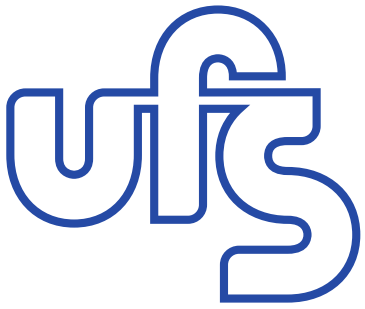
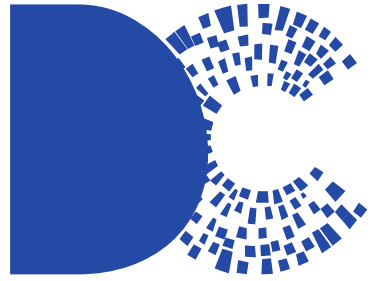


UNIVERSIDADE FEDERAL DE SERGIPE



DEPARTAMENTO DE COMPUTAÇÃO

# Árvore de prefixo

## Estruturas de Dados

Bruno Prado

Departamento de Computação / UFS

# Introdução

- O que é uma árvore de prefixo (*trie*)?
  - É uma árvore $k$-ária que possui vetores de tamanho $k$ para indexação dos nós filhos do alfabeto

# Introdução

- O que é uma árvore de prefixo (*trie*)?
  - É uma árvore $k$-ária que possui vetores de tamanho $k$ para indexação dos nós filhos do alfabeto
  - Sua construção é feita utilizando os caracteres dos termos utilizados para busca em cadeias de texto

# Introdução

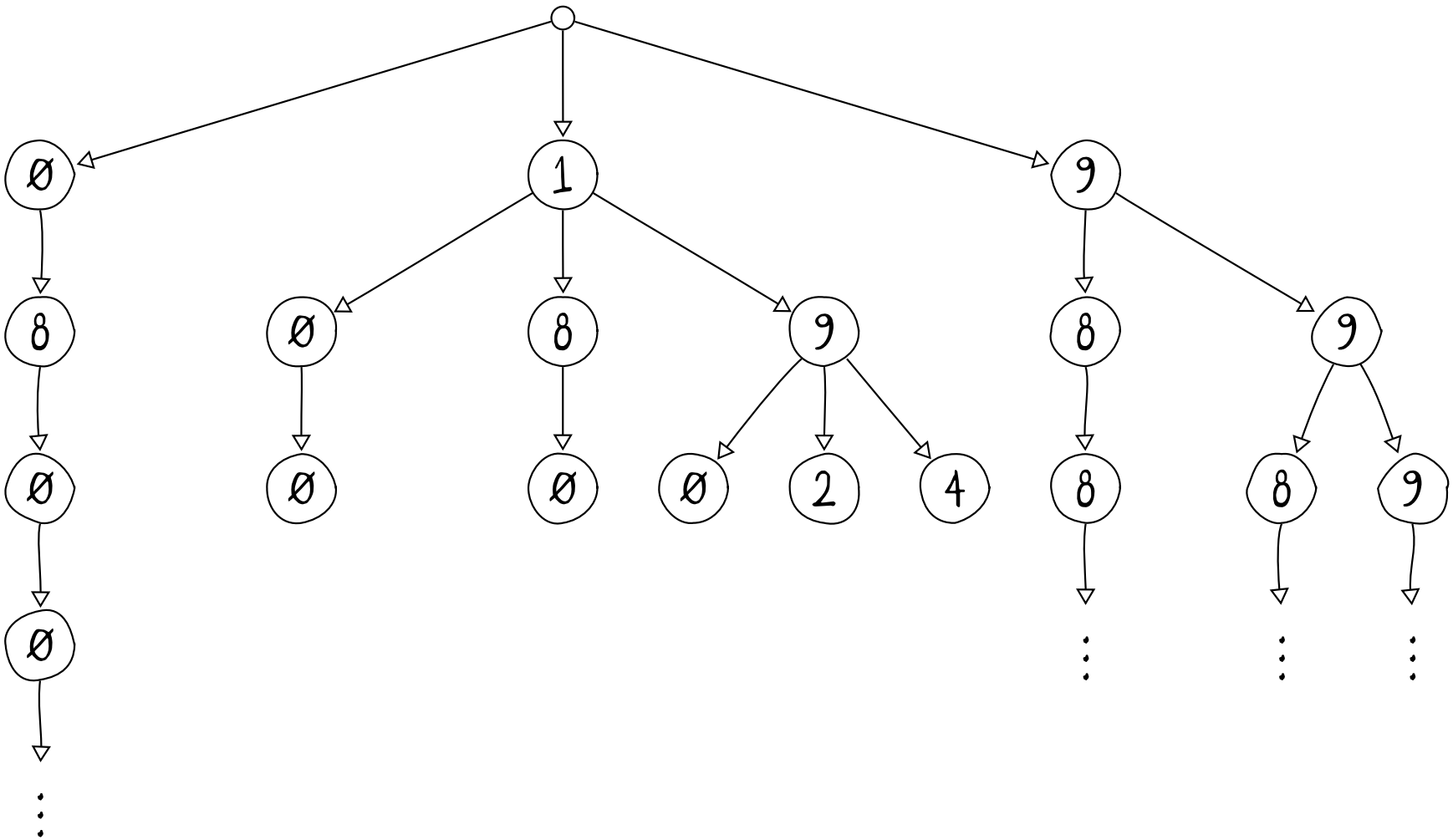
- O que é uma árvore de prefixo (*trie*)?
  - É uma árvore $k$-ária que possui vetores de tamanho $k$ para indexação dos nós filhos do alfabeto
  - Sua construção é feita utilizando os caracteres dos termos utilizados para busca em cadeias de texto
  - Em cada nível da árvore é possível visualizar todos os termos que possuem o mesmo prefixo

# Introdução

- O que é uma árvore de prefixo (*trie*)?
  - É uma árvore *k*-ária que possui vetores de tamanho *k* para indexação dos nós filhos do alfabeto
  - Sua construção é feita utilizando os caracteres dos termos utilizados para busca em cadeias de texto
  - Em cada nível da árvore é possível visualizar todos os termos que possuem o mesmo prefixo
  - Em inglês *trie* vem da palavra *retrieval*, entretanto, para evitar confusão com o termo *tree*, é pronunciado como *try*

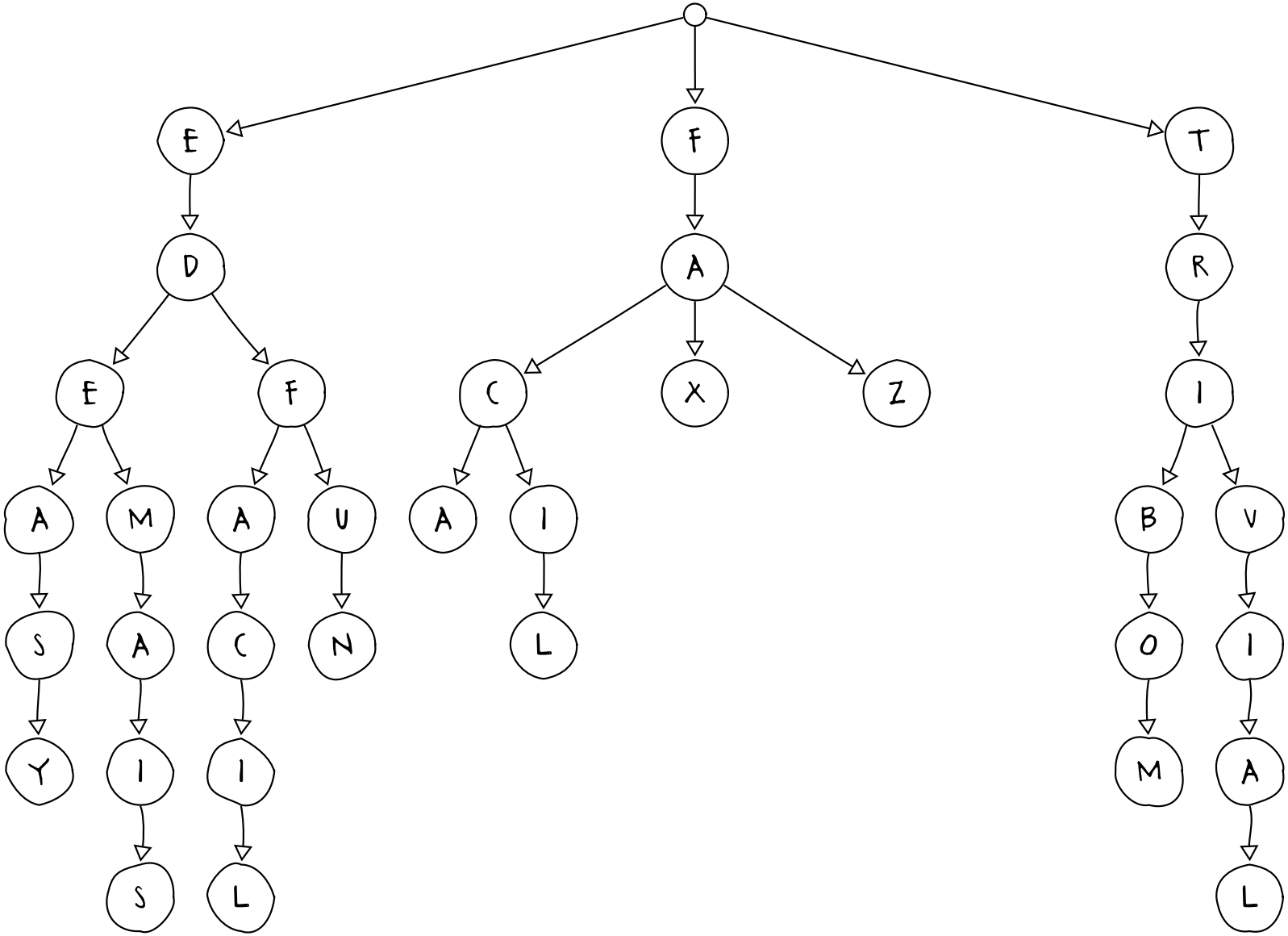
# Introdução

- Árvore de prefixo para números decimais ($k = 10$)



# Introdução

- Árvore de prefixo para letras minúsculas ($k = 26$)

