Encoding

- Partition the input string into blocks. Each block consists of a maximum prefix of the remaining text which is equal to a previous block plus one character.
- The partition is performed by building a trie that stores the blocks.

```
0 a b a a c b a a c a
```

```
0
```

```
a c a a b a a c a b
```
Encoding

- Partition the input string into blocks. Each block consists of a maximum prefix of the remaining text which is equal to a previous block plus one character.
- The partition is performed by building a trie that stores the blocks.

```
<table>
<thead>
<tr>
<th>1</th>
<th>a</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>a c a a b a a c a b</td>
</tr>
</tbody>
</table>
```
Partition the input string into blocks. Each block consists of a maximum prefix of the remaining text which is equal to a previous block plus one character.
The partition is performed by building a trie that stores the blocks.

```
Encoding

- **Partition the input string into blocks.** Each block consists of a maximum prefix of the remaining text which is equal to a previous block plus one character.
- **The partition is performed by building a trie that stores the blocks.**

![Trie diagram]

```

```
Partition the input string into blocks. Each block consists of a maximum prefix of the remaining text which is equal to a previous block plus one character.

The partition is performed by building a trie that stores the blocks.

Encoding

Encode each block by a pair \((i, c)\), where \(i\) is the number of the parent block, and \(c\) is the last character of the block.
Partition the input string into blocks. Each block consists of a maximum prefix of the remaining text which is equal to a previous block plus one character.

The partition is performed by building a trie that stores the blocks.
Encoding

- Partition the input string into blocks. Each block consists of a maximum prefix of the remaining text which is equal to a previous block plus one character.
- The partition is performed by building a trie that stores the blocks.
Partition the input string into blocks. Each block consists of a maximum prefix of the remaining text which is equal to a previous block plus one character.

The partition is performed by building a trie that stores the blocks.
Encoding

- Partition the input string into blocks. Each block consists of a maximum prefix of the remaining text which is equal to a previous block plus one character.
- The partition is performed by building a **trie** that stores the blocks.
- Encode each block by a pair \((i, c)\), where \(i\) is the number of the parent block, and \(c\) is the last character of the block.

\[
\begin{align*}
0 & \rightarrow a, b, c \\
1 & \rightarrow a, b \\
2 & \rightarrow c \\
3 & \rightarrow c \\
4 & \rightarrow a, b \\
5 & \rightarrow c \\
\end{align*}
\]

- (0,a) (0,c) (1,a) (0,b) (3,c) (1,b)

\[
a c a a b a a c a b
\]
Decoding

- To reconstruct the text, scan the encoded blocks, and construct the blocks tree.

(0, a) (0, c) (1, a) (0, b) (3, c) (1, b)
To reconstruct the text, scan the encoded blocks, and construct the blocks tree.
To reconstruct the text, scan the encoded blocks, and construct the blocks tree.
Decoding

To reconstruct the text, scan the encoded blocks, and construct the blocks tree.

```
(0,a) (0,c) (1,a) (0,b) (3,c) (1,b)
```

```
acaca

0
a

1
a

2
c

3

a
```
Decoding

To reconstruct the text, scan the encoded blocks, and construct the blocks tree.

```
(0,a) (0,c) (1,a) (0,b) (3,c) (1,b)
```

```
 a c a a b
```

```
a
```

```
1
```

```
2
```

```
4
```

```
3
```

```
0
```

```
a
```

```
c
```

```
b
```
To reconstruct the text, scan the encoded blocks, and construct the blocks tree.
To reconstruct the text, scan the encoded blocks, and construct the blocks tree.

```
(0,a) (0,c) (1,a) (0,b) (3,c) (1,b)
```

```
acacababc
```