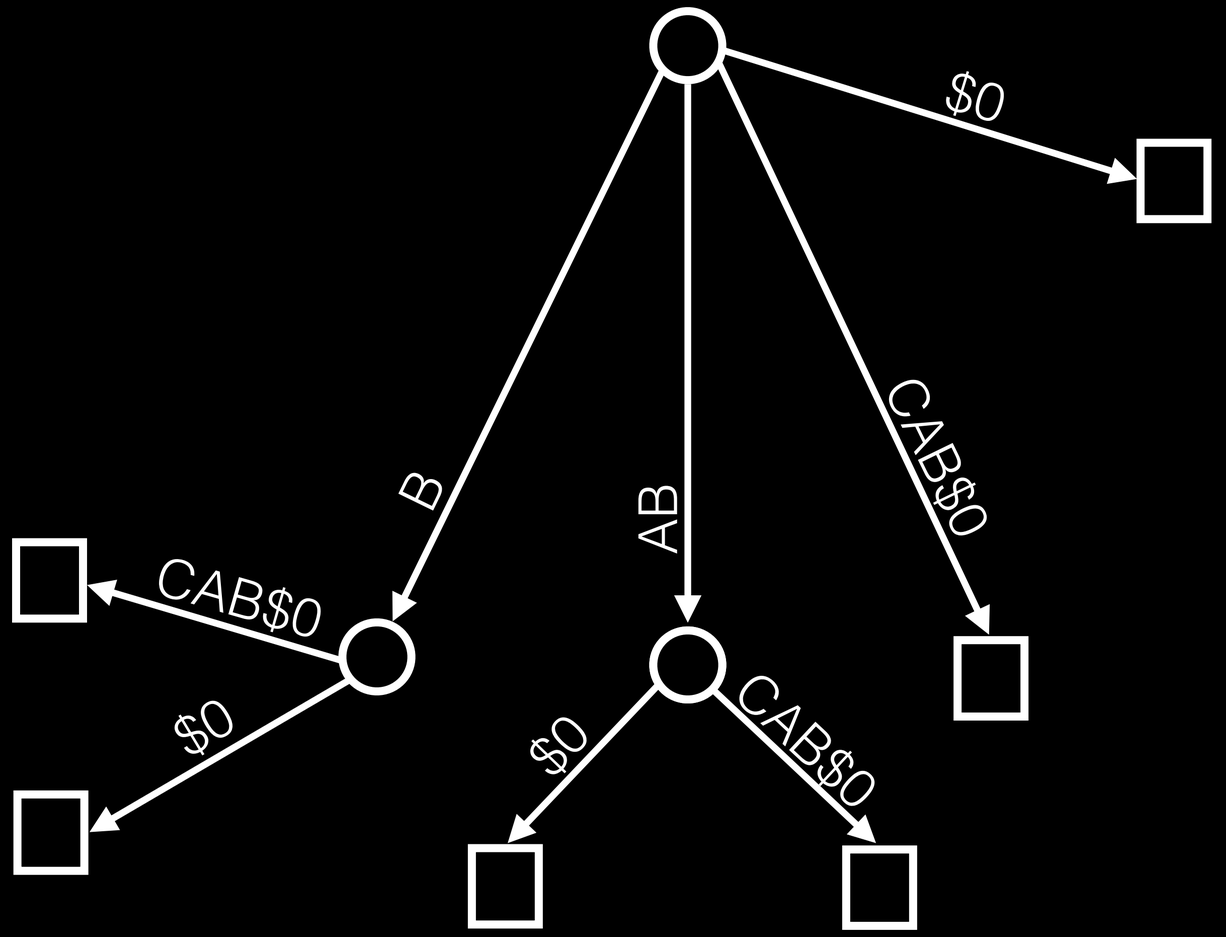
Suffixes

$0



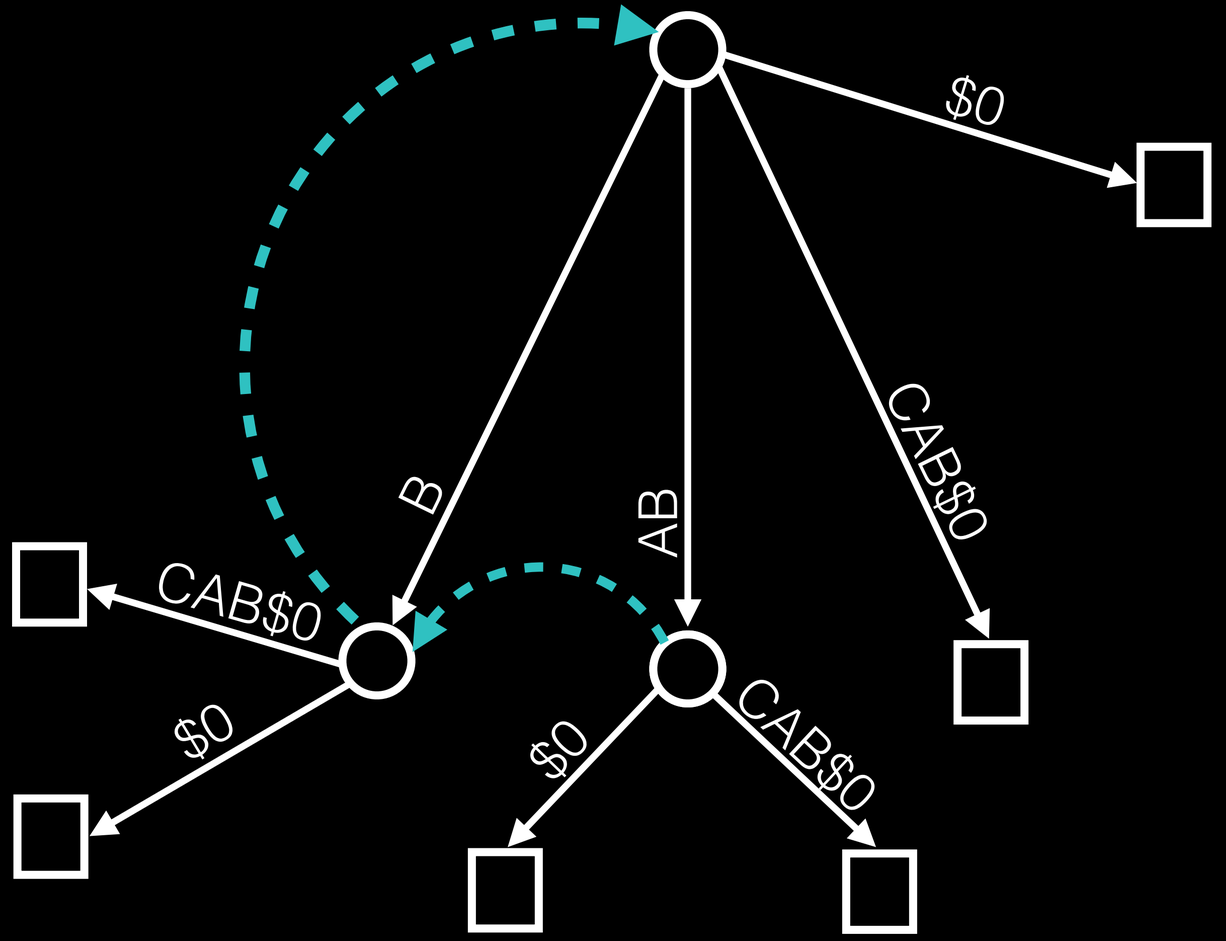
A B C A B $0

Suffixes



*Note: Real construction is more complex

Need to store links between internal nodes



Representation

Suffix Tree Struct

```
struct SuffixTree {
    Node *Root;
    size_t LeafEnd;
    ActiveState Active;
    …
};
```

Suffix Tree Struct

```
struct SuffixTree {
   Node *Root;
   size_t LeafEnd;
   ActiveState Active;

   StringType longestRepeatedSubstring();
   void findOccurrences(std::vector<int> &Occurrences,
                        const StringType &QueryStr);
   void prune(const StringType &Str);
   …
};
```

Node Struct

```
struct Node {
    Node *Parent;
    std::map<CharacterType, Node *> Children;
    size_t StartIdx;
    size_t EndIdx;
    size_t SuffixIndex;
    …
};
```

Node Struct

```
struct Node {
    Node *Parent;
    std::map<CharacterType, Node *> Children;
    size_t StartIdx;
    size_t EndIdx;
    size_t SuffixIndex;

    bool Valid;
    …
};
```

Outlining Example

FOO ( )

A `R1 = 0xDEADBEEF`

B `R3 = R2 + R1`

C `R1 = *R5`

D `R7 = 0xFEEDFACE`

E `R1 = R1 - 1`

BAR ( )

F `R7 = R3 + R2`

A `R1 = 0xDEADBEEF`

B `R3 = R2 + R1`

C `R1 = *R5`

G `R7 = 0xFACEFEED`

A `R1 = 0xDEADBEEF`

B `R3 = R2 + R1`

C `R1 = *R5`

E `R1 = R1 - 1`

String Encoding

FOO ( )

```
A  R1 = 0xDEADBEEF
B  R3 = R2 + R1
C  R1 = *R5
D  R7 = 0xFEEDFACE
E  R1 = R1 - 1
```

---► A B C D E

BAR ( )

```
F  R7 = R3 + R2
A  R1 = 0xDEADBEEF
B  R3 = R2 + R1
C  R1 = *R5
G  R7 = 0xFACEFEED
A  R1 = 0xDEADBEEF
B  R3 = R2 + R1
C  R1 = *R5
E  R1 = R1 - 1
```

---► F A B C G A B C E

String Encoding

FOO ( )

A `R1 = 0xDEADBEEF`
B `R3 = R2 + R1`
C `R1 = *R5`
D `R7 = 0xFEEDFACE`
E `R1 = R1 - 1`

BAR ( )

F `R7 = R3 + R2`
A `R1 = 0xDEADBEEF`
B `R3 = R2 + R1`
C `R1 = *R5`
G `R7 = 0xFACEFEED`
A `R1 = 0xDEADBEEF`
B `R3 = R2 + R1`
C `R1 = *R5`
E `R1 = R1 - 1`