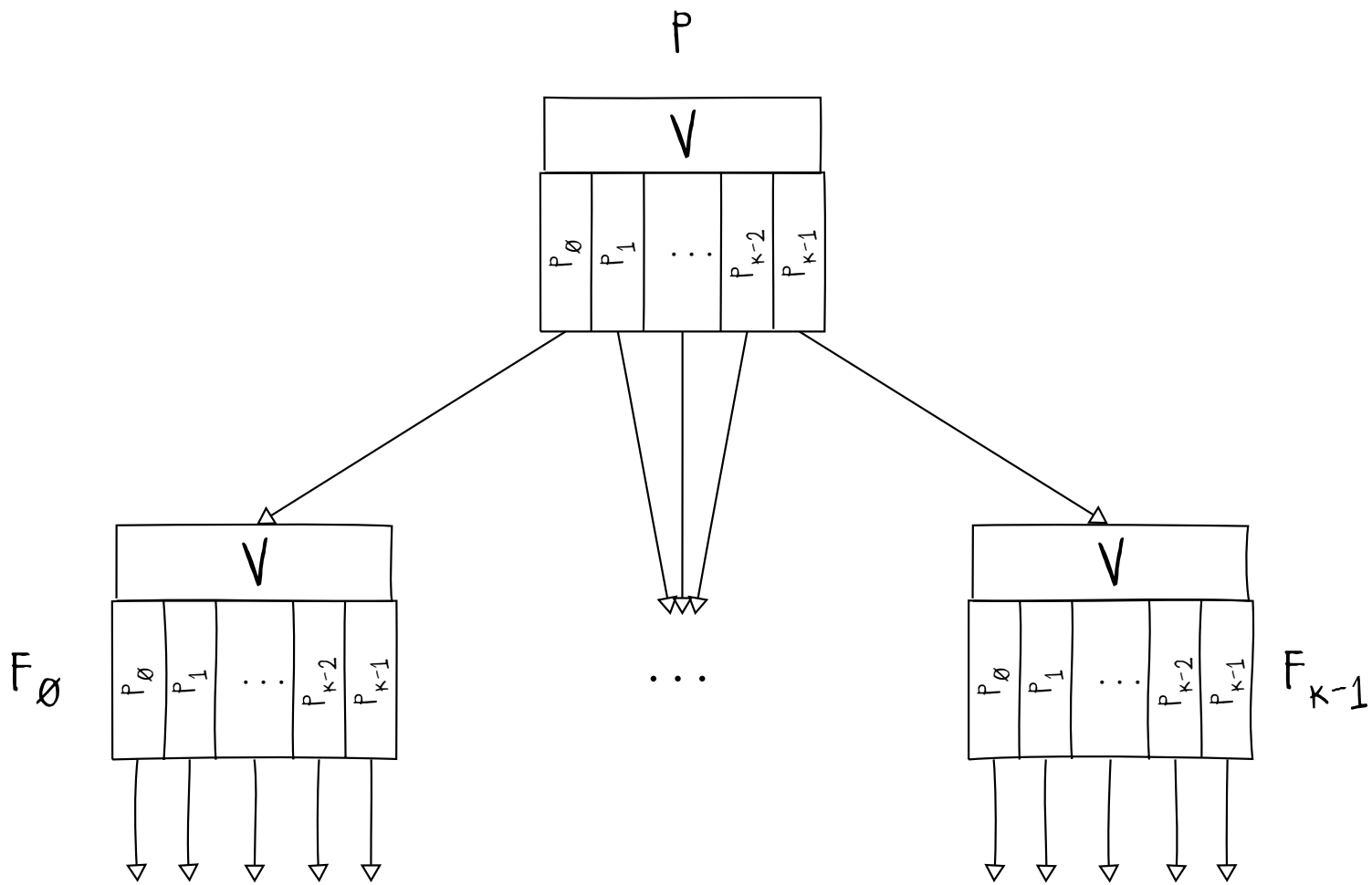


# Árvore de prefixo

- Operações básicas
  - Busca
  - Inserção
  - Remoção

# Árvore de prefixo

- Definição da estrutura



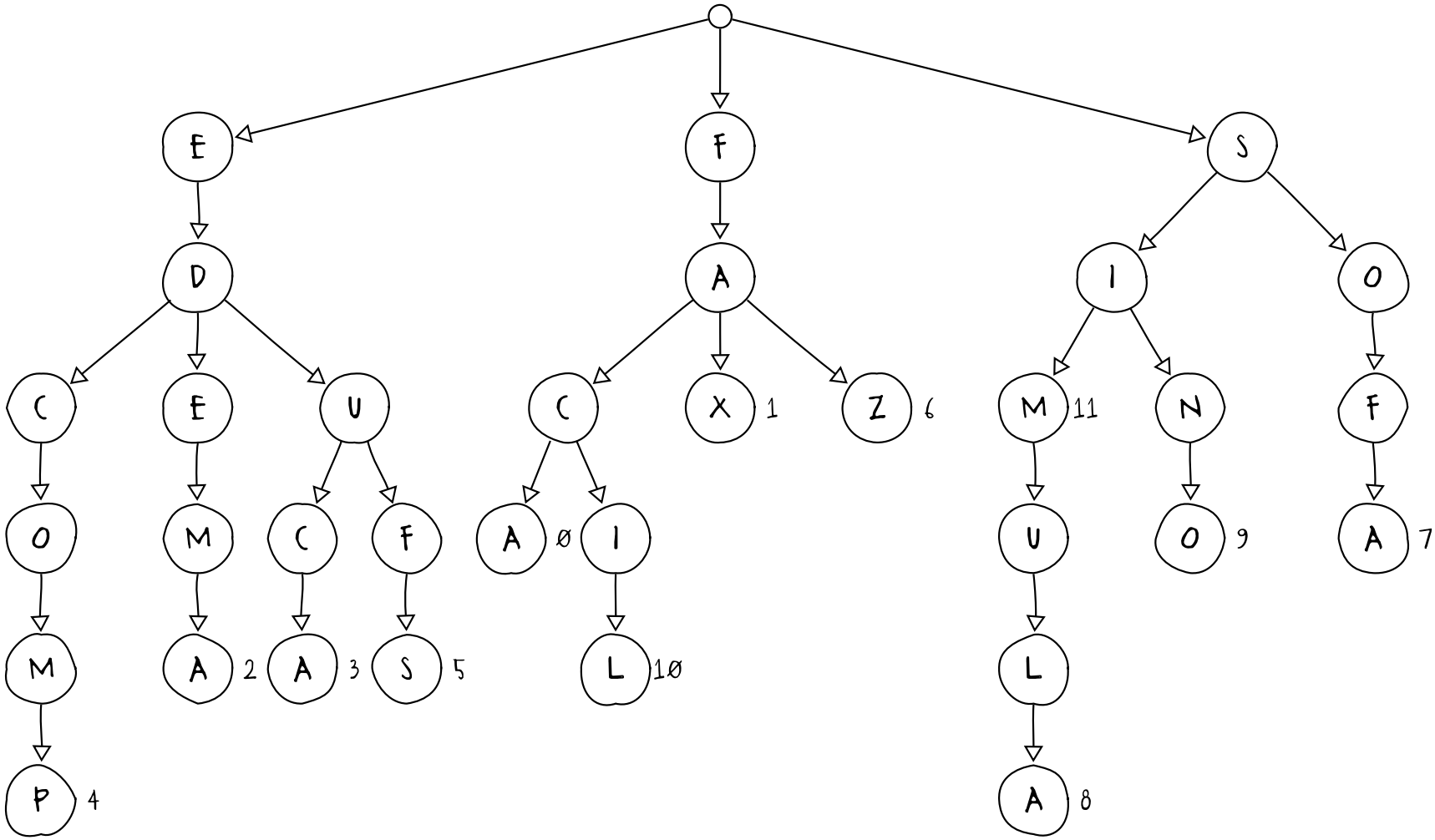
# Árvore de prefixo

- Implementação em C
  - Estrutura e ponteiros

```c
// Padrão de tipos por tamanho
#include <stdint.h>
// Estrutura de nó
typedef struct no {
    // Vetor de filhos
    struct no** P;
    // Valor associado
    uint32_t* V;
} no;
```

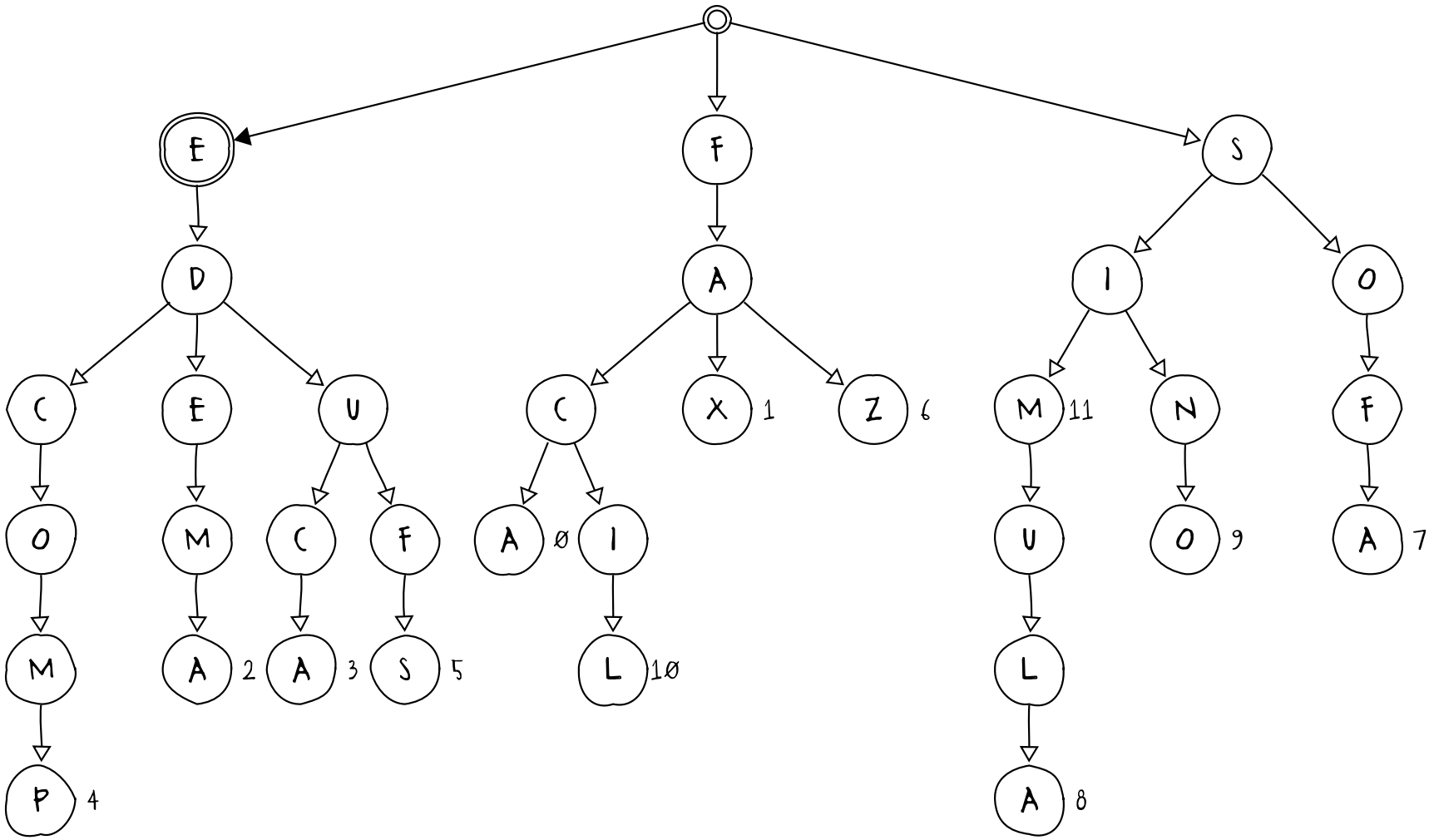
# Árvore de prefixo

- Operação de busca
  - Parâmetro: ed



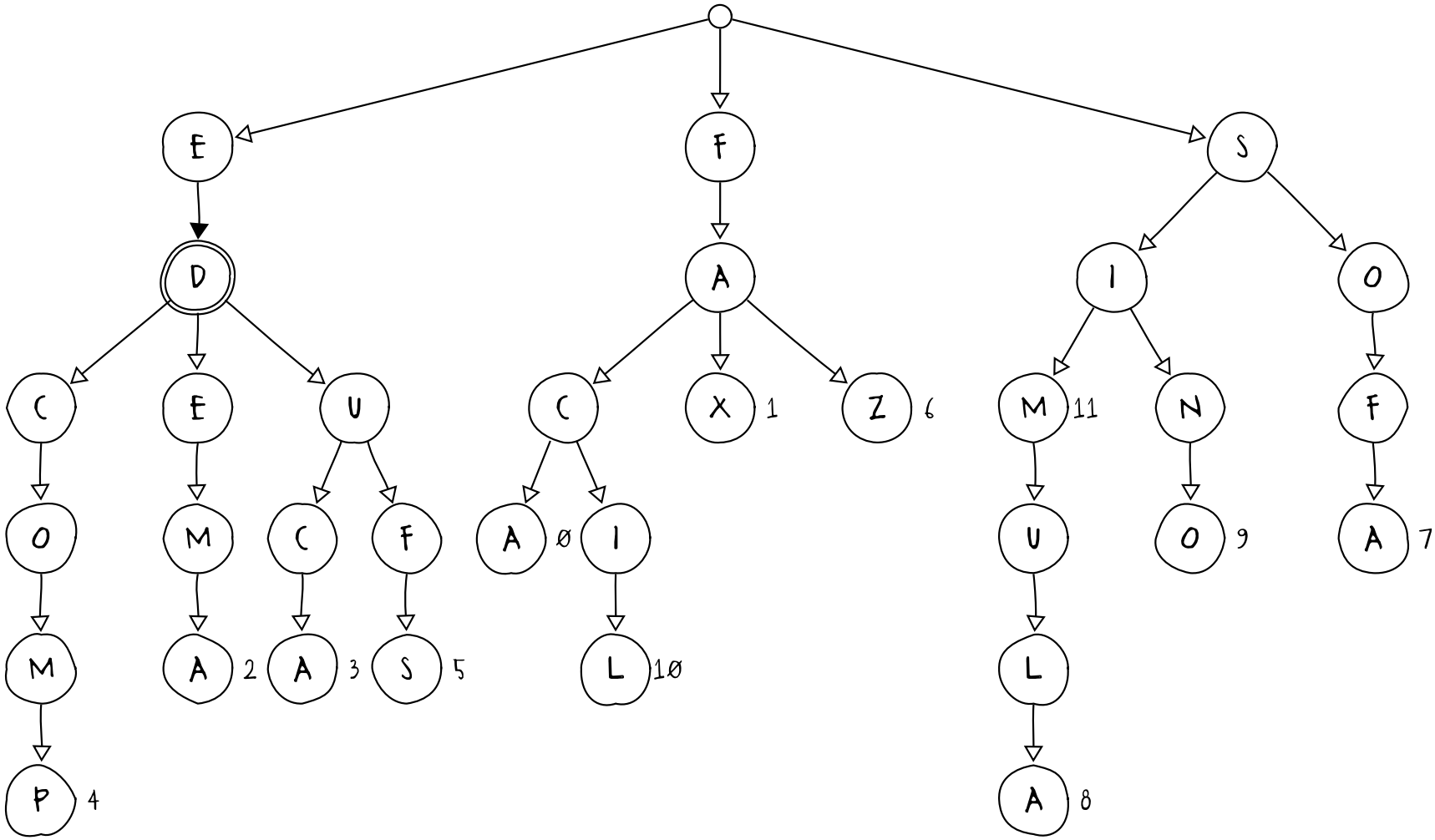
# Árvore de prefixo

- Operação de busca
  - Parâmetro: ed



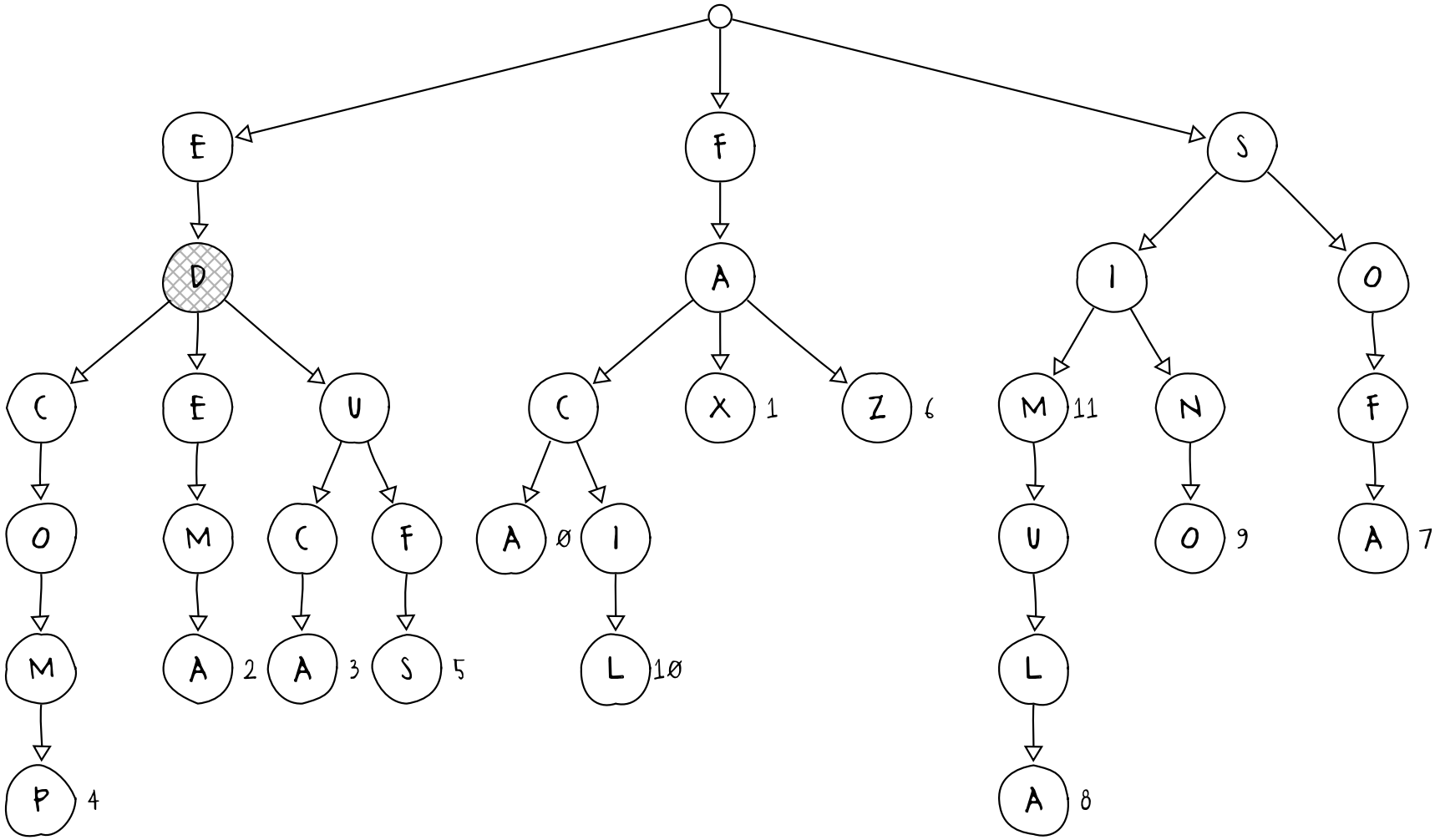
# Árvore de prefixo

- Operação de busca
  - Parâmetro: ed



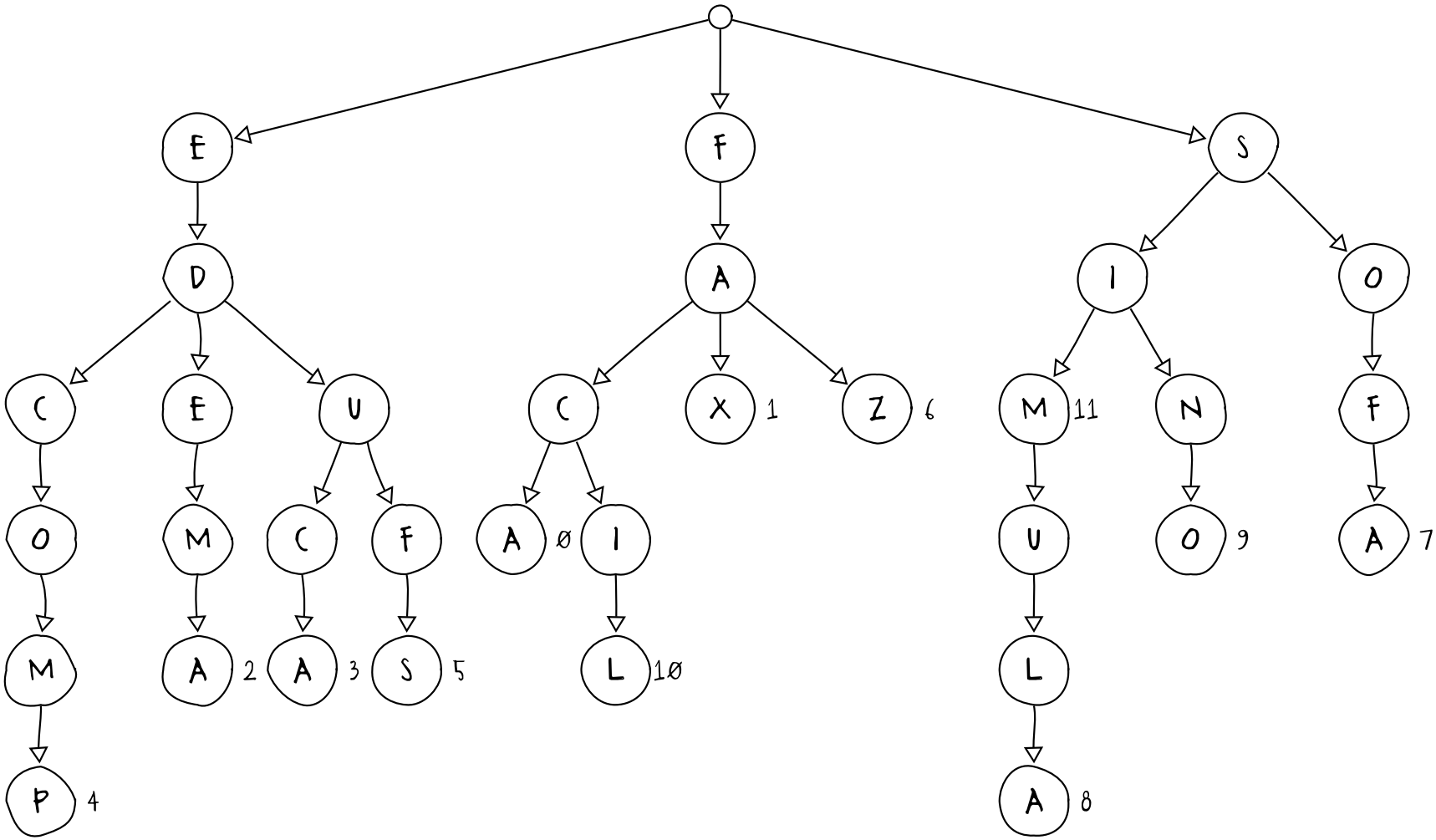
# Árvore de prefixo

- Operação de busca
  - Parâmetro: ed (**NULL**)