String Encoding

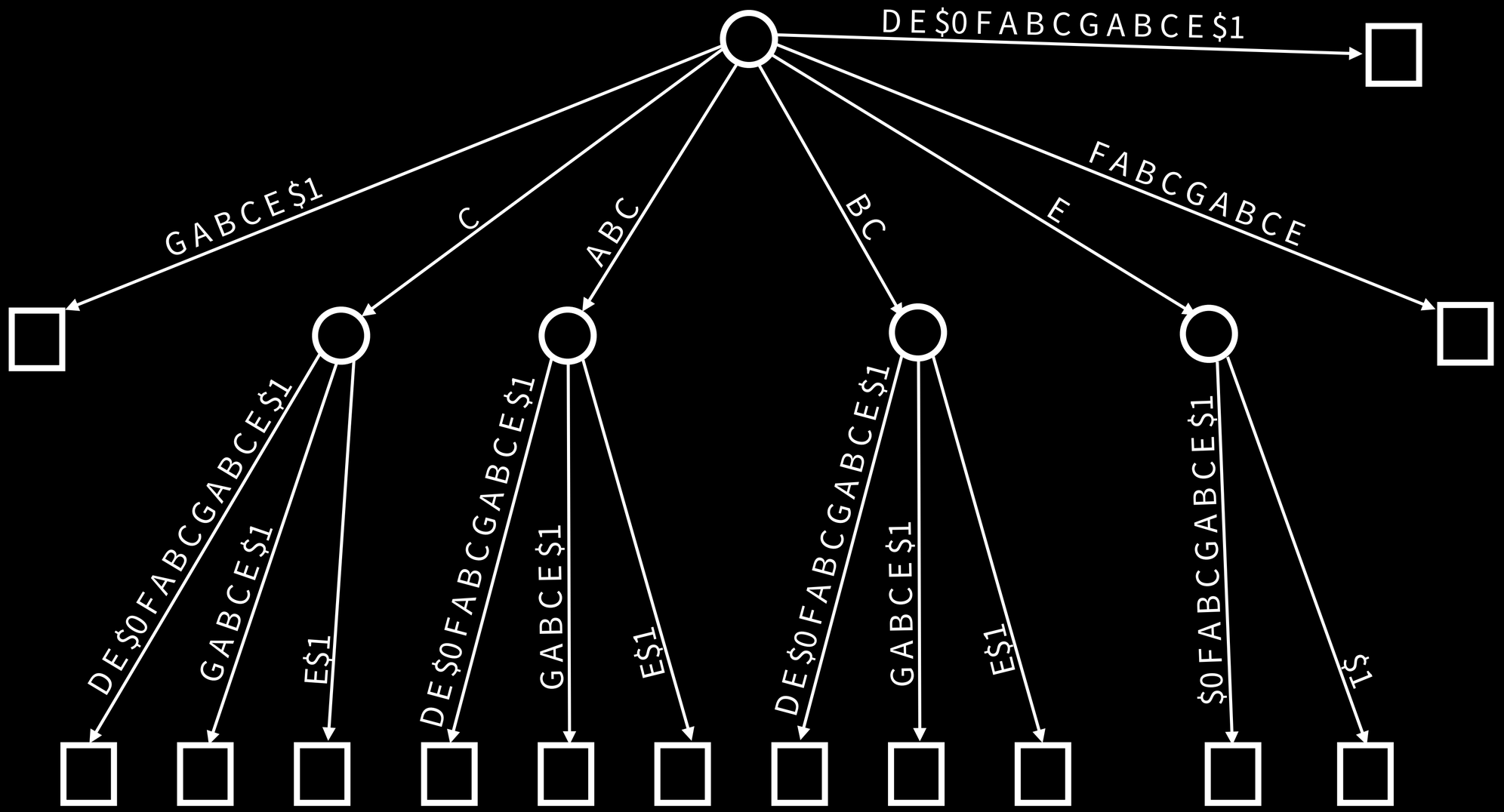
A B C D E $0 F A B C G A B C E $1

String Encoding

A B C D E $0 F A B C G A B C E $1

Find Candidates

A B C D E $0 F A B C G A B C E $1

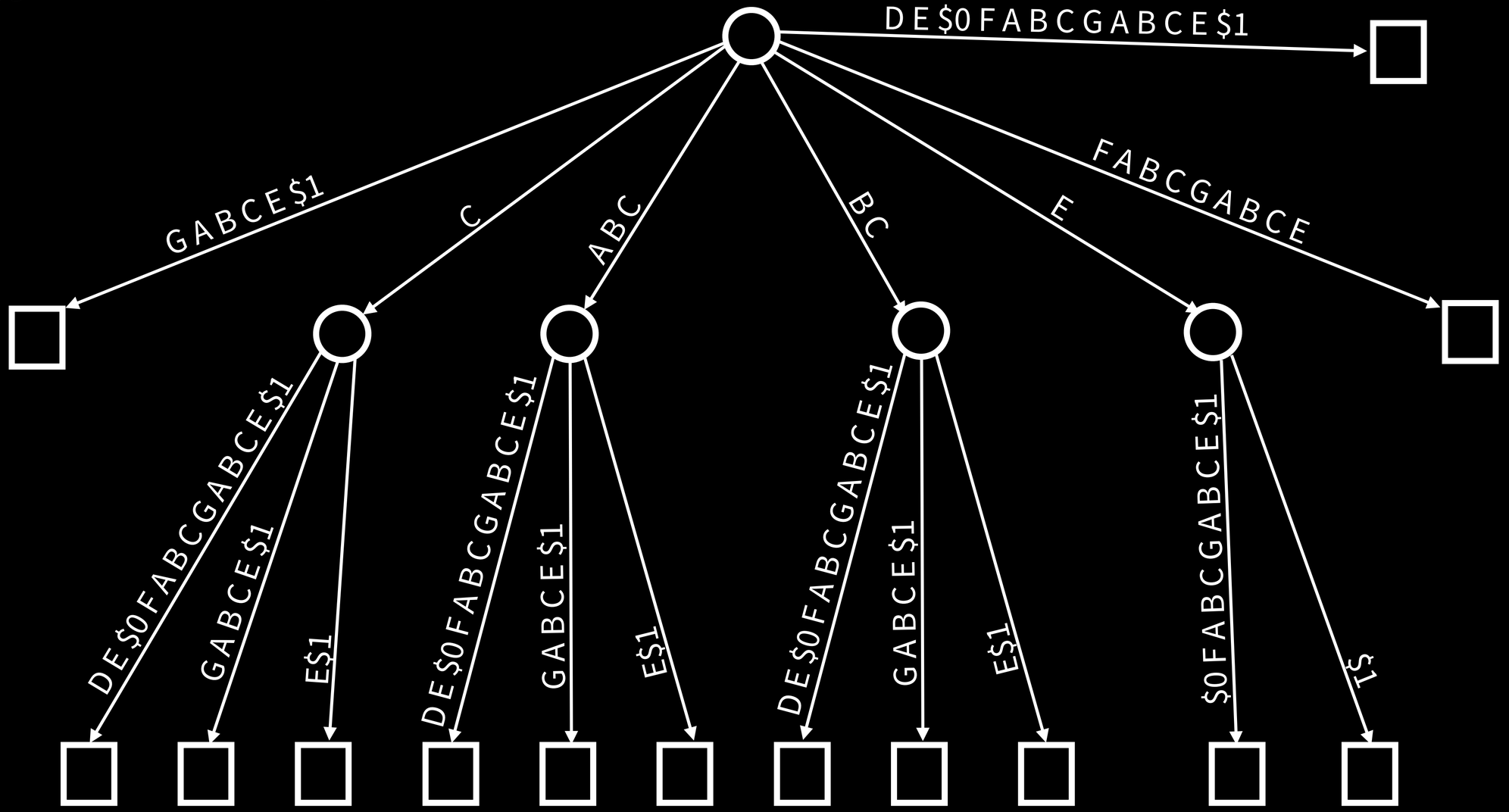


A B C D E $0 F A B C G A B C E $1

Longest repeated substring None

Length 0

Occurrences 0

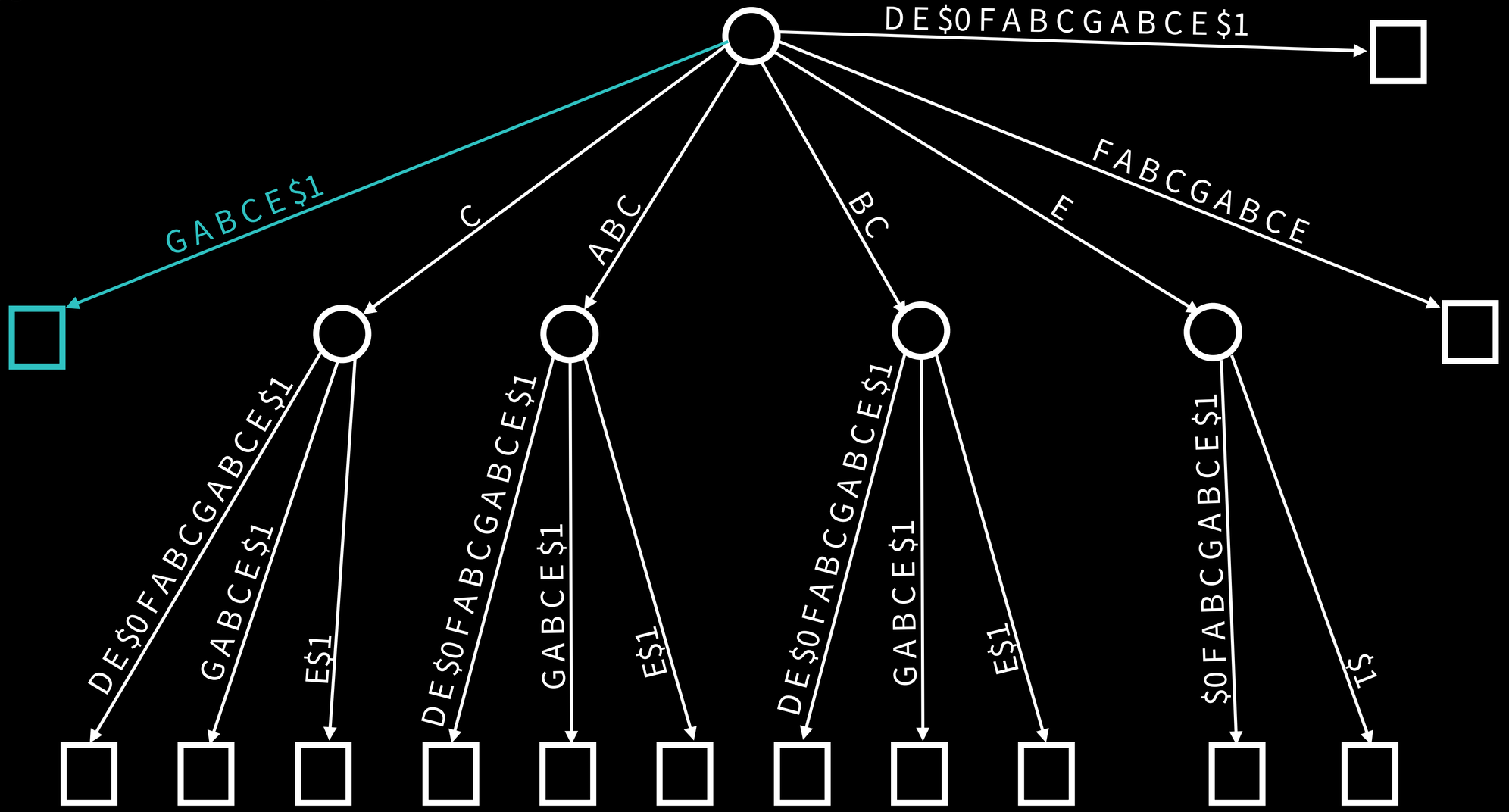


A B C D E $0 F A B C G A B C E $1

Longest repeated substring None

Length 0

Occurrences 0

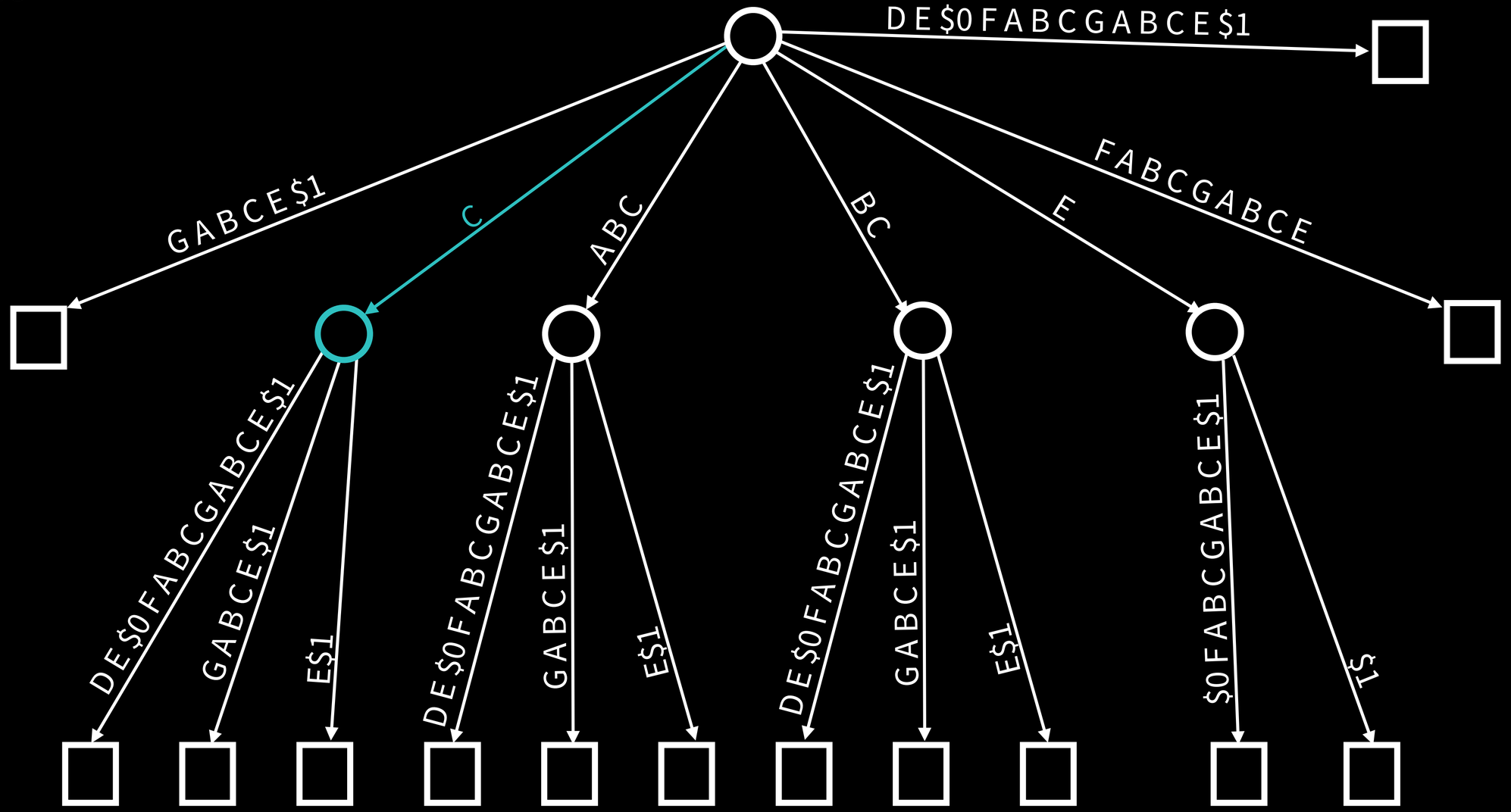


A B C D E $0 F A B C G A B C E $1