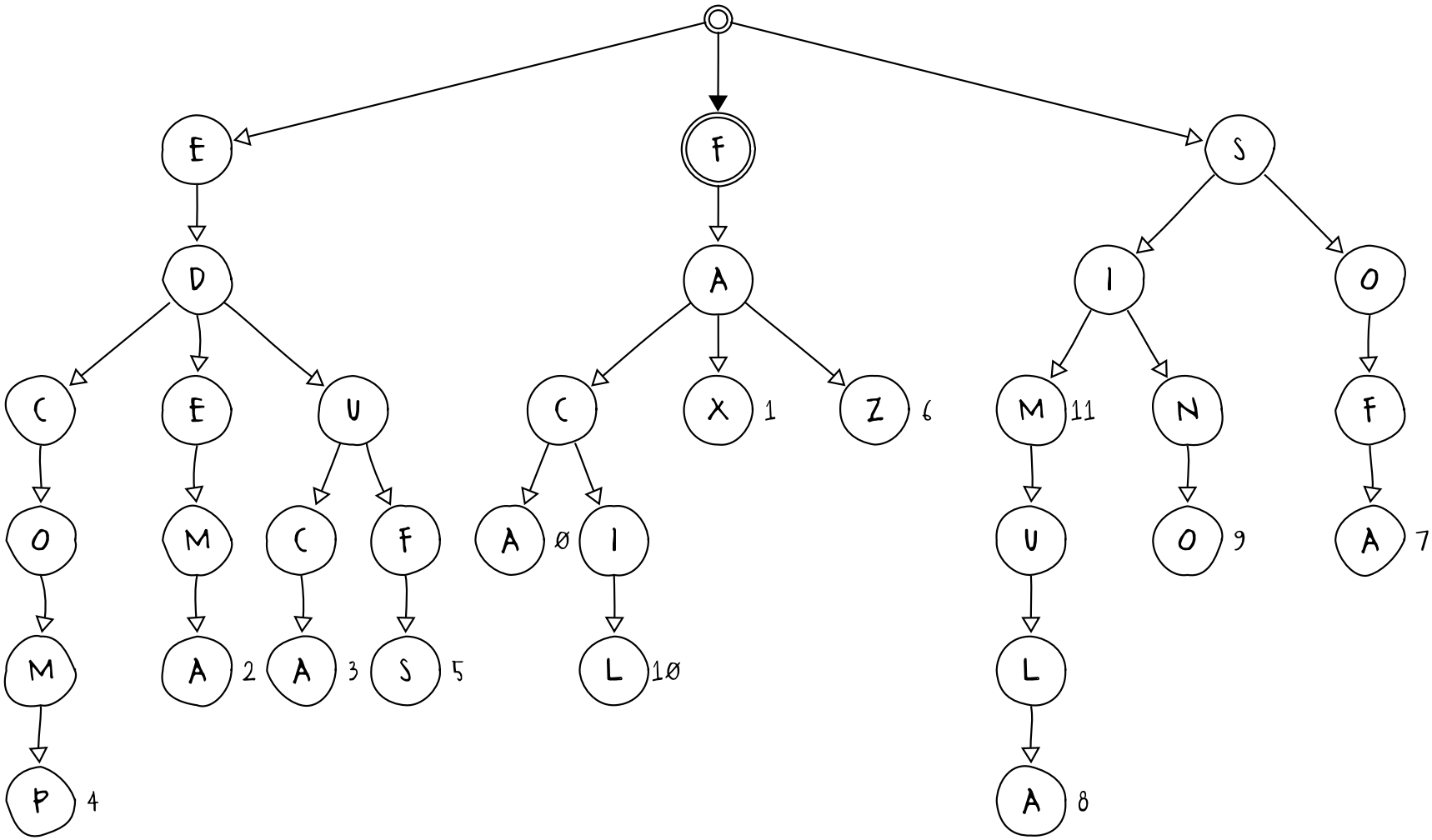
# Árvore de prefixo

- Operação de busca
  - Parâmetro: facil



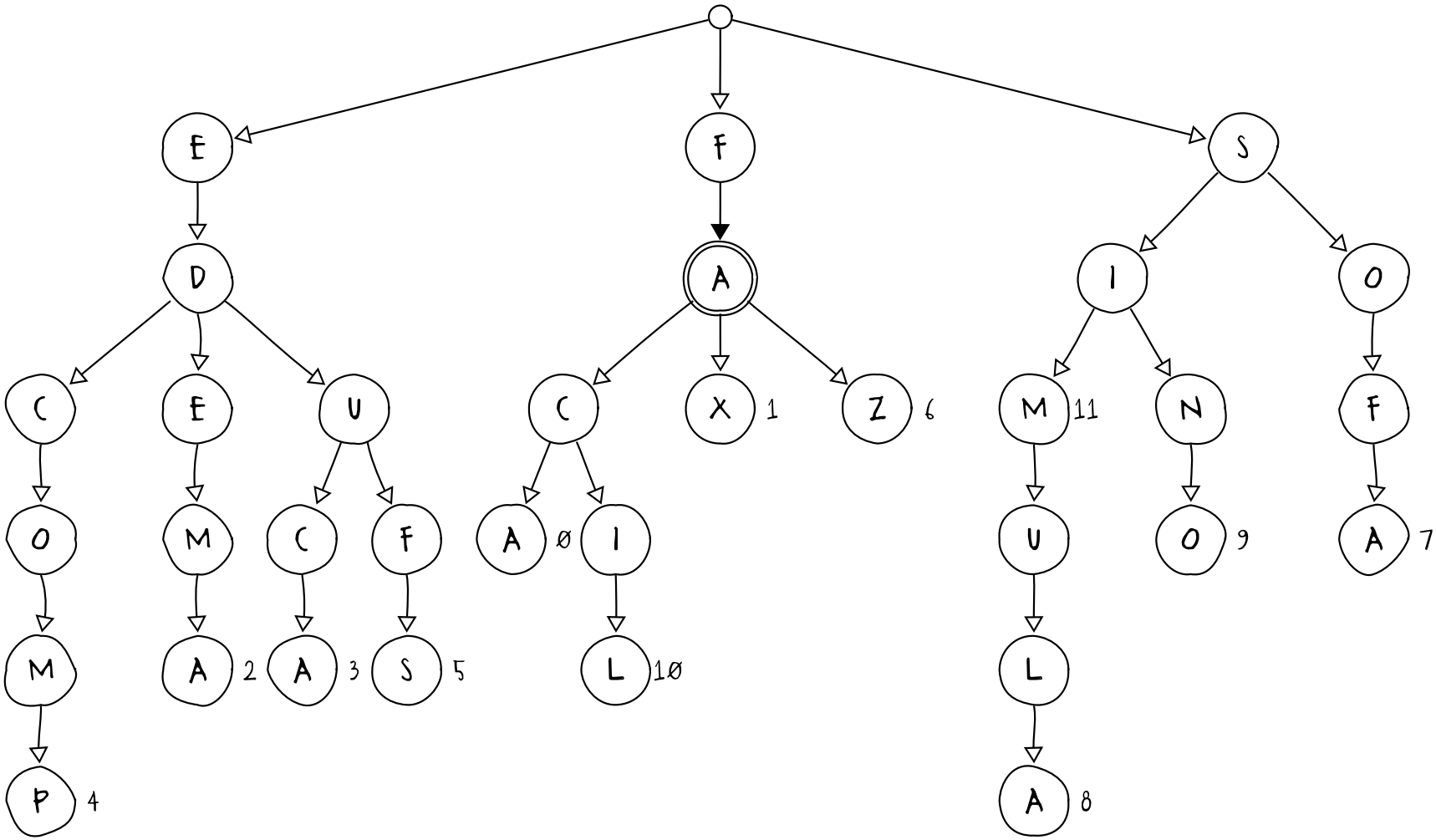
# Árvore de prefixo

- Operação de busca
  - Parâmetro: facil



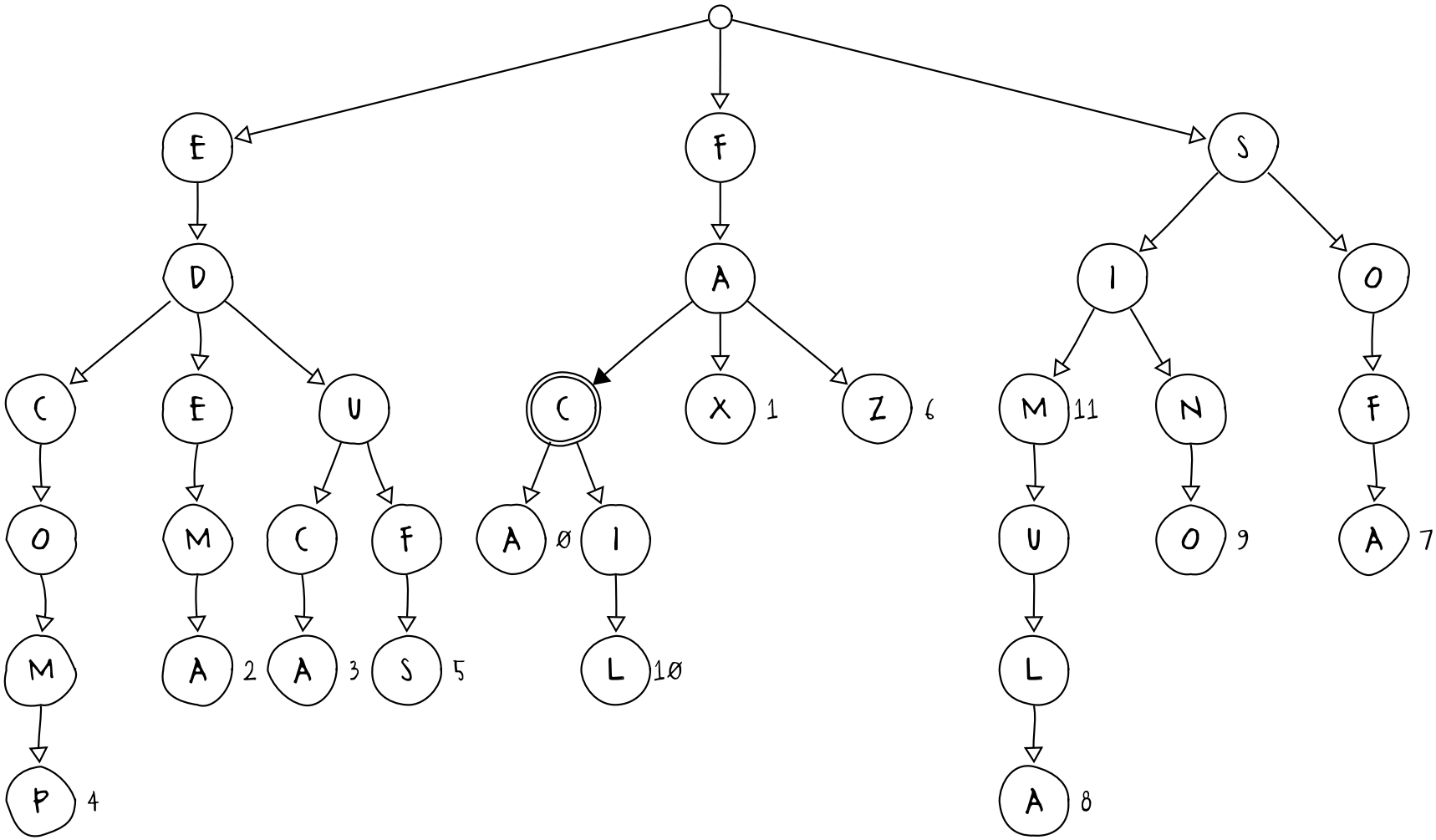
# Árvore de prefixo

- Operação de busca
  - Parâmetro: facil



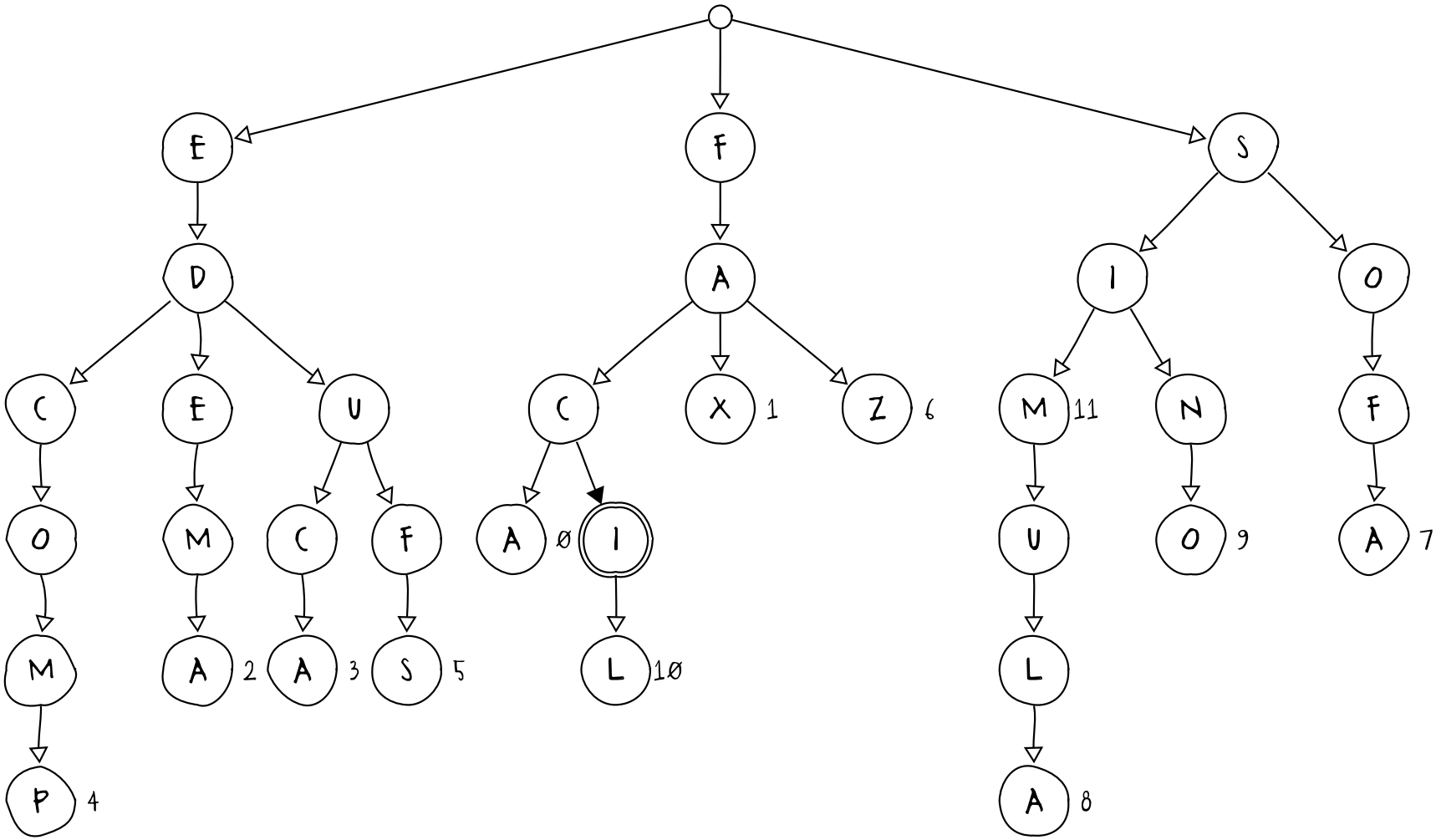
# Árvore de prefixo

- Operação de busca
  - Parâmetro: facil



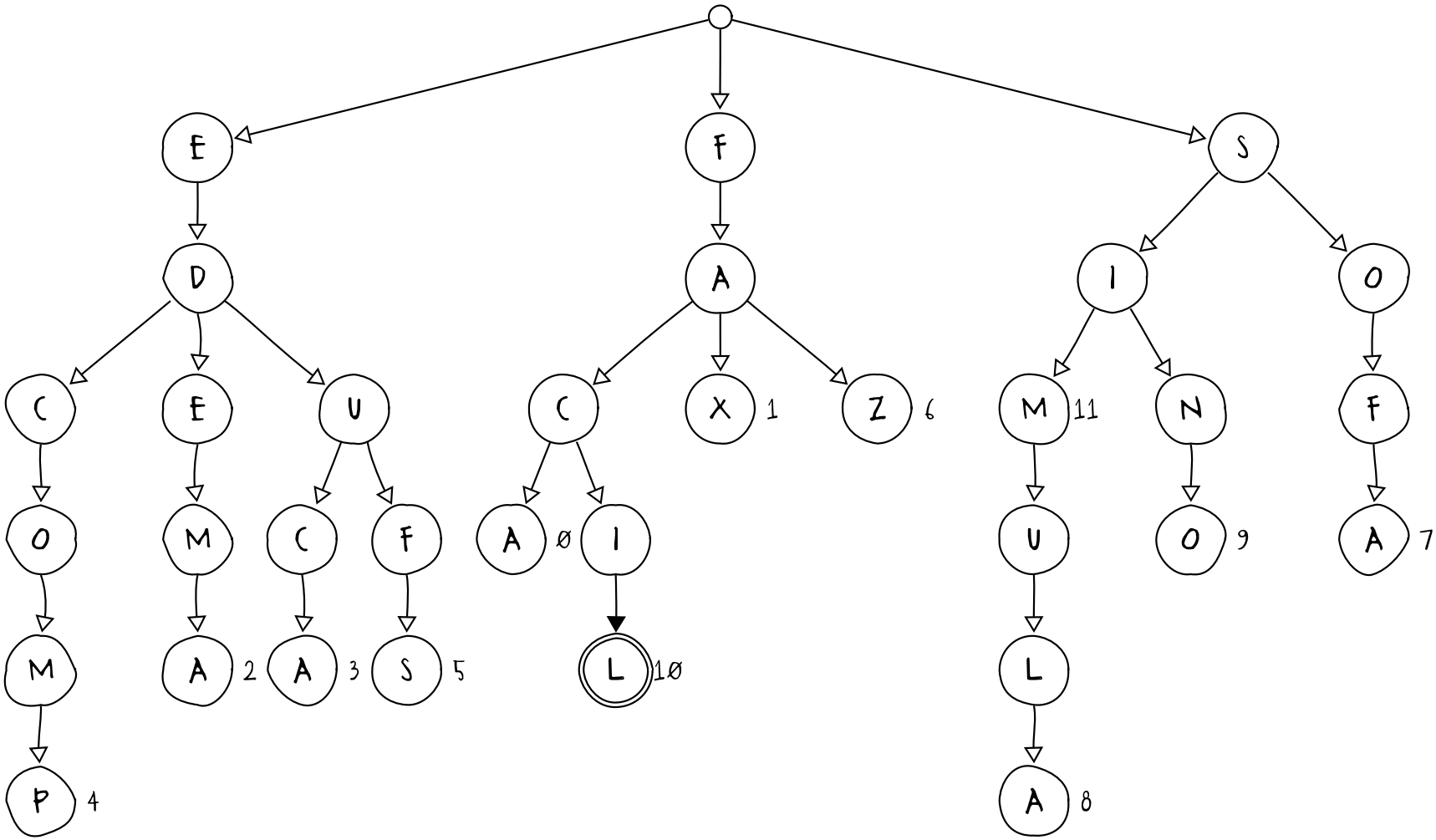
# Árvore de prefixo

- Operação de busca
  - Parâmetro: facil



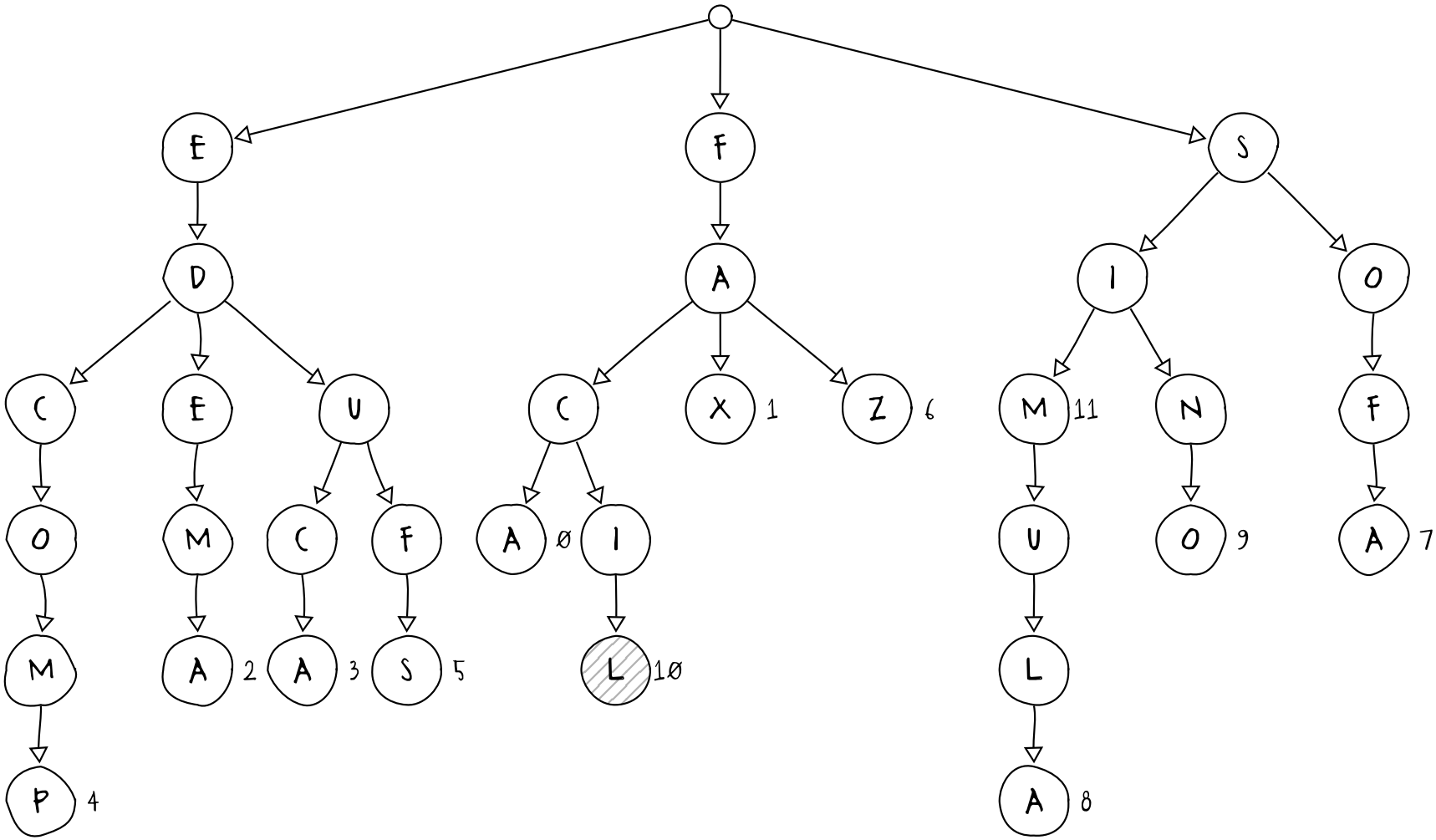
# Árvore de prefixo

- Operação de busca
  - Parâmetro: faci



# Árvore de prefixo

- Operação de busca
  - Parâmetro: facil (**10**)