Longest repeated substring C

Length 1

Occurrences 3

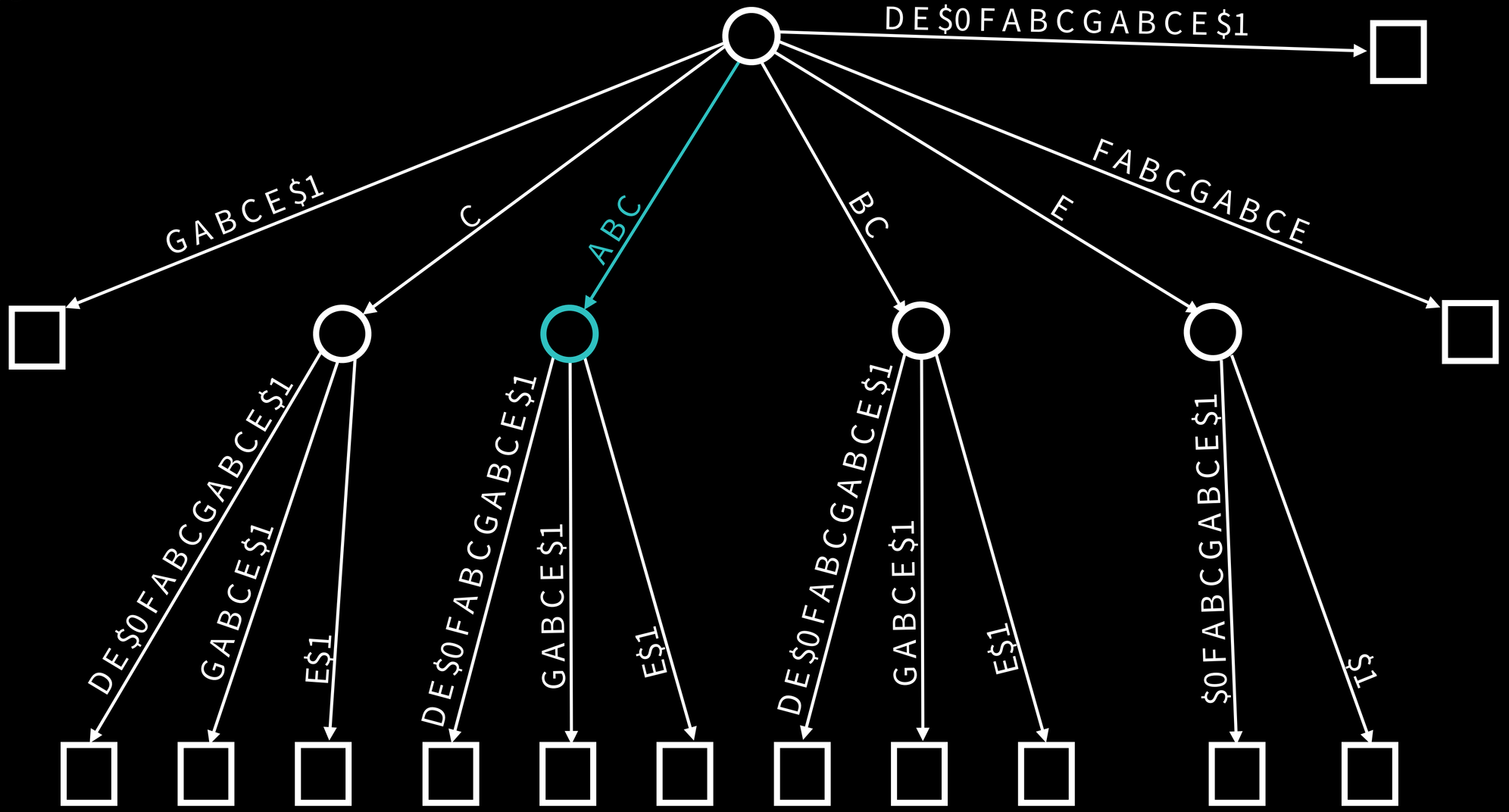


A B C D E $0 F A B C G A B C E $1

Longest repeated substring ABC

Length 3

Occurrences 3

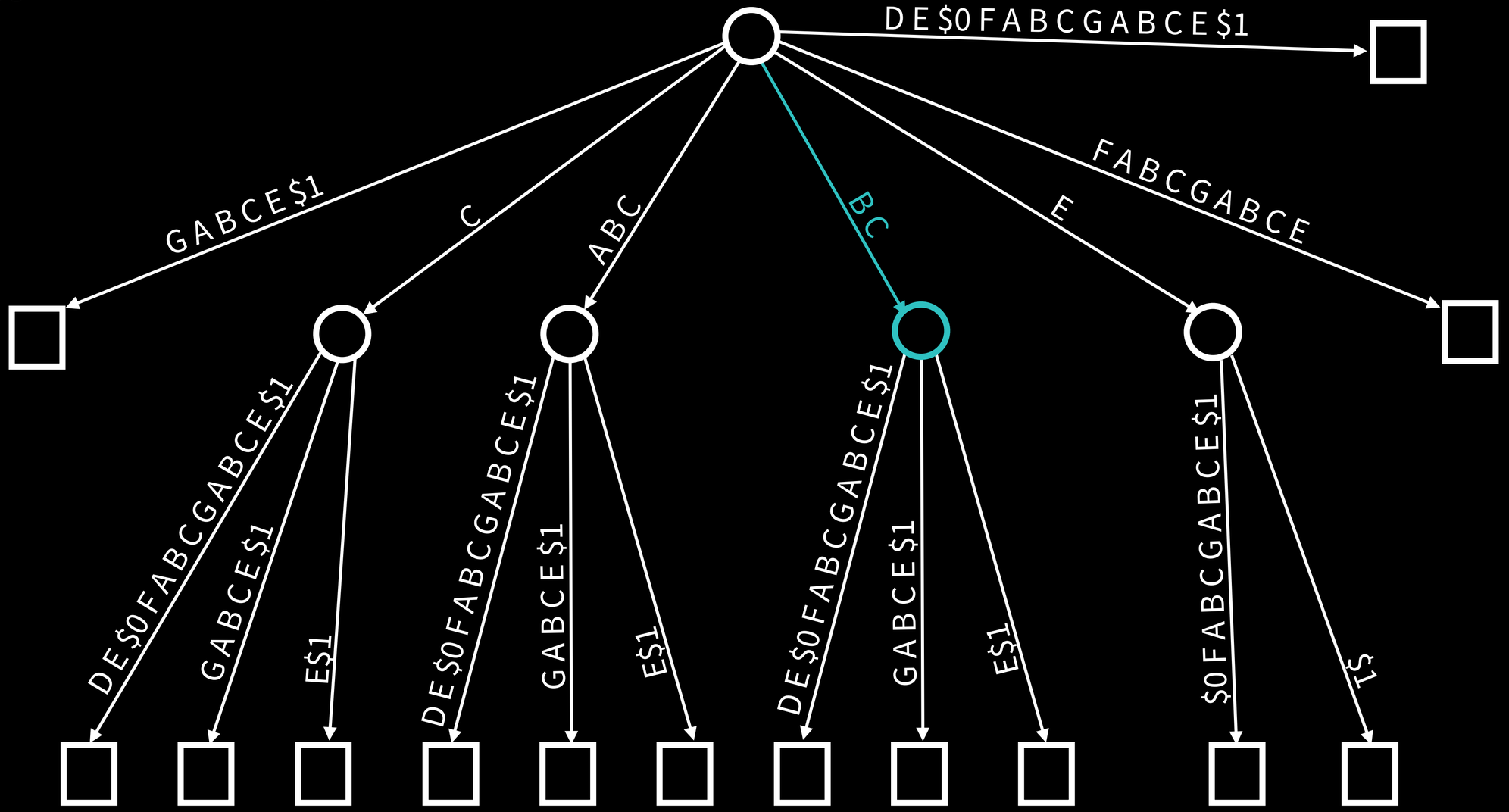


A B C D E $0 F A B C G A B C E $1

Longest repeated substring ABC

Length 3

Occurrences 3

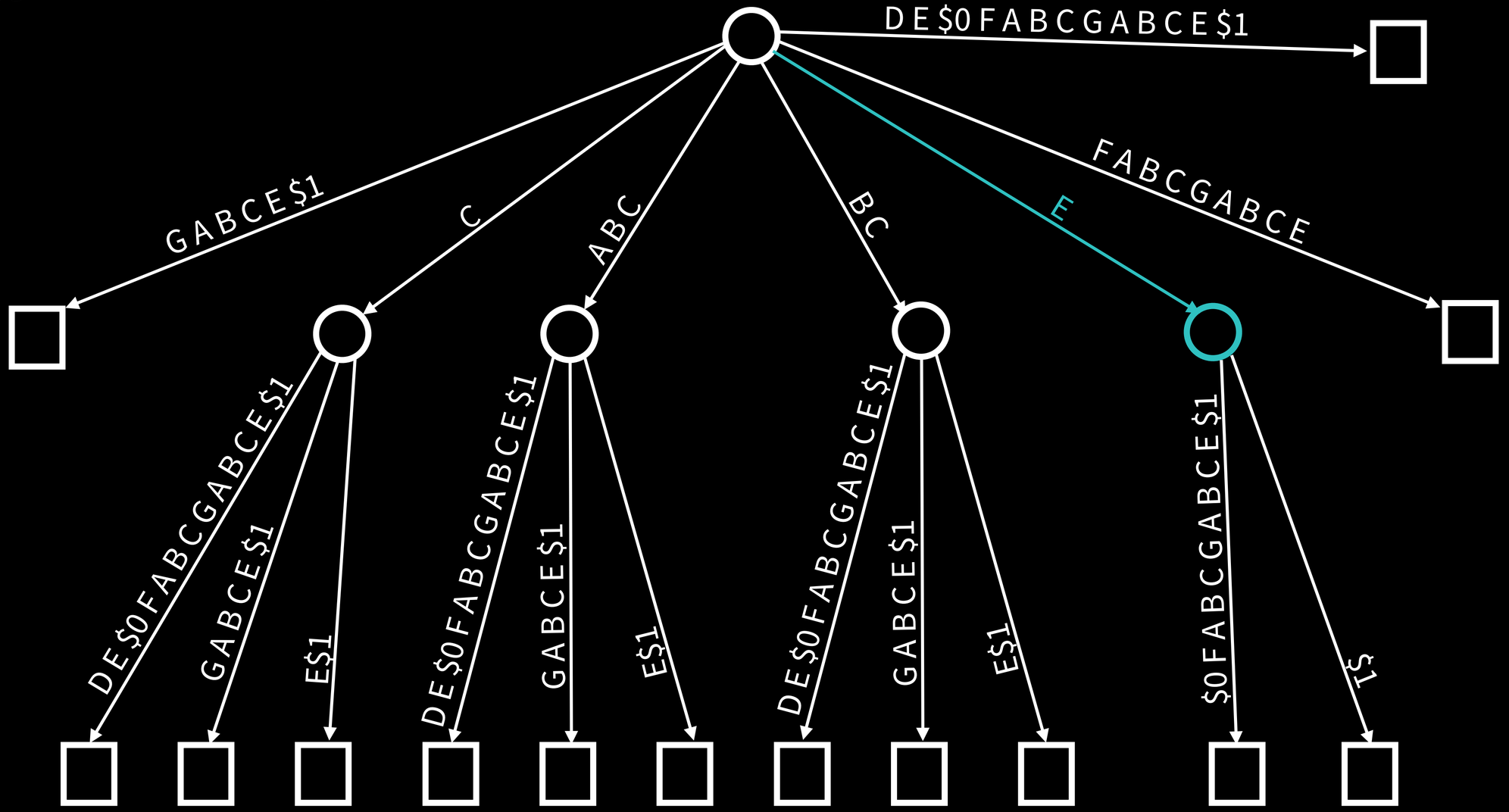


A B C D E $0 F A B C G A B C E $1

Longest repeated substring ABC

Length 3

Occurrences 3

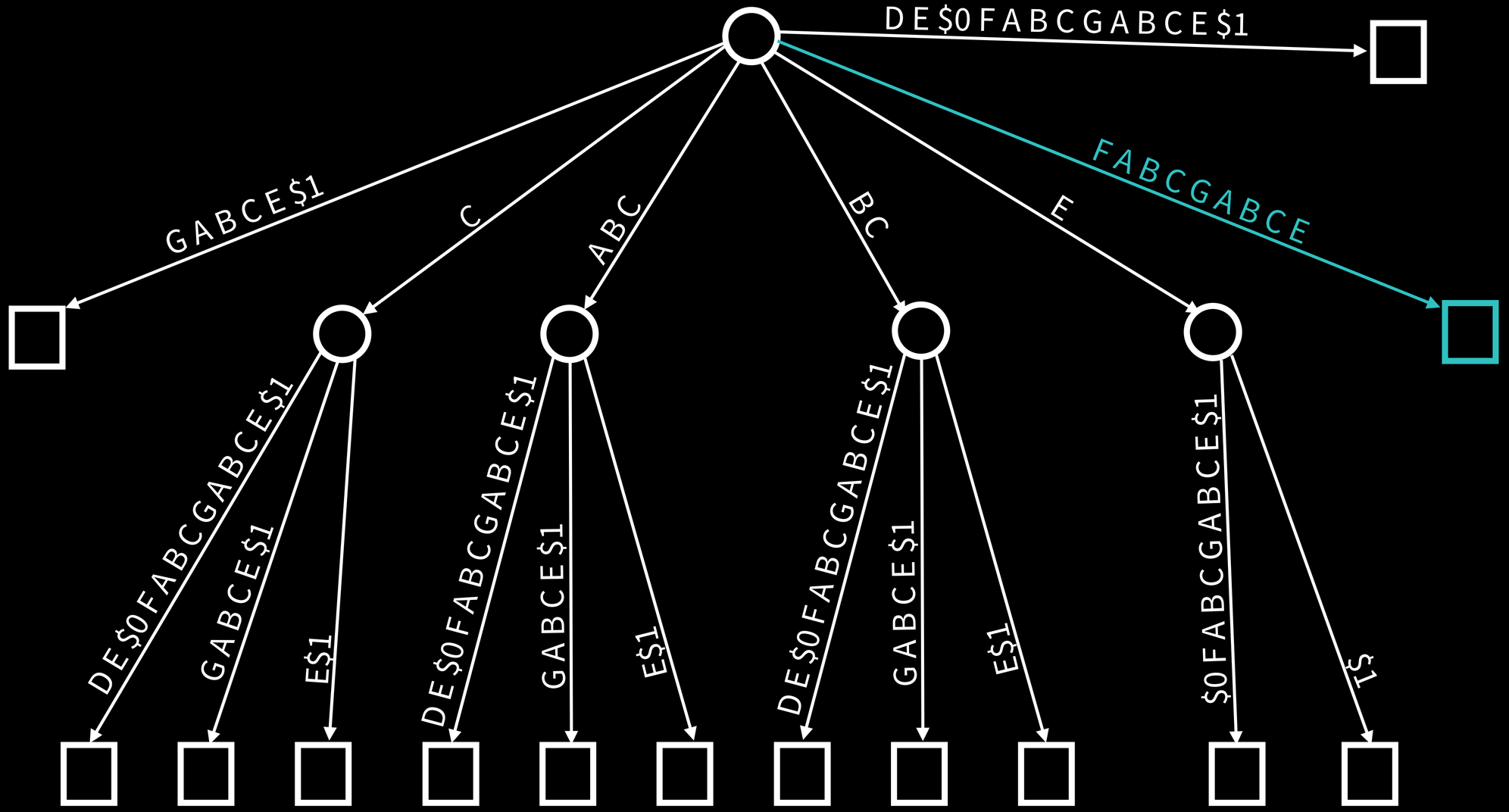


A B C D E $0 F A B C G A B C E $1

Longest repeated substring ABC

Length 3

Occurrences 3

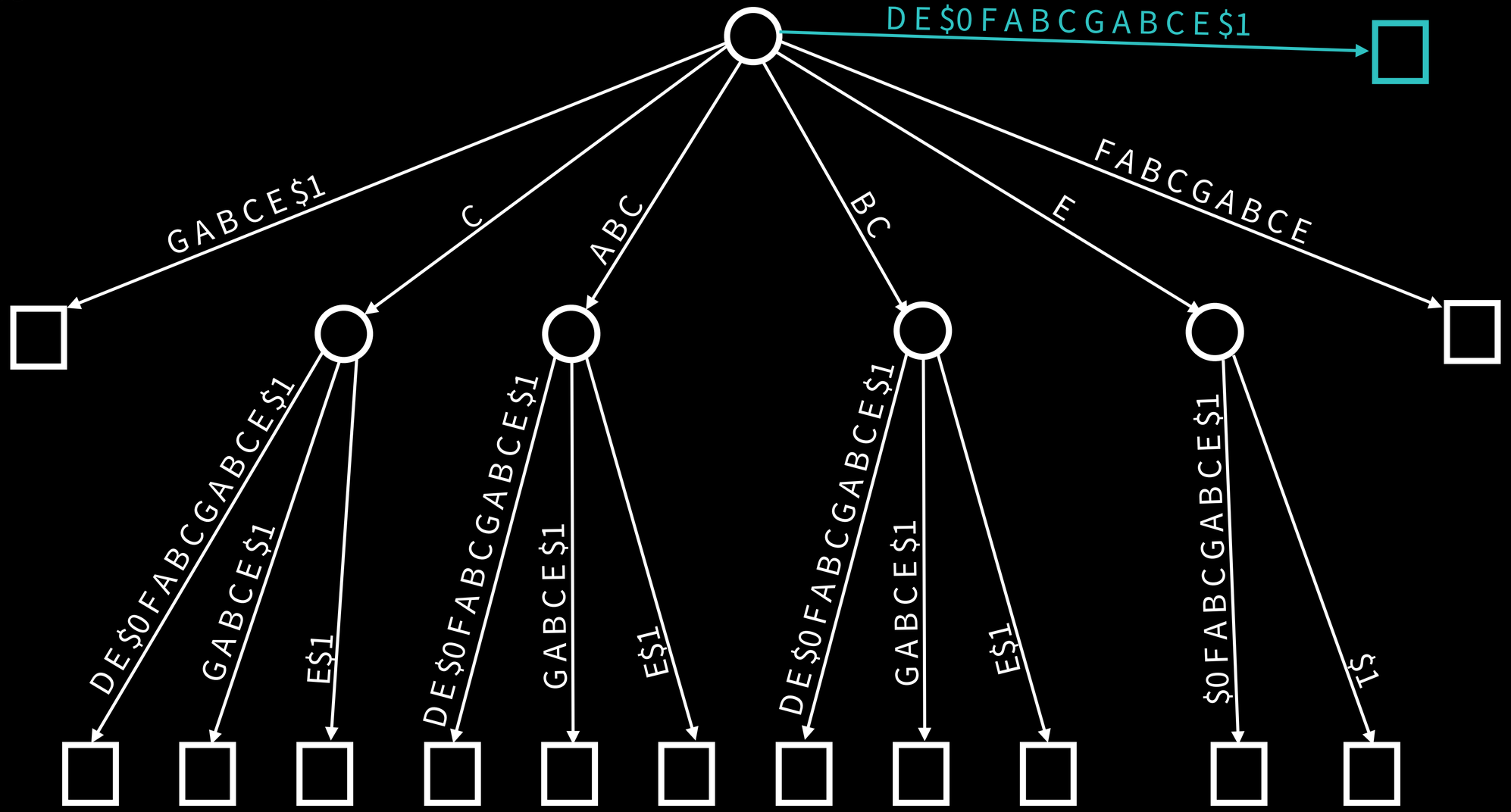


A B C D E $0 F A B C G A B C E $1

Longest repeated substring ABC

Length 3

Occurrences 3

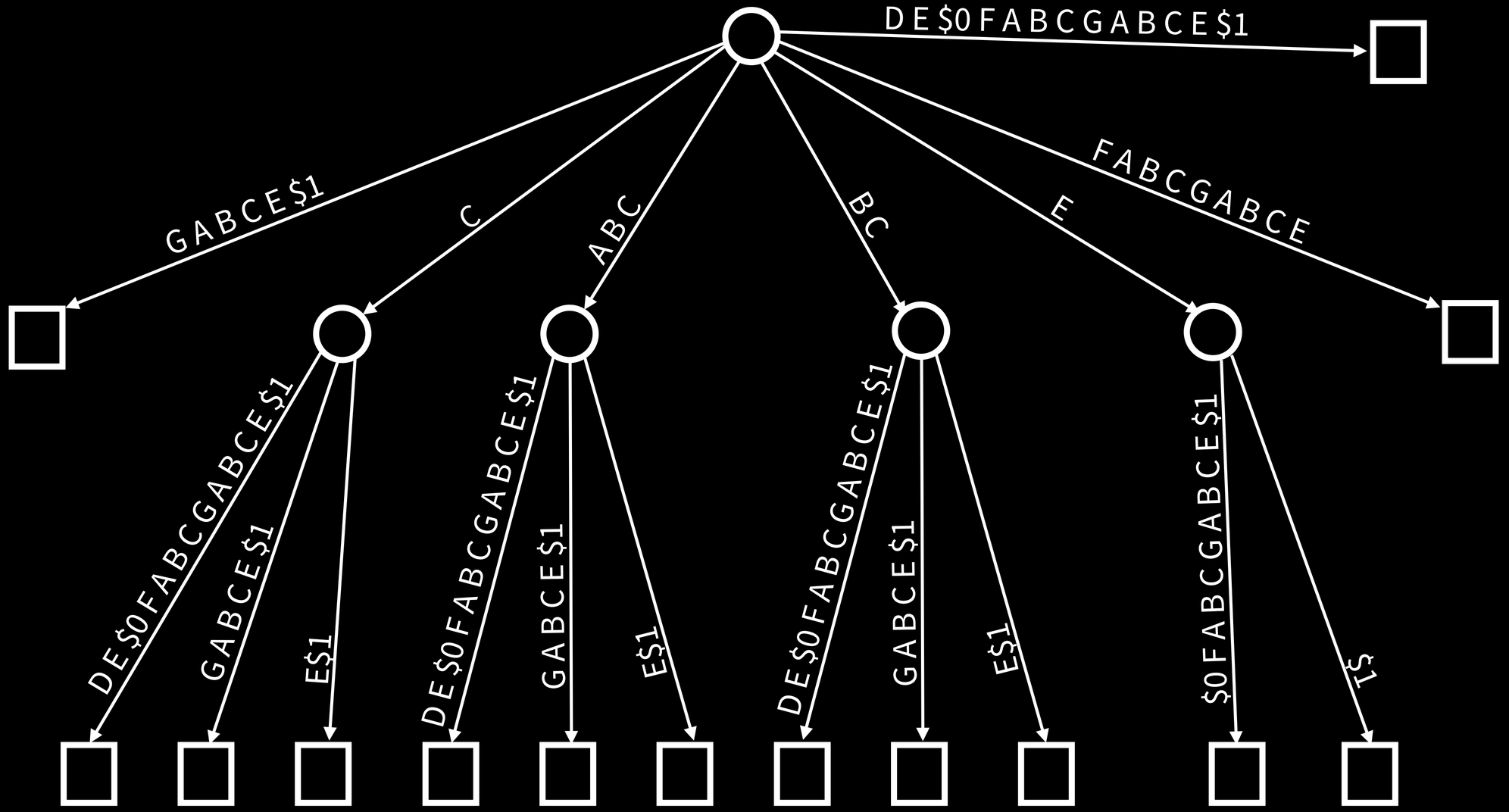


A B C D E $0 F A B C G A B C E $1

Longest repeated substring ABC

Length 3

Occurrences 3

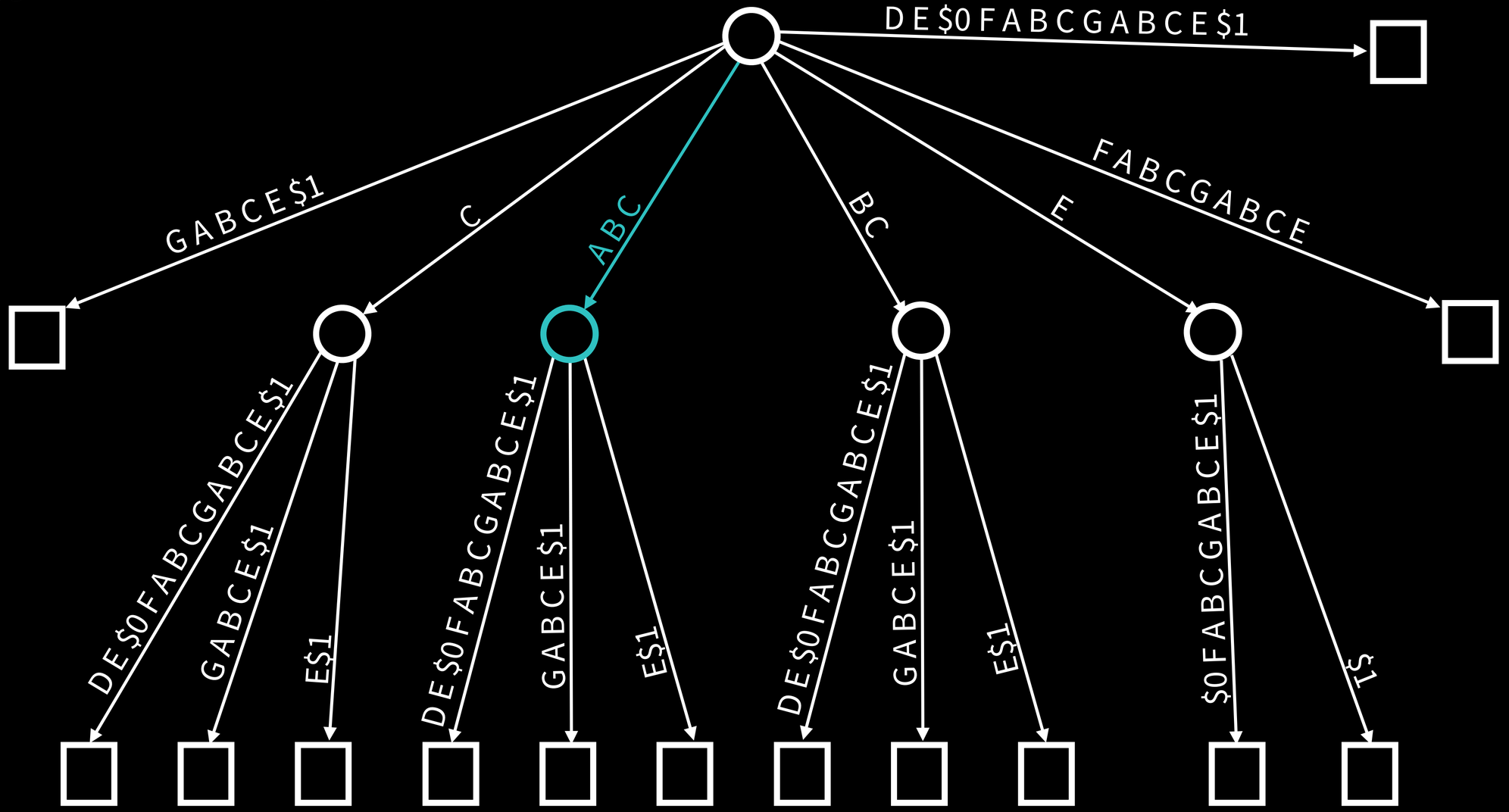


A B C D E $0 F A B C G A B C E $1

Longest repeated substring ABC

Length 3

Occurrences 3

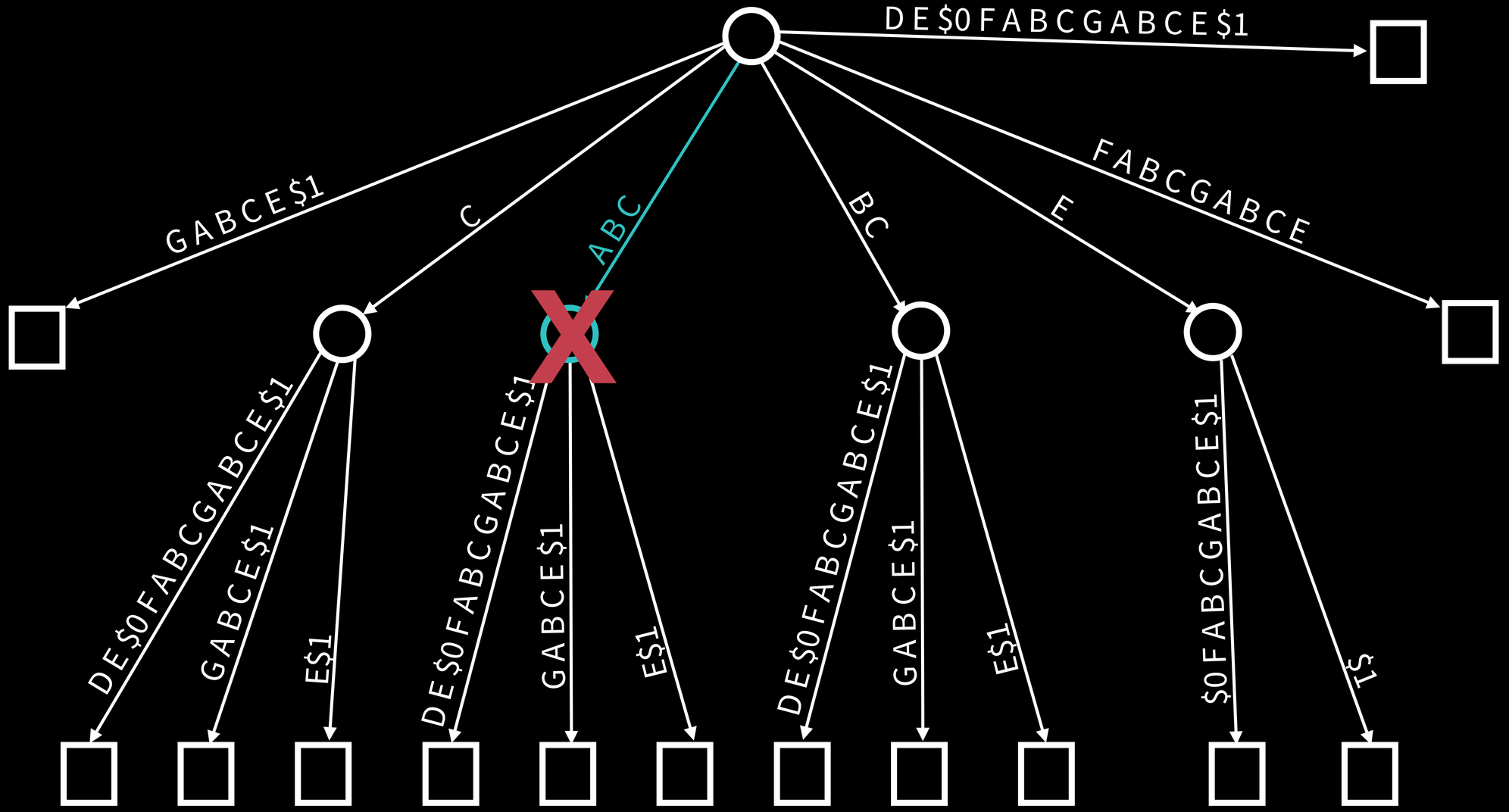


A B C D E $0 F A B C G A B C E $1

Longest repeated substring ABC