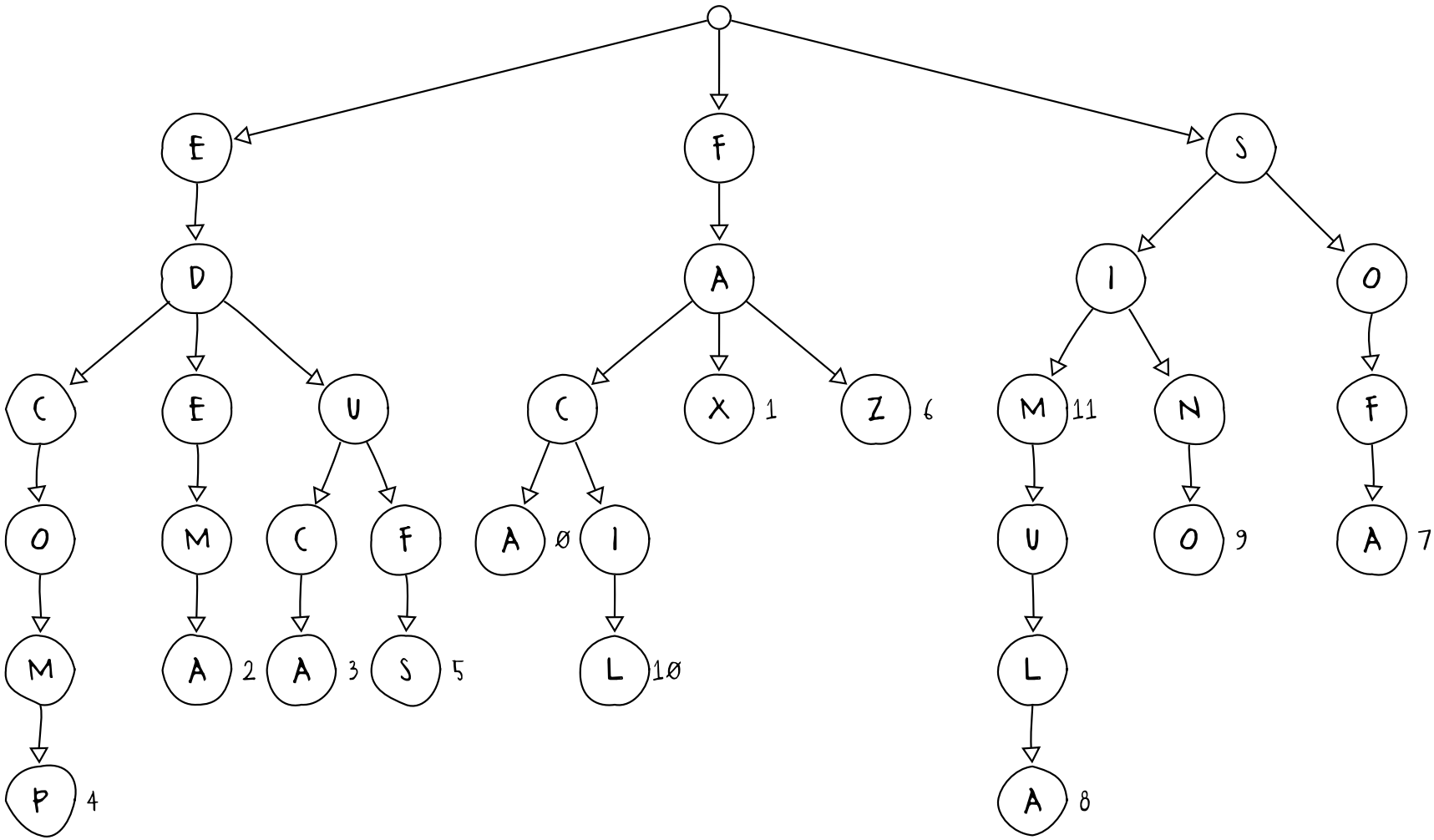
# Árvore de prefixo

- Implementação em C
  - Busca por cadeia de caracteres

```
1  // Função de busca de cadeia
2  no* busca(no* x, char* p, uint32_t d) {
3      // Inicializando resultado como nulo
4      no* r = NULL;
5      // Checando se x é nulo
6      if(x != NULL) {
7          // Verificando se todos os caracteres da cadeia
               já foram utilizados na busca
8          if(d == strlen(p))
9              r = x;
10         // Chamada recursiva no próximo nível
11         else
12             r = busca(x->P[indice(p, d)], p, d + 1);
13     }
14     // Retornando resultado
15     return r;
16 }
```