Length 3

Occurrences 3

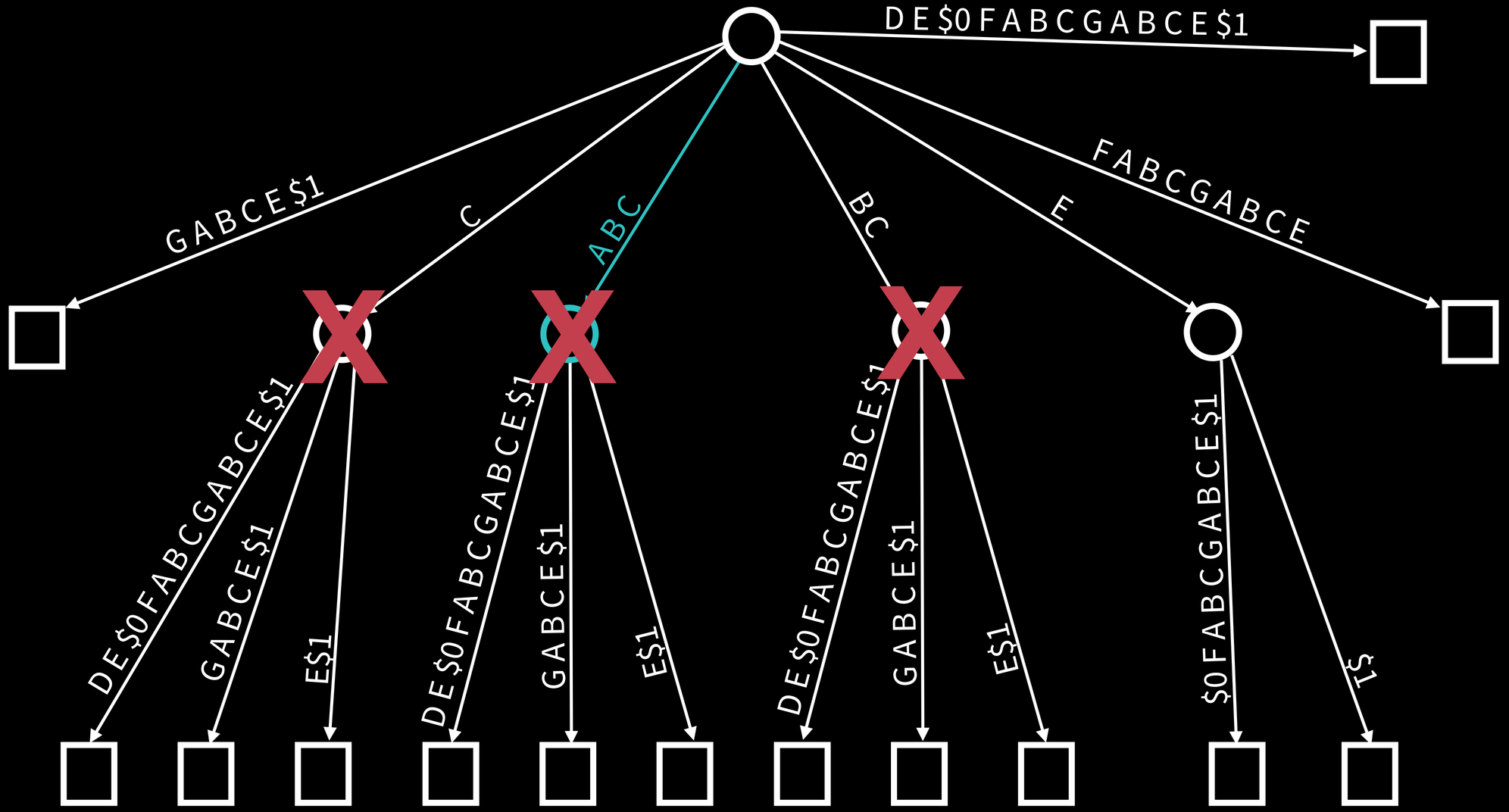


A B C D E $0 F A B C G A B C E $1

Longest repeated substring ABC

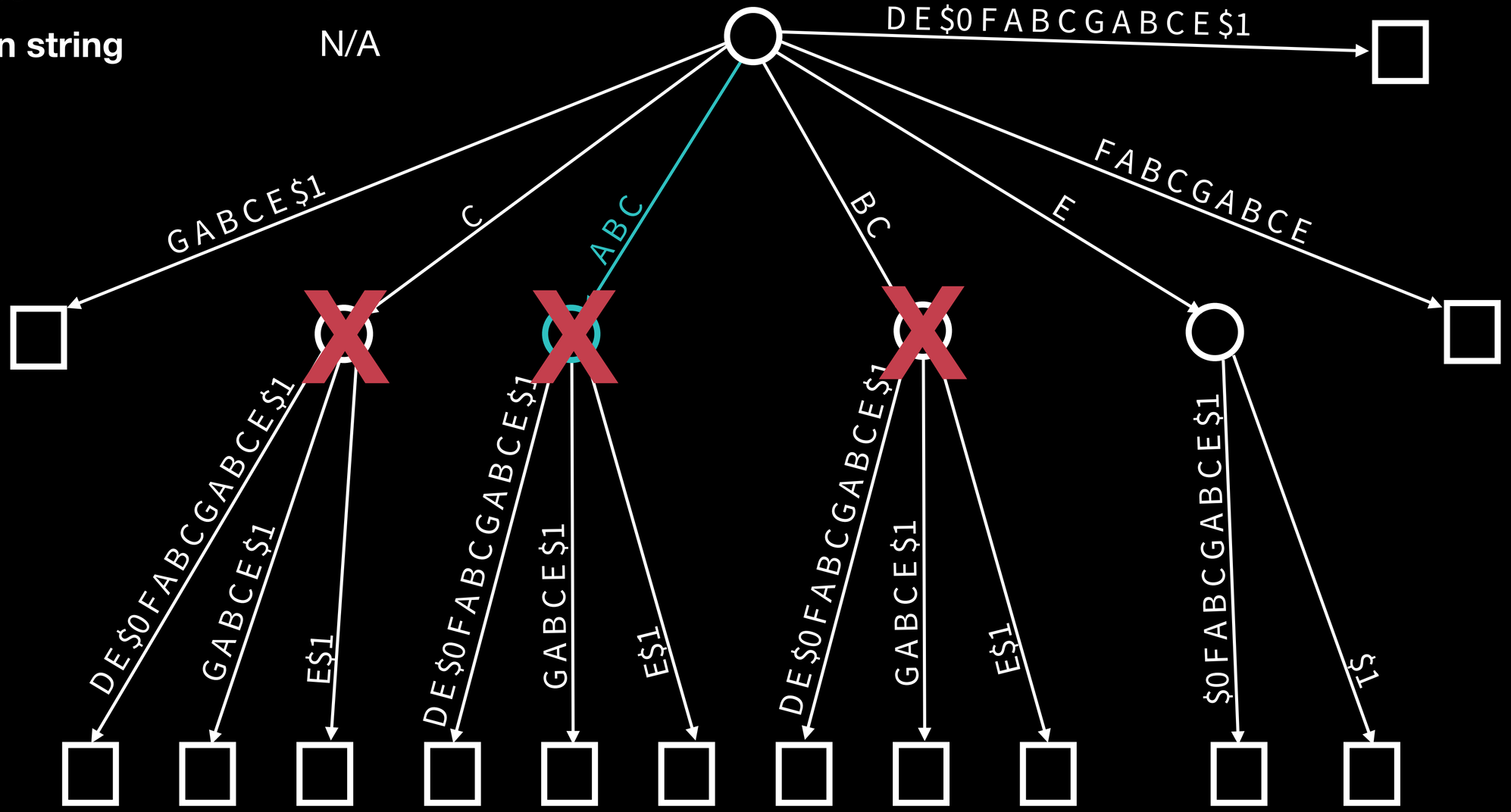
Length 3

Occurrences 3



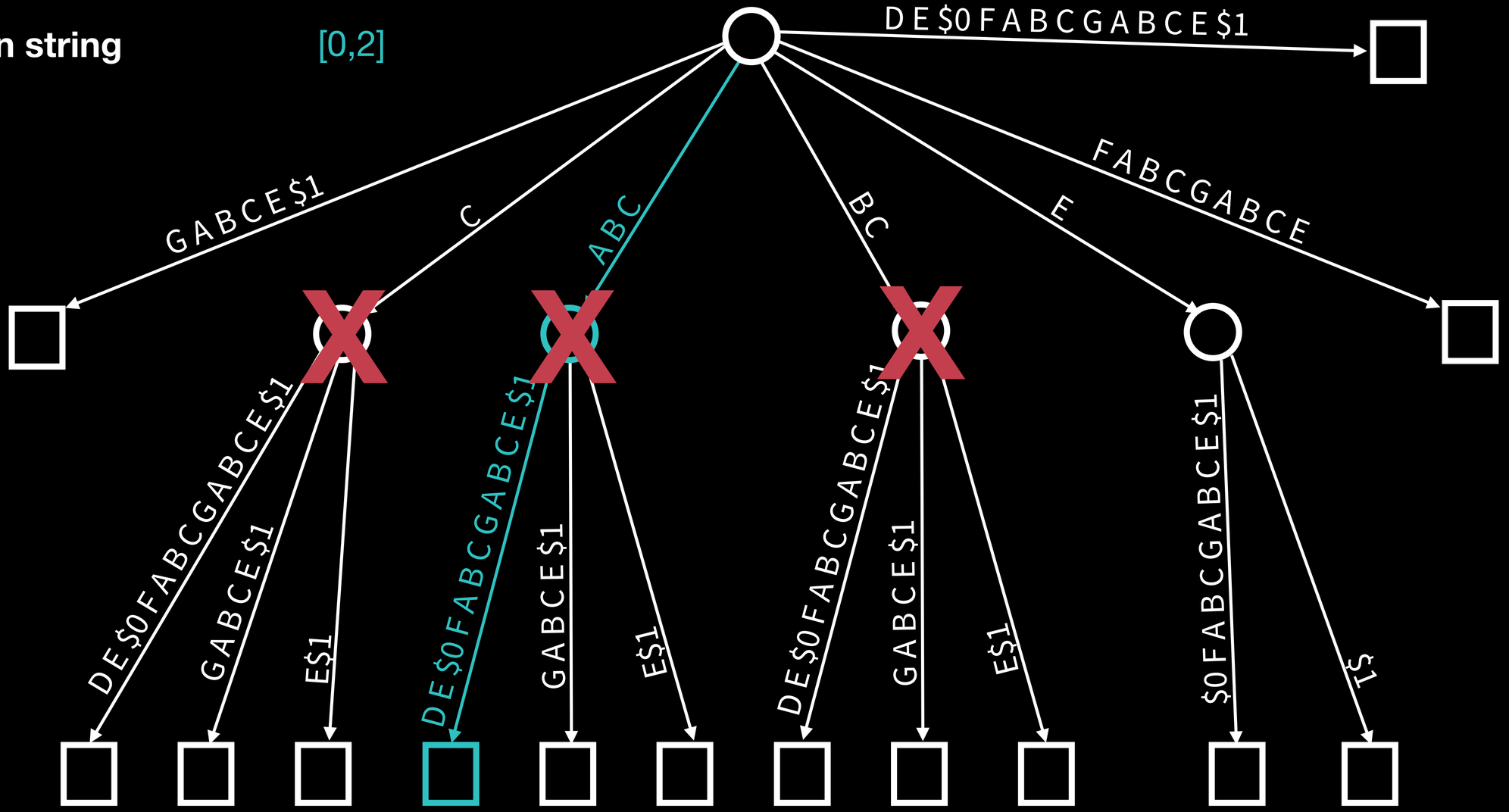
A B C D E $0 F A B C G A B C E $1

Longest repeated substring	ABC
Length	3
Occurrences	3
Locations in string	N/A



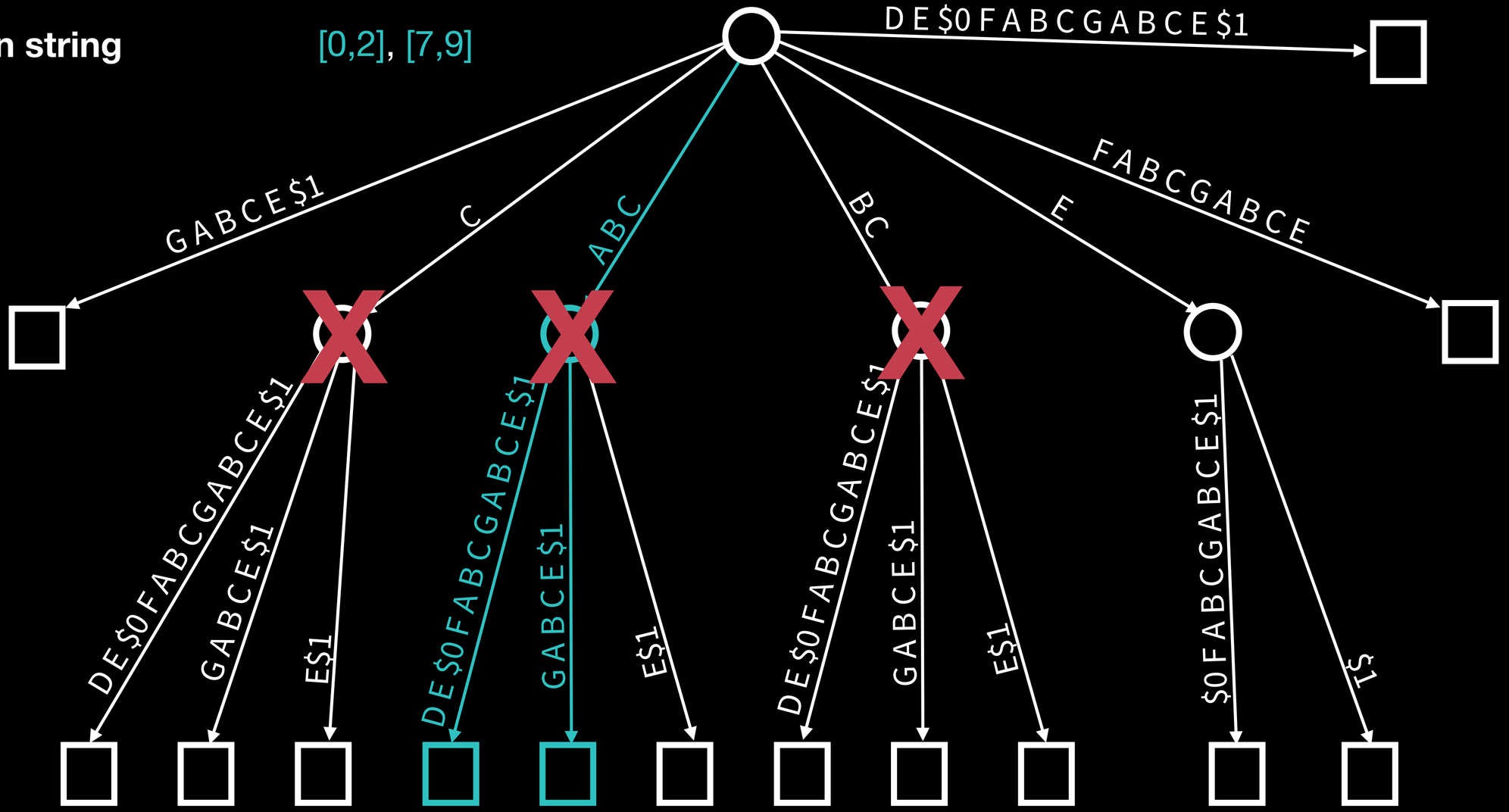
A B C D E $0 F A B C G A B C E $1

Longest repeated substring	ABC
Length	3
Occurrences	3
Locations in string	[0,2]