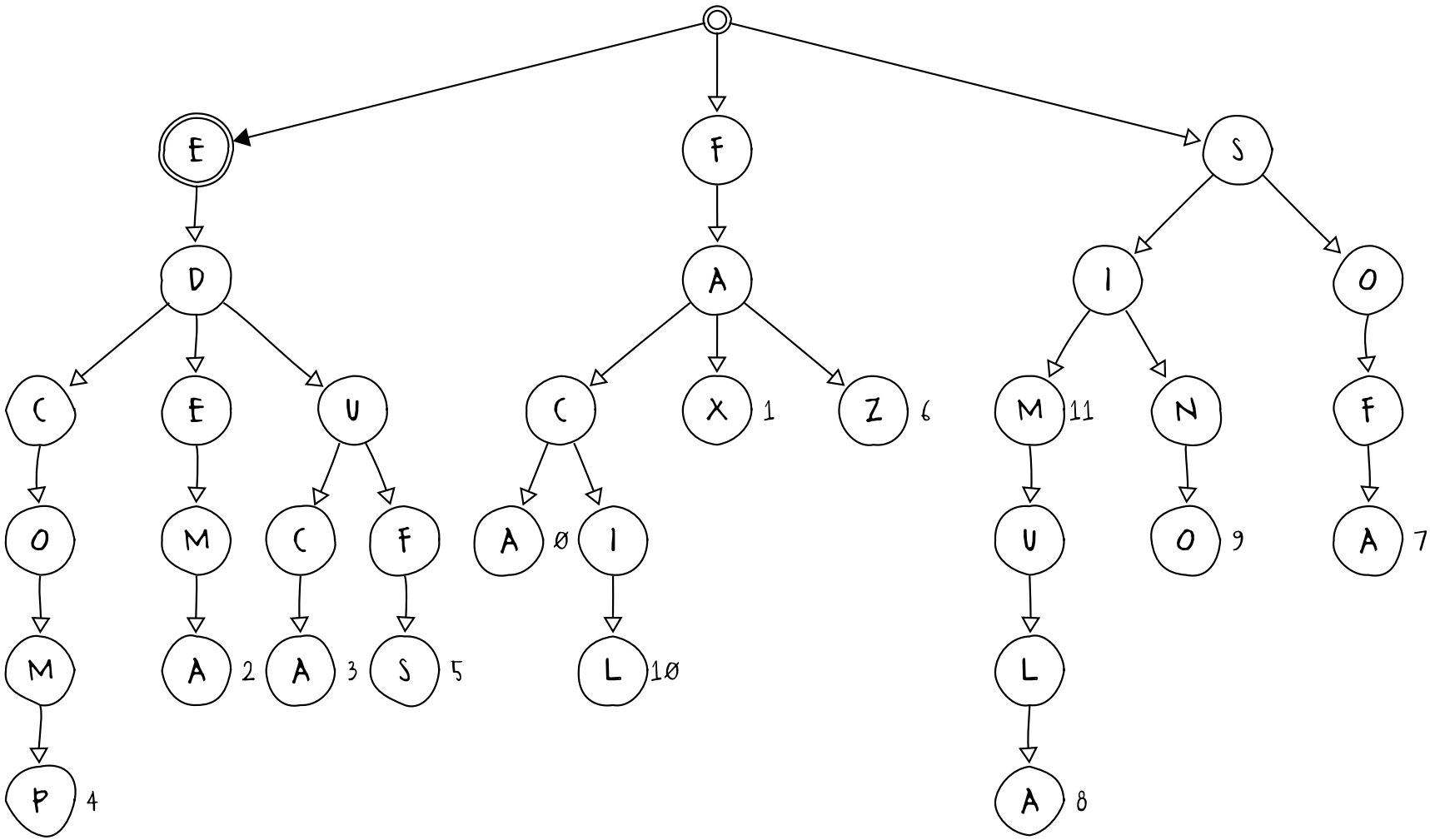
# Árvore de prefixo

- Operação de inserção
  - Parâmetro: ed



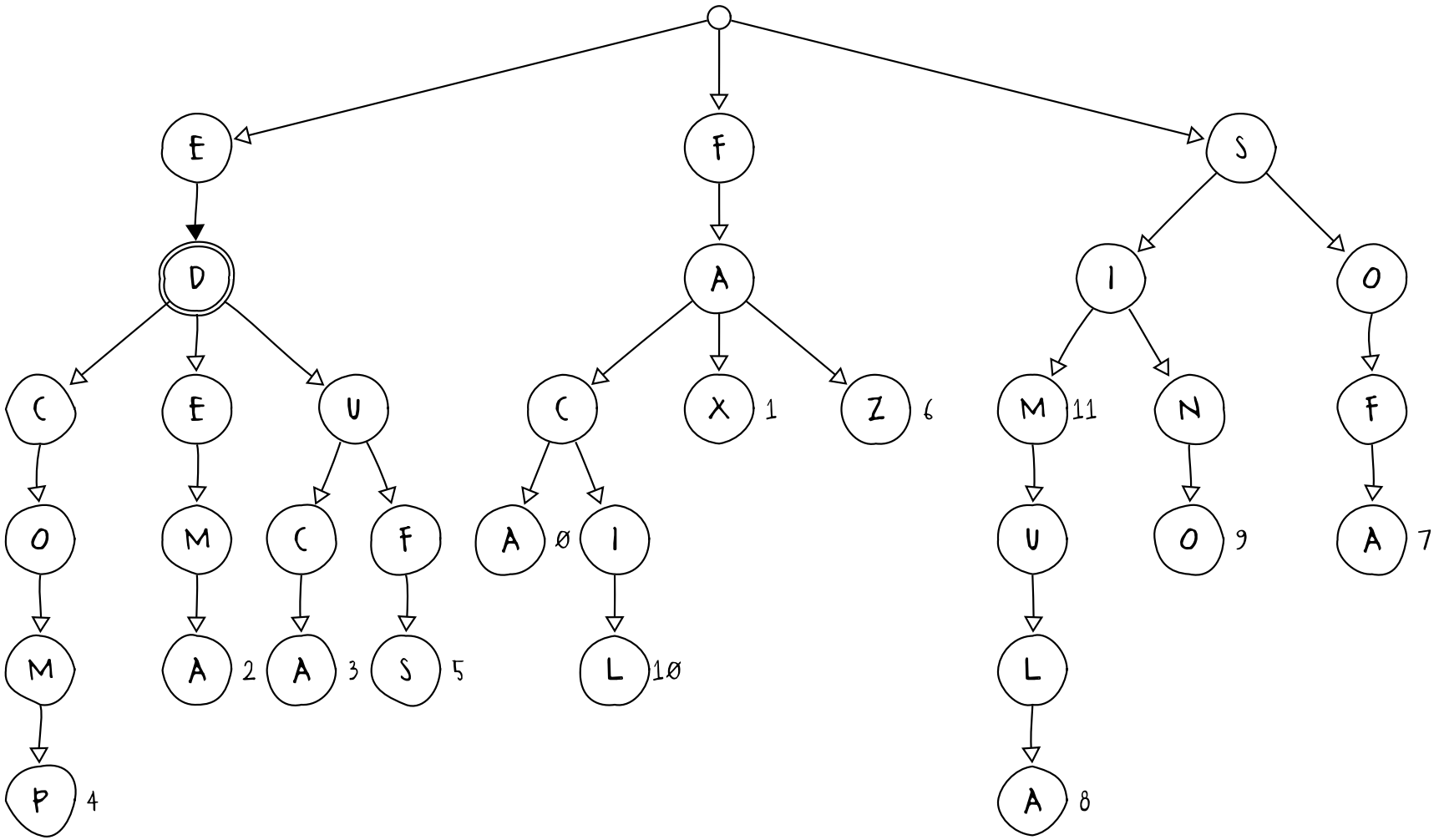
# Árvore de prefixo

- Operação de inserção
  - Parâmetro: ed



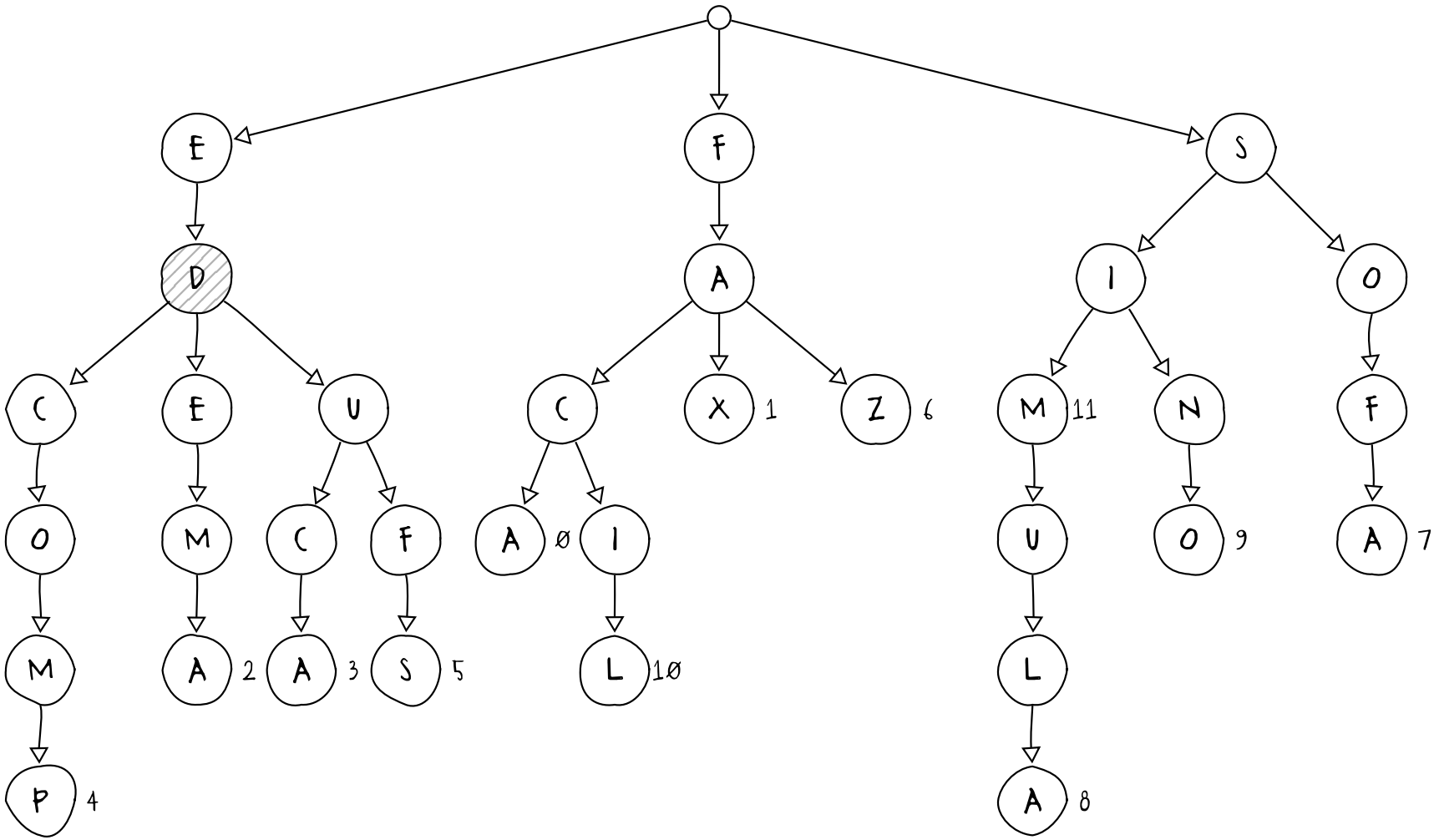
# Árvore de prefixo

- Operação de inserção
  - Parâmetro: ed



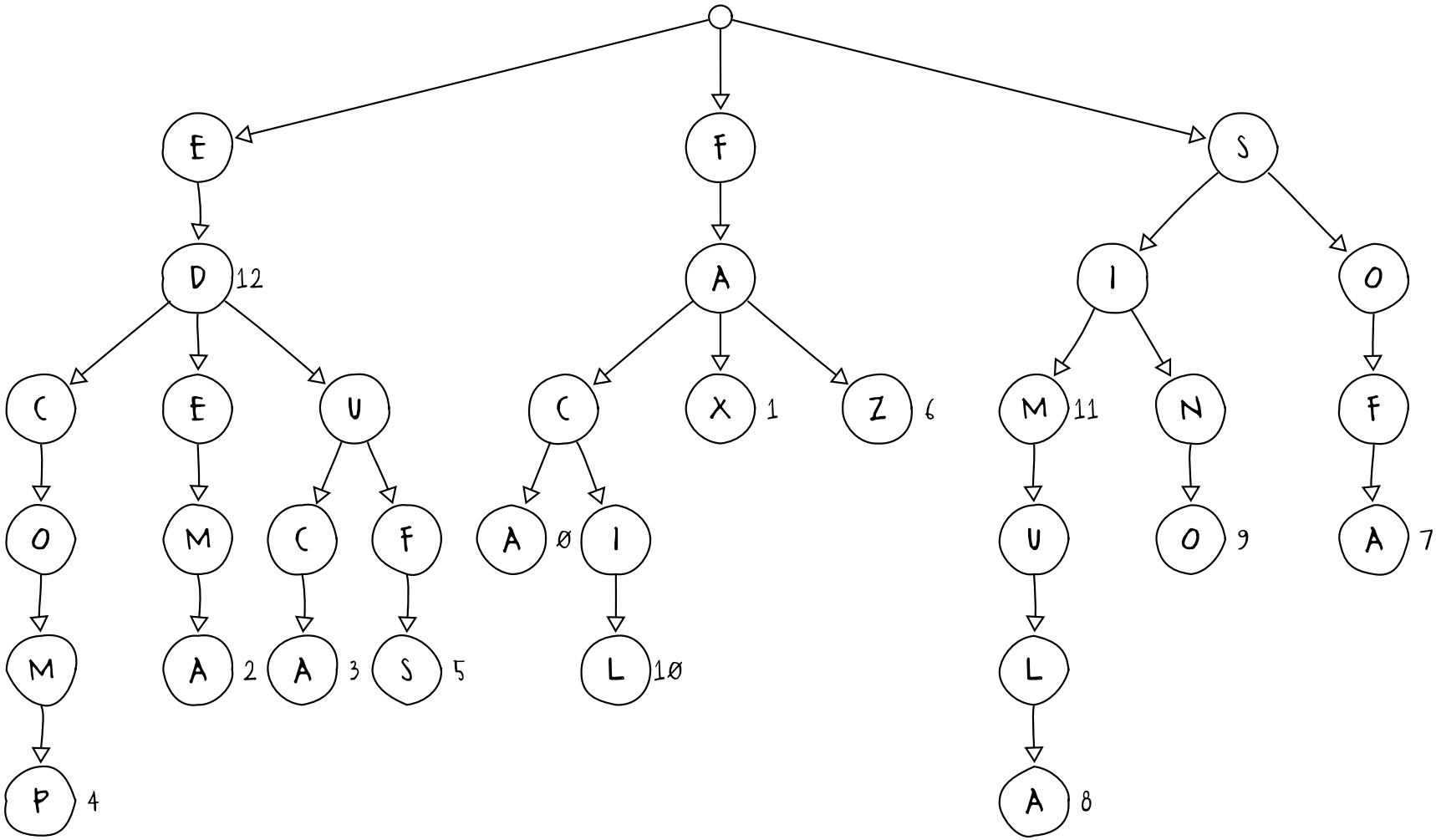
# Árvore de prefixo

- Operação de inserção
  - Parâmetro: ed



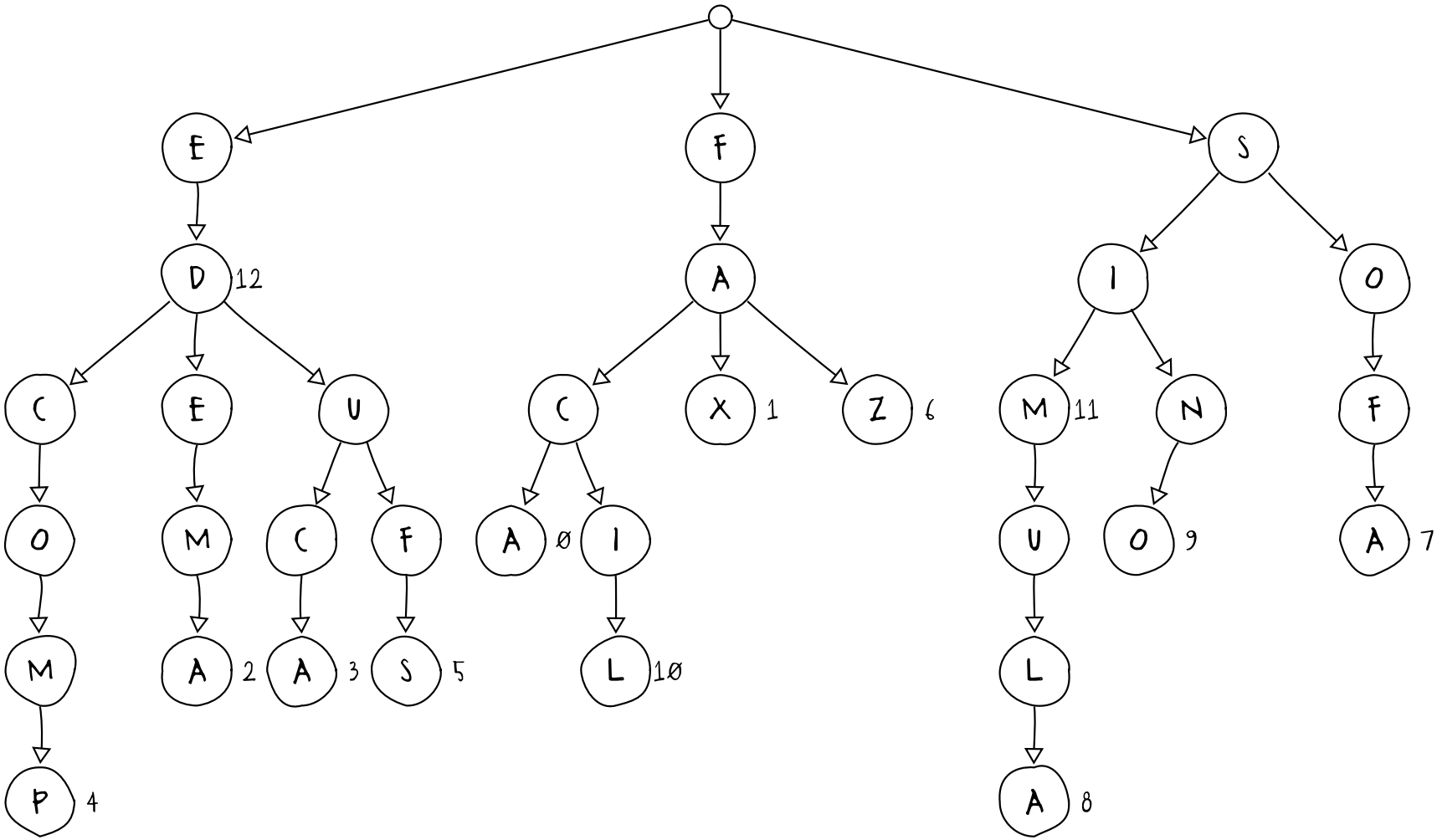
# Árvore de prefixo

- Operação de inserção
  - Parâmetro: ed (**12**)



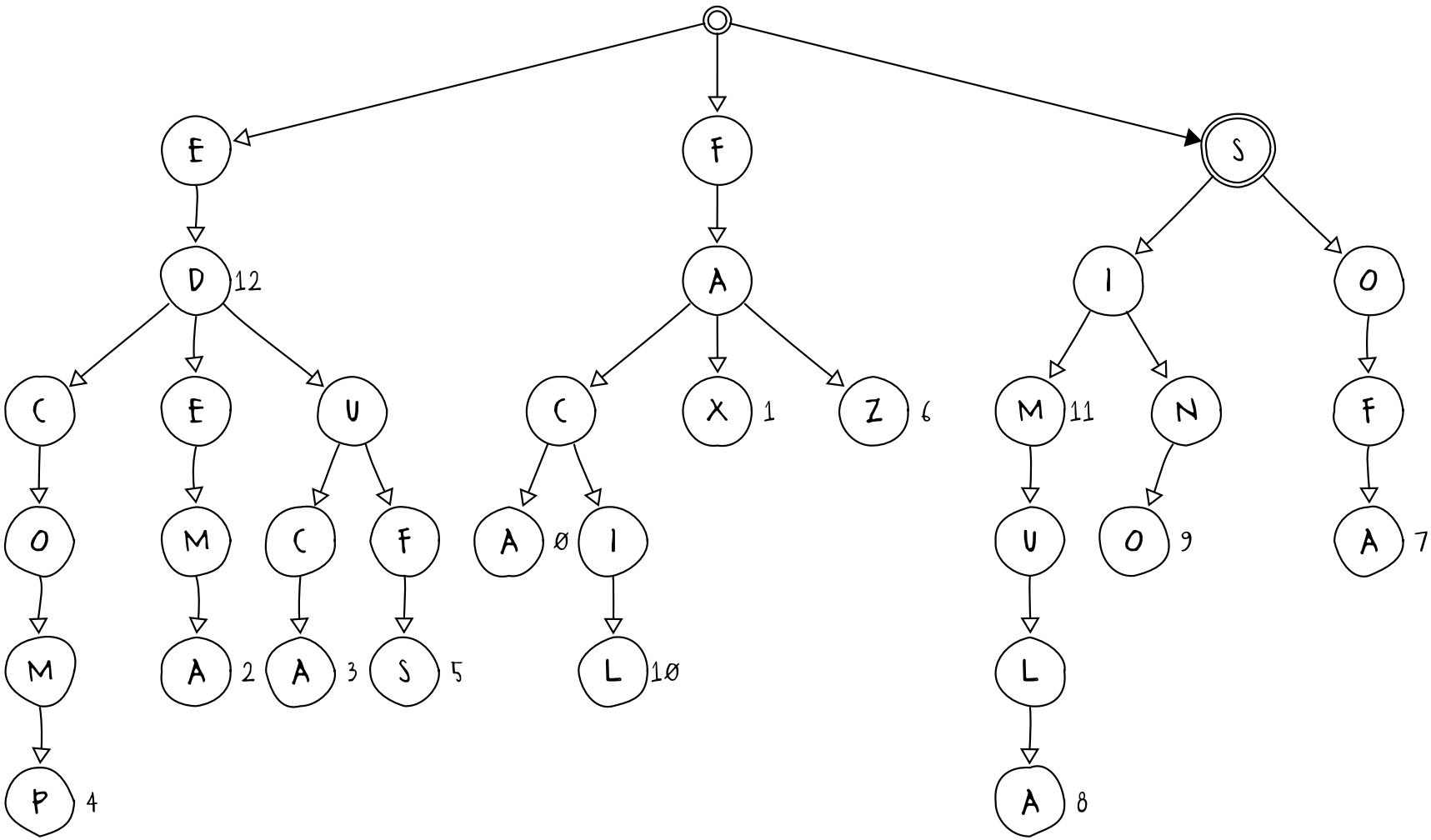
# Árvore de prefixo

- Operação de inserção
  - Parâmetro: sinuca



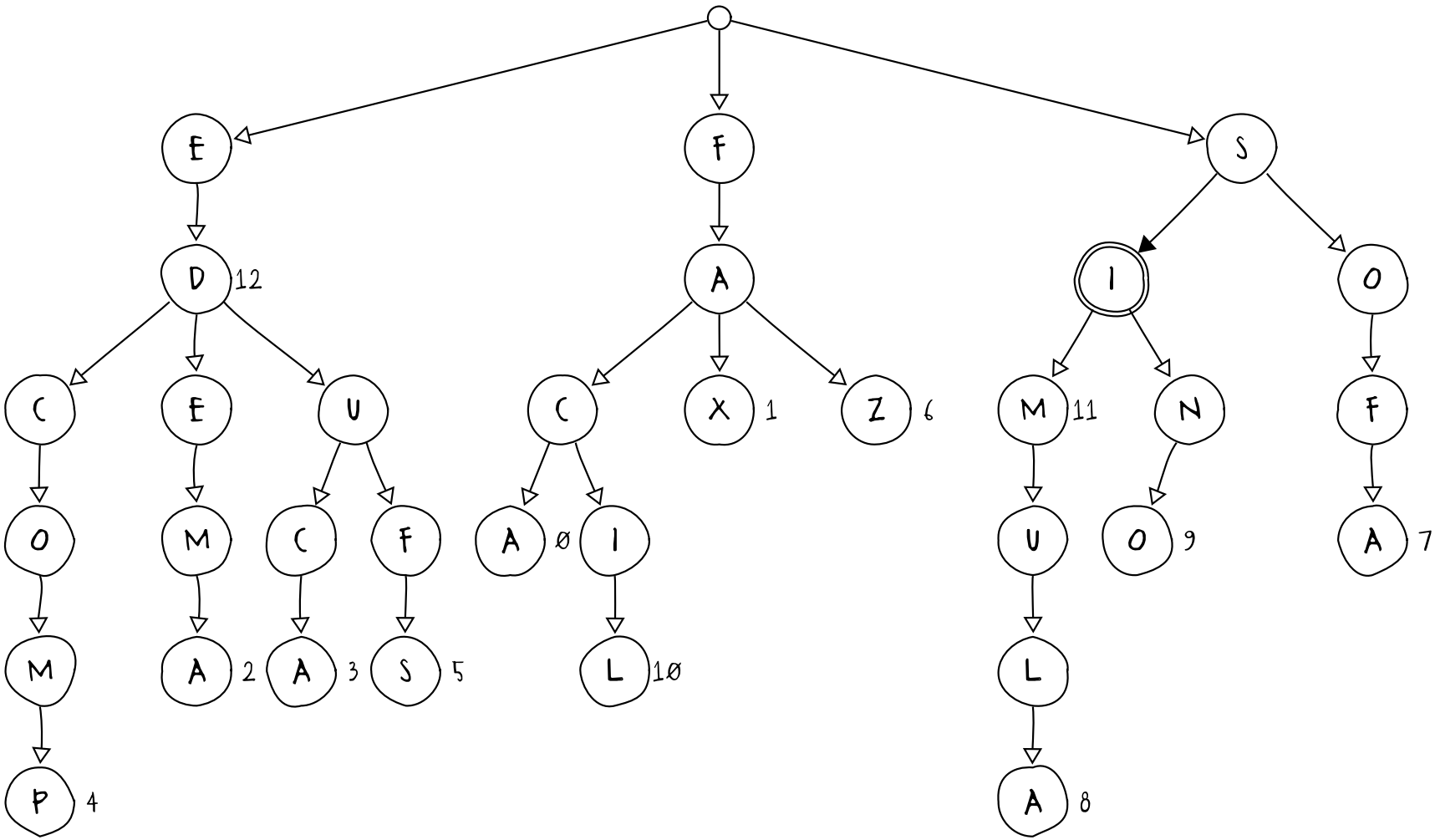
# Árvore de prefixo

- Operação de inserção
  - Parâmetro: sinuca



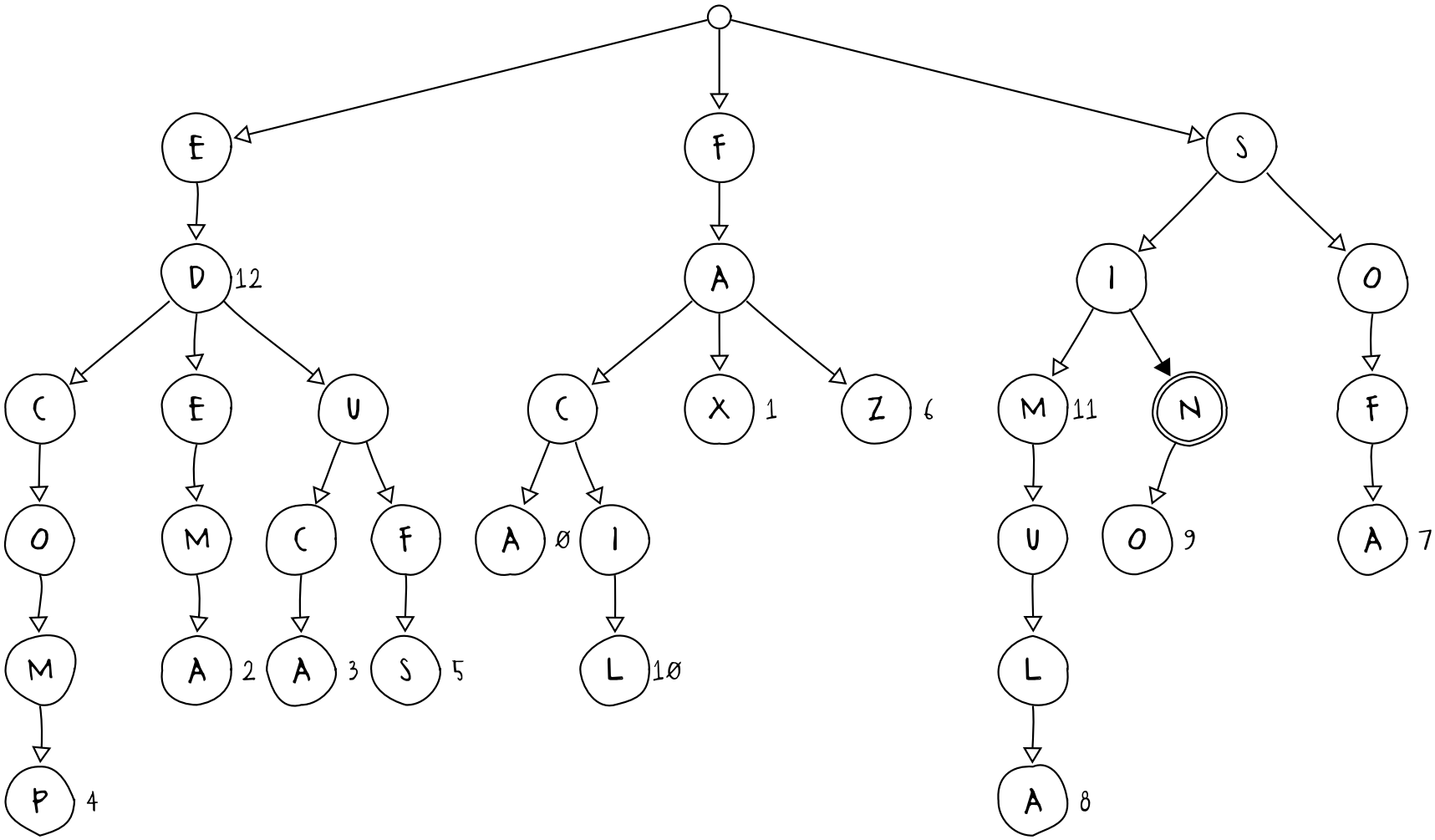
# Árvore de prefixo

- Operação de inserção
  - Parâmetro: sinuca



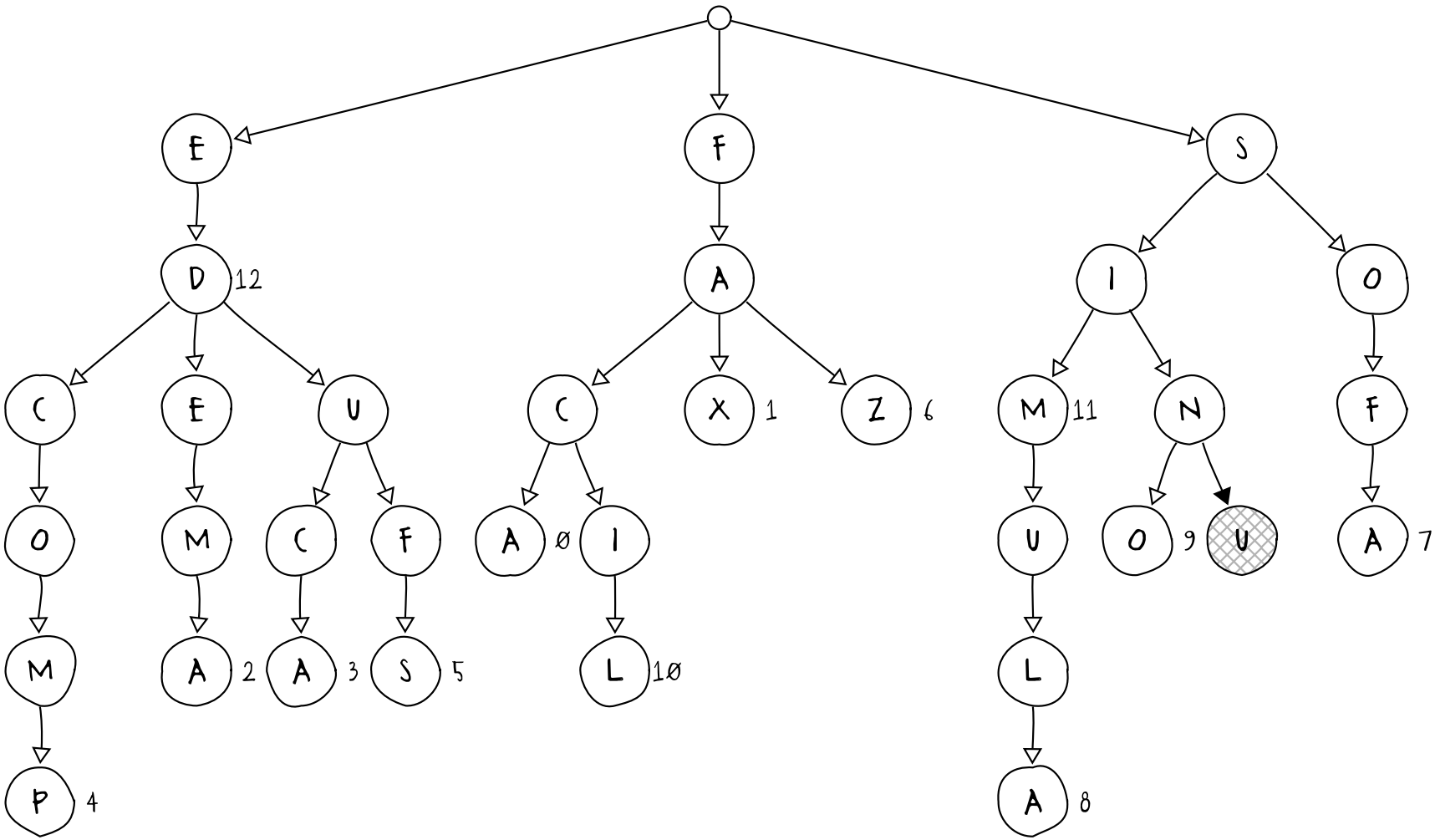
# Árvore de prefixo

- Operação de inserção
  - Parâmetro: sinuca



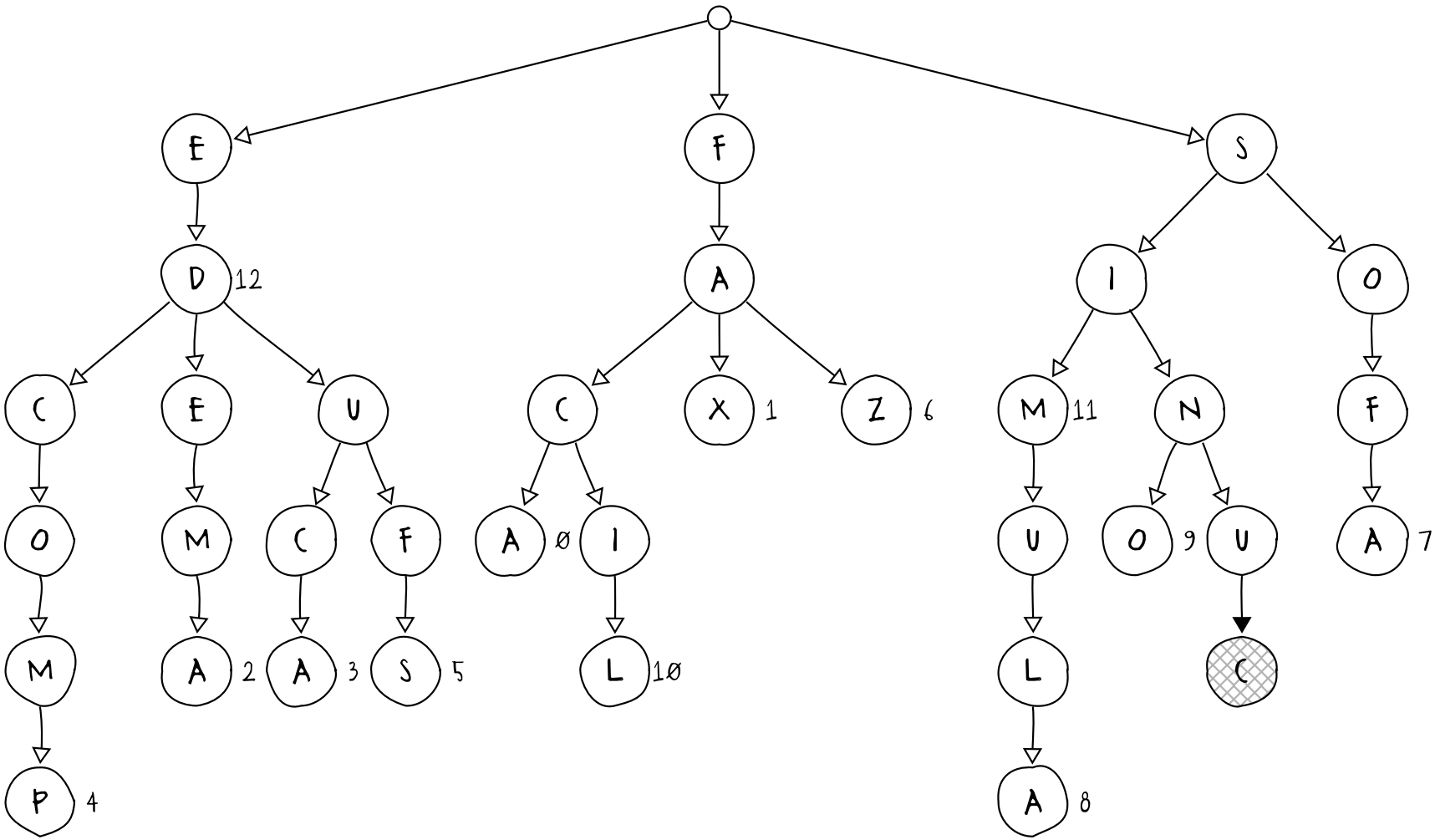
# Árvore de prefixo

- Operação de inserção
  - Parâmetro: sinuca



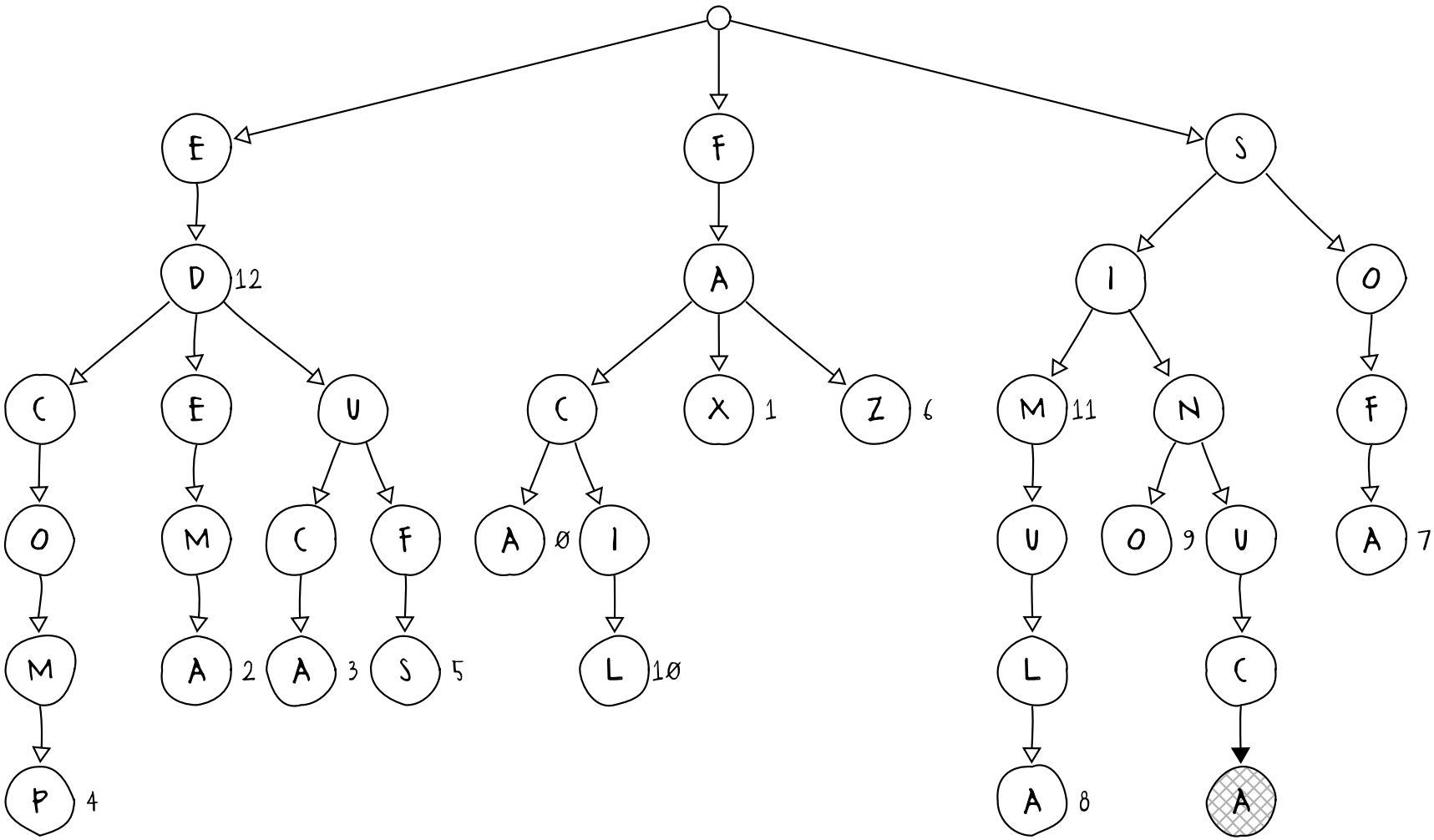
# Árvore de prefixo

- Operação de inserção
  - Parâmetro: sinuca



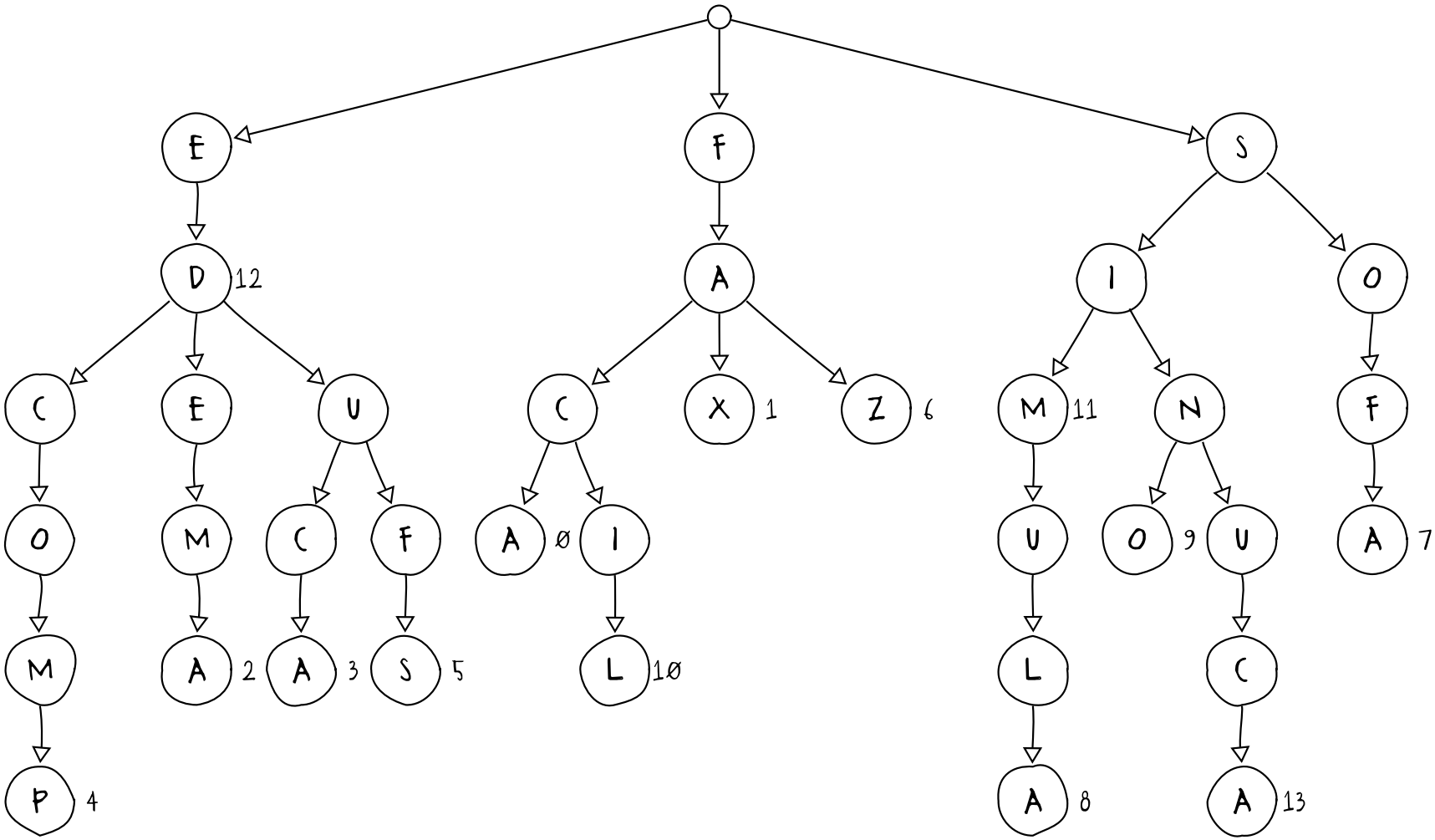
# Árvore de prefixo

- Operação de inserção
  - Parâmetro: sinuca



# Árvore de prefixo

- Operação de inserção