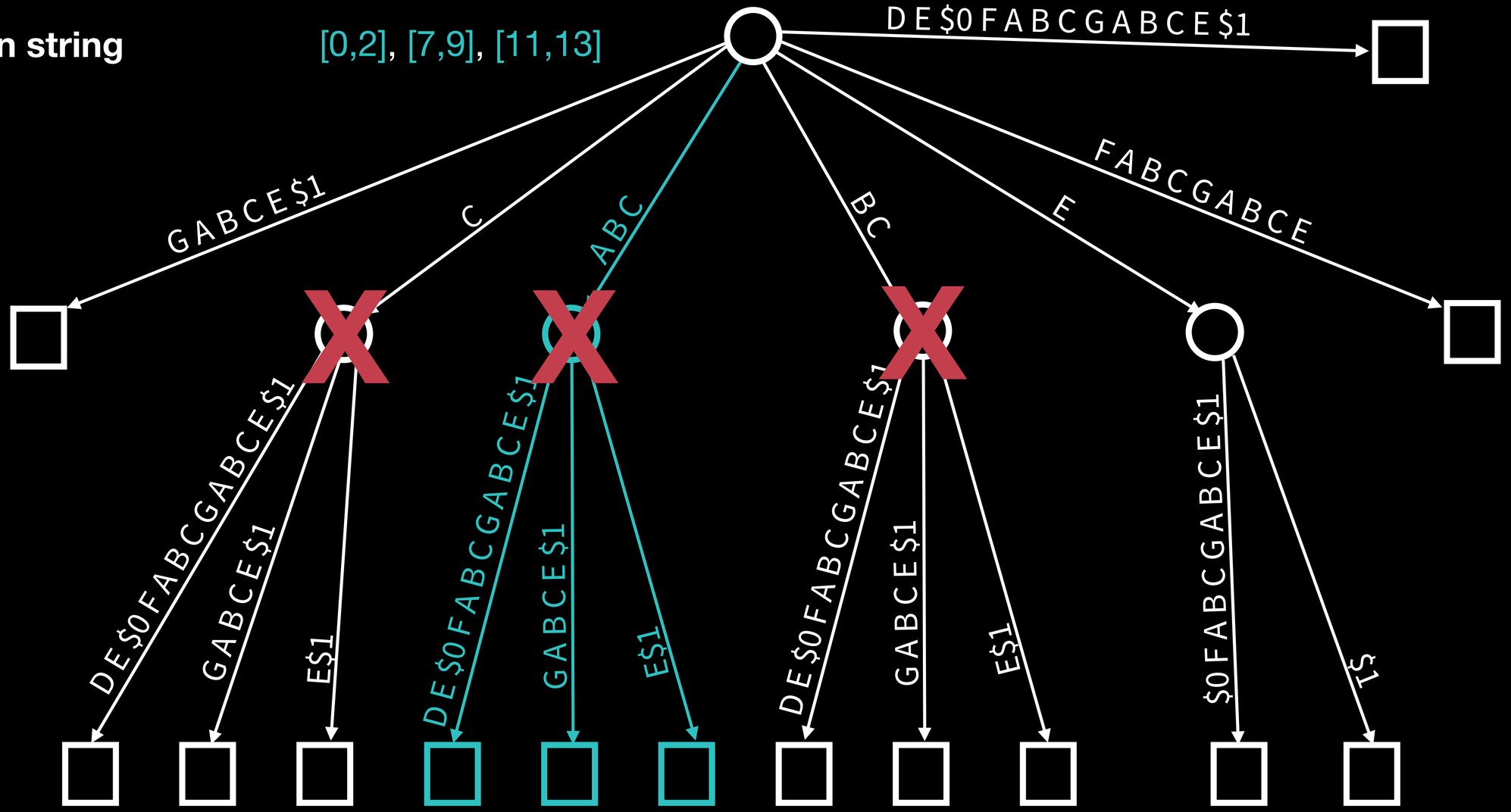
A B C D E $0 F A B C G A B C E $1

Longest repeated substring	ABC
Length	3
Occurrences	3
Locations in string	[0,2], [7,9]



A B C D E $0 F A B C G A B C E $1

Longest repeated substring	ABC
Length	3
Occurrences	3
Locations in string	[0,2], [7,9], [11,13]



Outlining

Outlining ABC

Longest repeated substring	ABC
Length	3
Occurrences	3
Locations in string	[0,2], [7,9], [11,13]

A B C D E $0 F A B C G A B C E $1

FOO ( )

A `R1 = 0xDEADBEEF`

B `R3 = R2 + R1`

C `R1 = *R5`

D `R7 = 0xFEEDFACE`

E `R1 = R1 - 1`

BAR ( )

F `R7 = R3 + R2`

A `R1 = 0xDEADBEEF`

B `R3 = R2 + R1`

C `R1 = *R5`

G `R7 = 0xFACEFEED`

A `R1 = 0xDEADBEEF`

B `R3 = R2 + R1`

C `R1 = *R5`

E `R1 = R1 - 1`

Insert Function

Longest repeated substring	ABC
Length	3
Occurrences	3
Locations in string	[0,2], [7,9], [11,13]

A B C D E $0 F A B C G A B C E $1

OUTLINED ( )

A R1 = 0xDEADBEEF

B R3 = R2 + R1

C R1 = *R5

FOO ( )

A R1 = 0xDEADBEEF

B R3 = R2 + R1

C R1 = *R5

D R7 = 0xFEEDFACE

E R1 = R1 - 1

BAR ( )

F R7 = R3 + R2

A R1 = 0xDEADBEEF

B R3 = R2 + R1

C R1 = *R5

G R7 = 0xFACEFEED

A R1 = 0xDEADBEEF

B R3 = R2 + R1

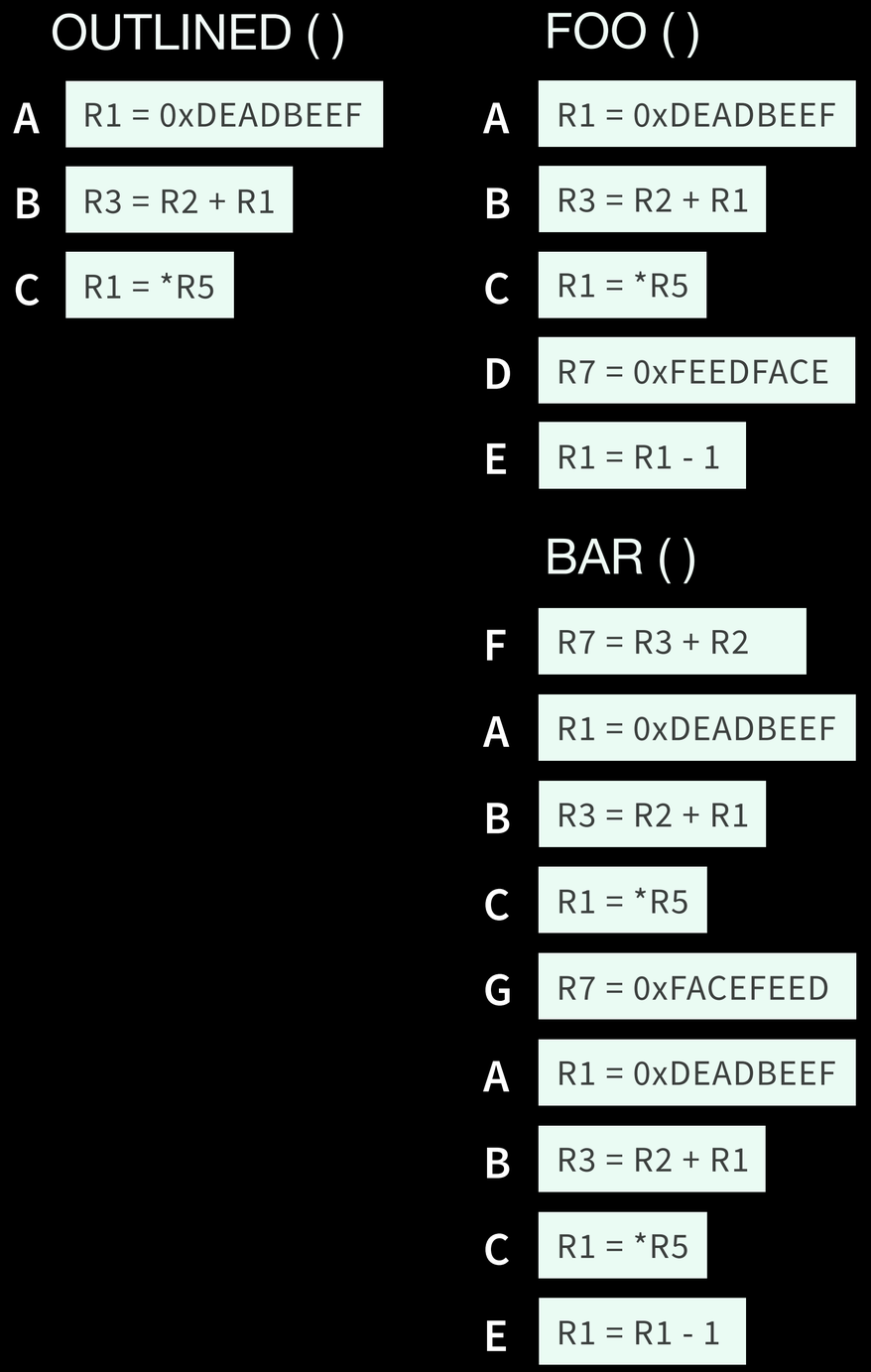
C R1 = *R5

E R1 = R1 - 1

Insert Calls

Longest repeated substring	ABC
Length	3
Occurrences	3
Locations in string	[0,2], [7,9], [11,13]

A B C D E $0 F A B C G A B C E $1



Insert Calls

Longest repeated substring	ABC
Length	3
Occurrences	3
Locations in string	[0,2], [7,9], [11,13]

A B C D E $0 F A B C G A B C E $1

Insert Calls

Longest repeated substring	ABC
Length	3
Occurrences	3
Locations in string	[0,2], [7,9], [11,13]

X D E $0 F A B C G A B C E $1

OUTLINED ( )

A R1 = 0xDEADBEEF

B R3 = R2 + R1

C R1 = *R5

FOO ( )

X call OUTLINED( )

D R7 = 0xFEEDFACE

E R1 = R1 - 1

BAR ( )

F R7 = R3 + R2

A R1 = 0xDEADBEEF

B R3 = R2 + R1

C R1 = *R5

G R7 = 0xFACEFEED

A R1 = 0xDEADBEEF

B R3 = R2 + R1

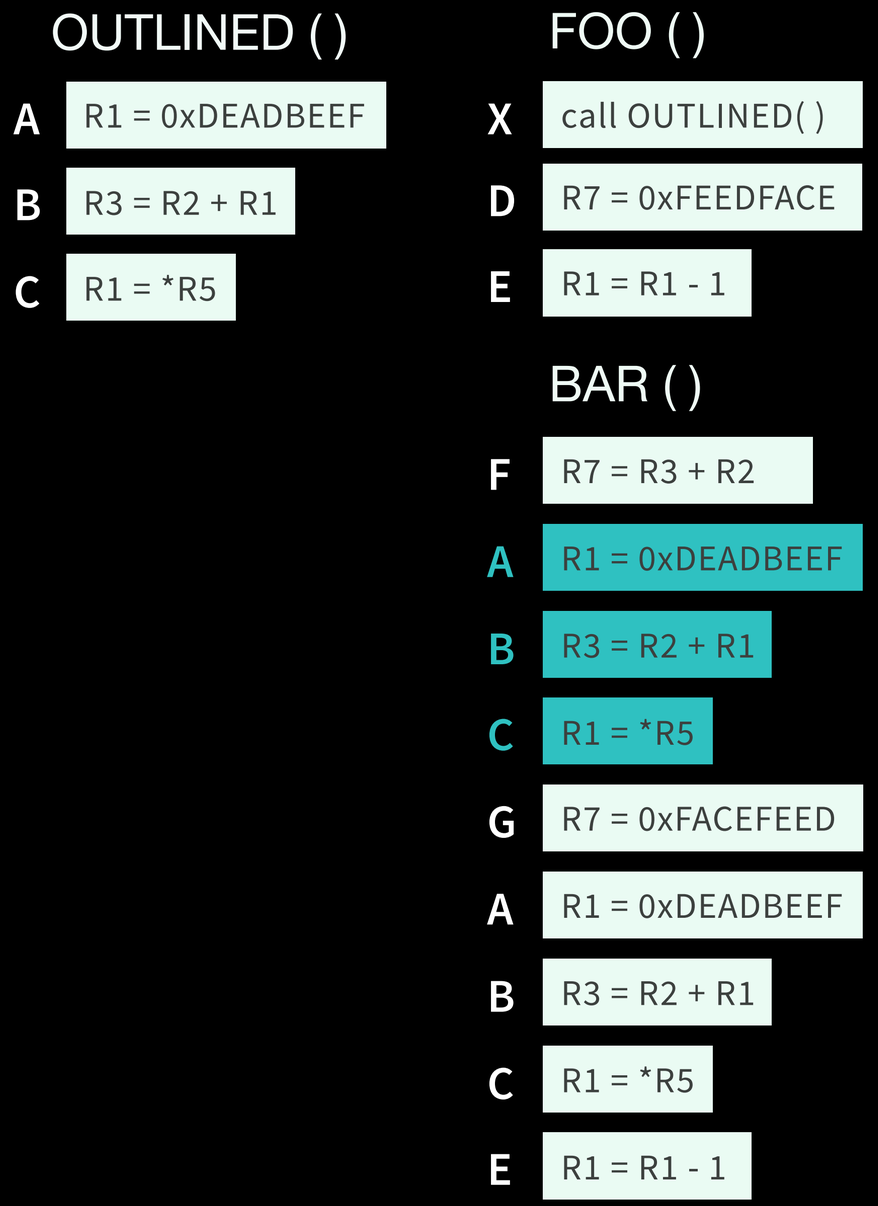
C R1 = *R5

E R1 = R1 - 1

Insert Calls

Longest repeated substring	ABC
Length	3
Occurrences	3
Locations in string	[0,2], [7,9], [11,13]

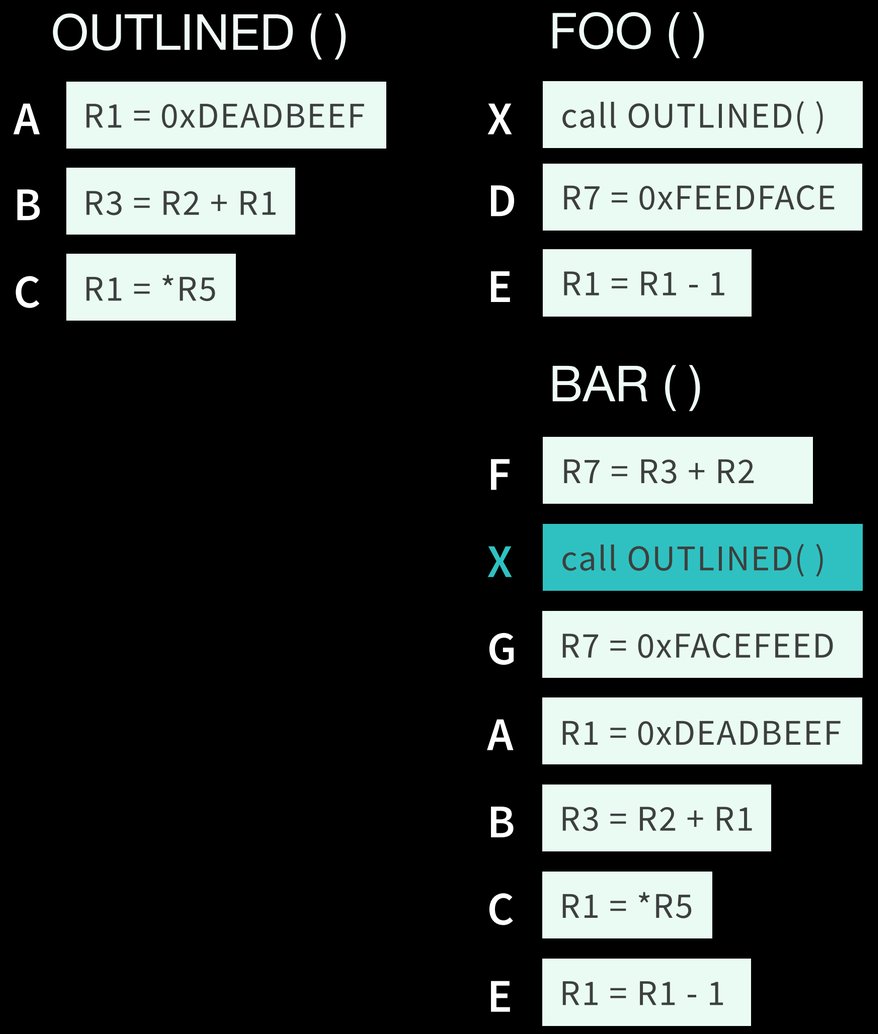
X D E $0 F A B C G A B C E $1



Insert Calls

Longest repeated substring	ABC
Length	3
Occurrences	3
Locations in string	[0,2], [7,9], [11,13]

X D E $0 F X G A B C E $1



Insert Calls

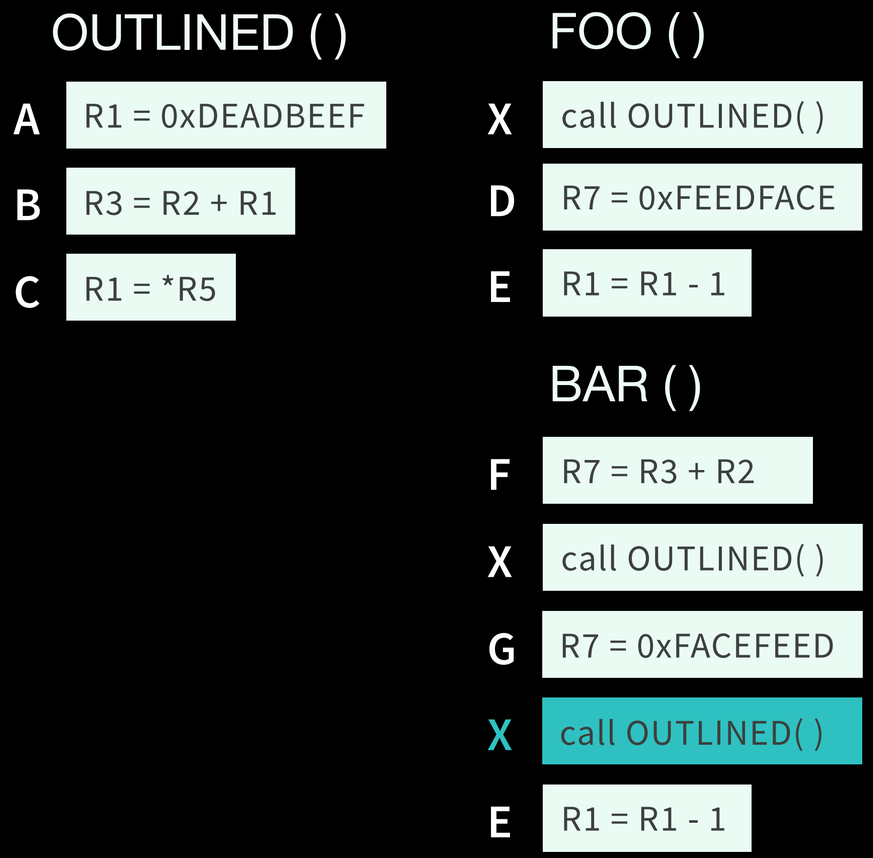
Longest repeated substring	ABC
Length	3
Occurrences	3
Locations in string	[0,2], [7,9], [11,13]

X D E $0 F X G A B C E $1

Insert Calls

Longest repeated substring	ABC
Length	3
Occurrences	3
Locations in string	[0,2], [7,9], [11,13]

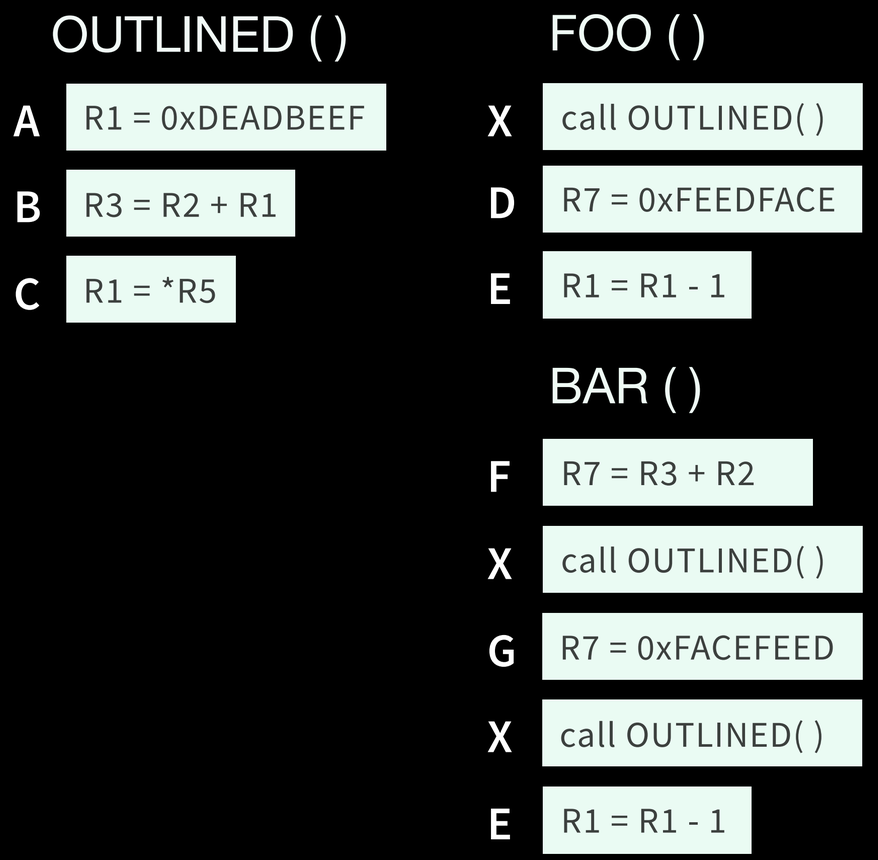
X D E $0 F X G X E $1



Done!

Longest repeated substring	ABC
Length	3
Occurrences	3
Locations in string	[0,2], [7,9], [11,13]

X D E $0 F X G X E $1



Limitations

Unsafe Sequences

```
…
  subq  $16, %rsp
  movl  $0, -4(%rbp)
  movl  $99, -8(%rbp)
  cmpl  $0, -8(%rbp)
  jle   LEQ_ZERO
  movl  -8(%rbp), %eax
  addl  $1, %eax
  movl  %eax, -8(%rbp)
  jmp   PRINT_RESULT
LEQ_ZERO:
  movl  $0, -8(%rbp)
PRINT_RESULT:
  leaq  STRING(%rip), %rdi
  movl  -8(%rbp), %esi
  movb  $0, %al
  callq _printf
…
```

```
…
  subq  $16, %rsp
  movl  $0, -4(%rbp)
  movl  $99, -8(%rbp)
  cmpl  $0, -8(%rbp)
  jle   LEQ_ZERO
  movl  -8(%rbp), %eax
  addl  $1, %eax
  movl  %eax, -8(%rbp)
  jmp   PRINT_RESULT
LEQ_ZERO:
  movl  $0, -8(%rbp)
PRINT_RESULT:
  leaq  STRING(%rip), %rdi
  movl  -8(%rbp), %esi
  movb  $0, %al
  callq _printf
…
```

```
…
  subq  $16, %rsp
  movl  $0, -4(%rbp)
  movl  $99, -8(%rbp)
  cmpl  $0, -8(%rbp)
  jle   LEQ_ZERO
  movl  -8(%rbp), %eax
  addl  $1, %eax
  movl  %eax, -8(%rbp)
  jmp   PRINT_RESULT
LEQ_ZERO:
  movl  $0, -8(%rbp)
PRINT_RESULT:
  leaq  STRING(%rip), %rdi
  movl  -8(%rbp), %esi
  movb  $0, %al
  callq _printf
…
```

```
 …
   subq  $16, %rsp
   movl  $0, -4(%rbp)
   movl  $99, -8(%rbp)
   callq OUTLINED
PRINT_RESULT:
   leaq  STRING(%rip), %rdi
   movl  -8(%rbp), %esi
   movb  $0, %al
   callq _printf
 …
```

```
OUTLINED:
   cmpl  $0, -8(%rbp)
   jle   LEQ_ZERO
   movl  -8(%rbp), %eax
   addl  $1, %eax
   movl  %eax, -8(%rbp)
   jmp   PRINT_RESULT
LEQ_ZERO:
   movl  $0, -8(%rbp)
   retq
```

```
 …
   subq  $16, %rsp
   movl  $0, -4(%rbp)
   movl  $99, -8(%rbp)
   callq OUTLINED
PRINT_RESULT:
   leaq  STRING(%rip), %rdi
   movl  -8(%rbp), %esi
   movb  $0, %al
   callq _printf
 …
```

```
OUTLINED:
   cmpl  $0, -8(%rbp)
   jle   LEQ_ZERO
   movl  -8(%rbp), %eax
   addl  $1, %eax
   movl  %eax, -8(%rbp)
   jmp   PRINT_RESULT
LEQ_ZERO:
   movl  $0, -8(%rbp)
   retq
```

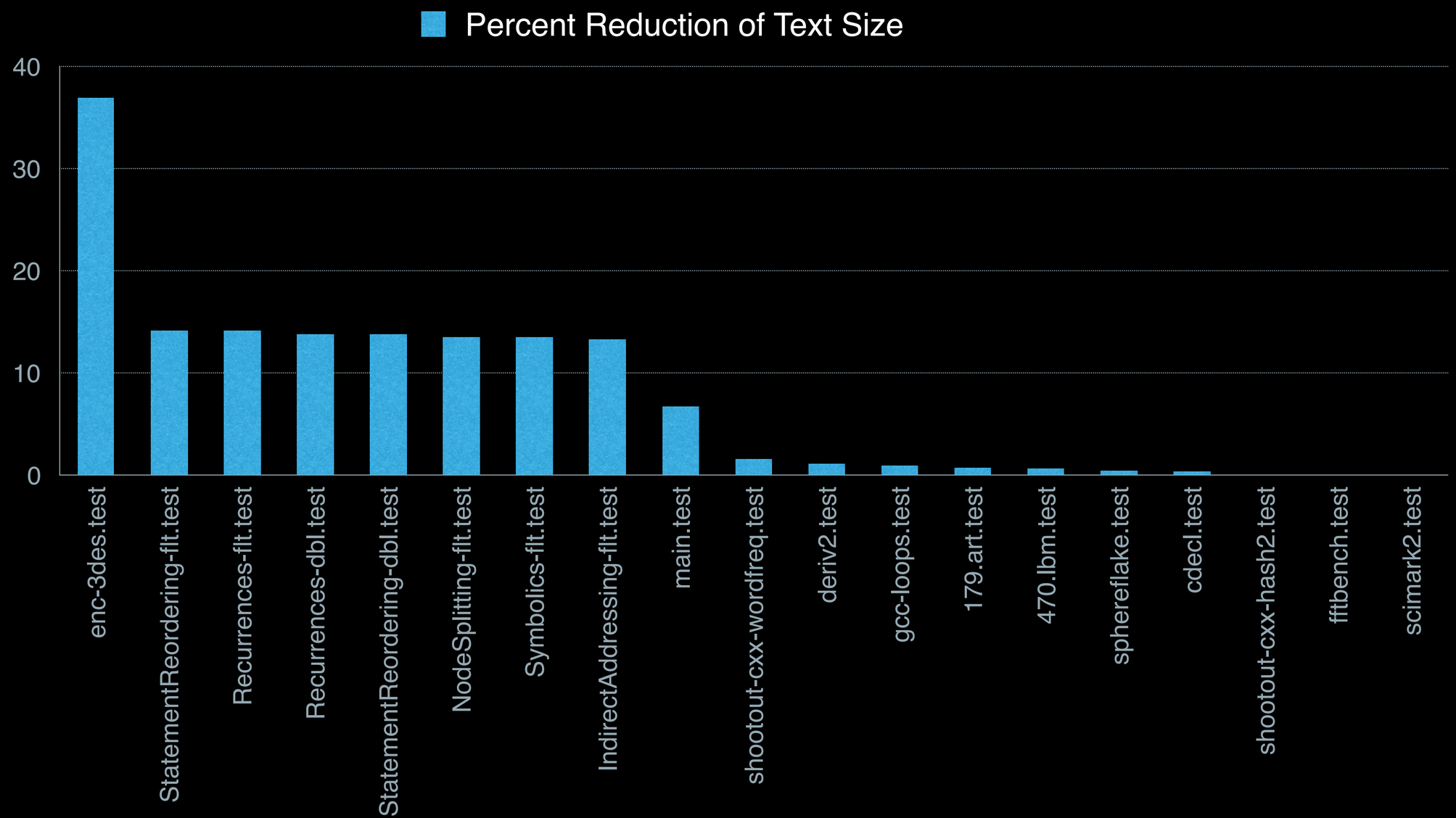
Memory Overhead

Memory Reduction

- Suffix arrays
- Sliding window
- Hierarchical approach

Results

Outlining on the LLVM Test Suite (x86-64)



Outliner-Friendly Programs

- Heavy macro usage
- Manually unrolled loops
- Heavy template usage
- Automatically generated code

Outliner-Unfriendly Programs

- Very small
- Lots of unsafe cases
- Unfortunate register allocation

Future Work

- Performance impact and outlining
- ARM outlining
- IR outlining, pre-register allocation outlining
- String algorithms and code analysis

Summary

- Good preliminary results
- Potential problems can be avoided
- Lots in the future for outlining

RFC: http://lists.llvm.org/pipermail/llvm-dev/2016-August/104170.html

Questions

References

- Ukkonen's algorithm: https://www.cs.helsinki.fi/u/ukkonen/SuffixT1withFigs.pdf
- Suffix Trees and Suffix Arrays: http://web.cs.iastate.edu/~cs548/suffix.pdf
- Generalized suffix trees for biological sequence data: applications and implementation: http://ieeexplore.ieee.org/document/323593/?reload=true&arnumber=323593