  - Parâmetro: sinuca (**13**)



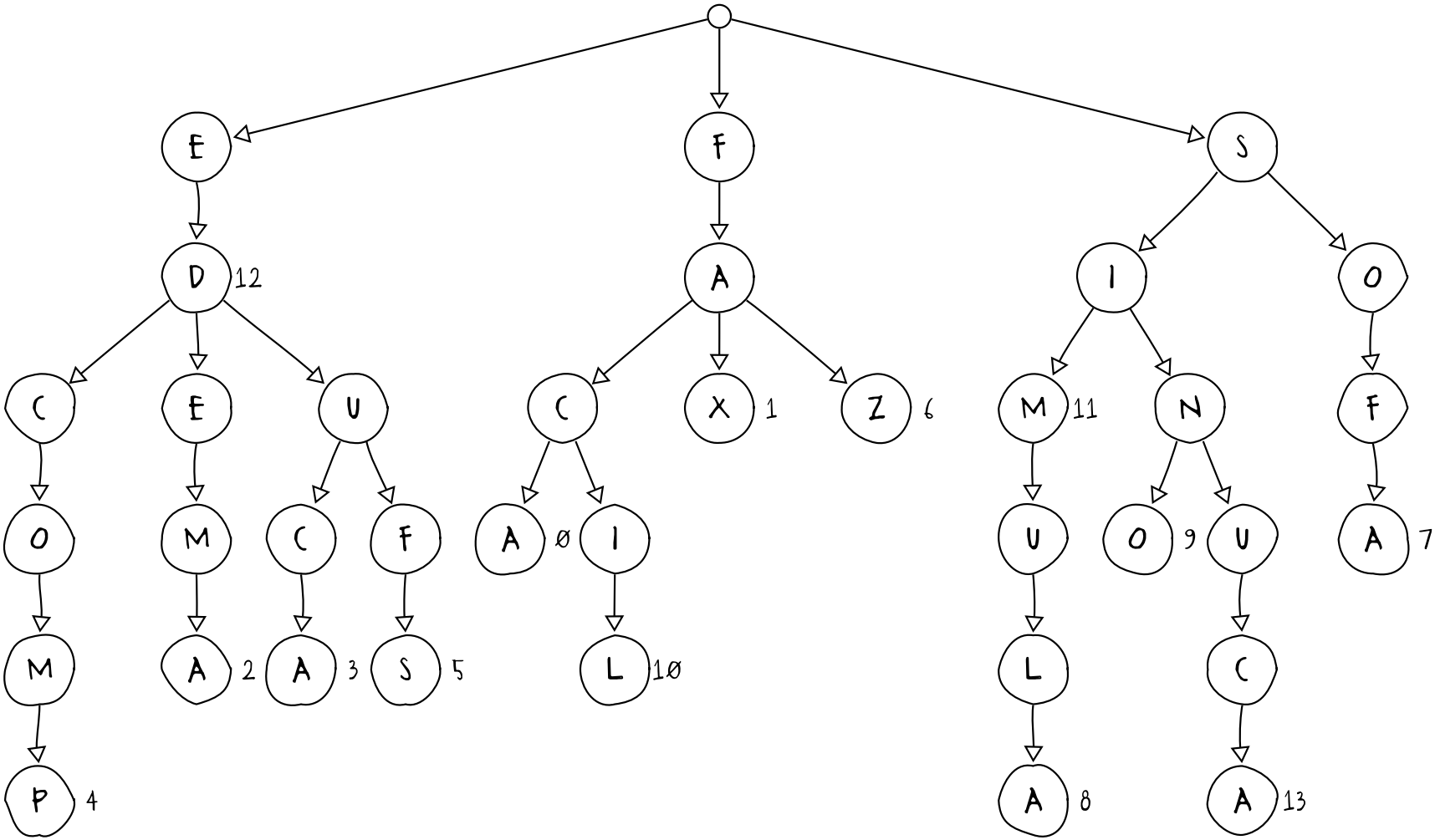
# Árvore de prefixo

- Implementação em C
  - Inserção de cadeia de caracteres

```
1  // Procedimento de inserção de cadeia
2  void insercao(no* x, char* p, uint32_t d, uint32_t v) {
3      // Verificando se x é nulo
4      if(x == NULL)
5          // Criando nó
6          x = criar_no();
7      // Verificando se todos os caracteres da cadeia já
           foram inseridos
8      if(d == strlen(p)) {
9          // Atribuindo índice
10         x->V = (uint32_t*)(malloc(sizeof(uint32_t)));
11         *x->V = v;
12     }
13     // Chamada recursiva no próximo nível
14     else
15         insercao(x->P[indice(p, d)], p, d + 1, v);
16 }
```

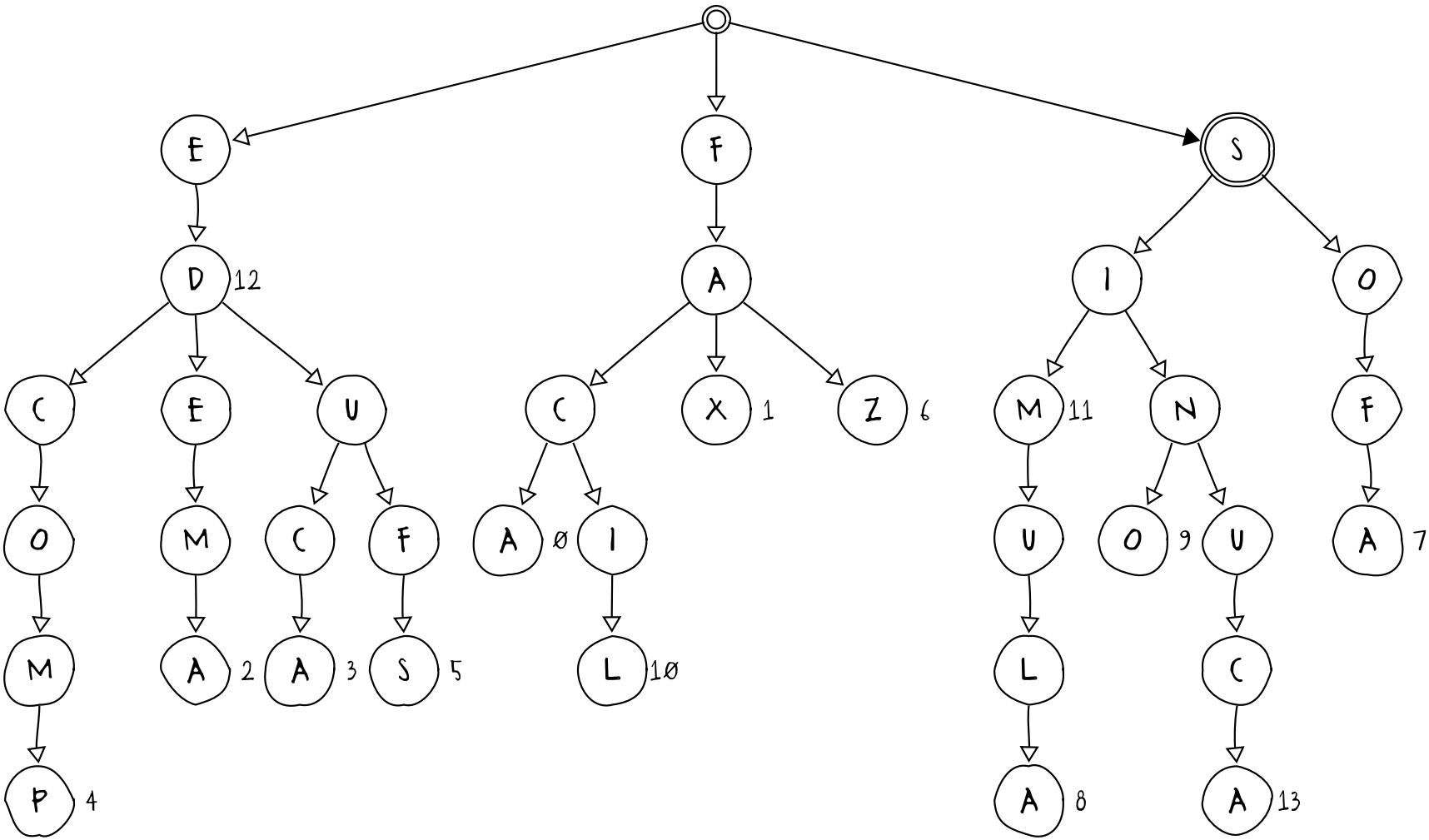
# Árvore de prefixo

- Operação de remoção
  - Parâmetro: sim



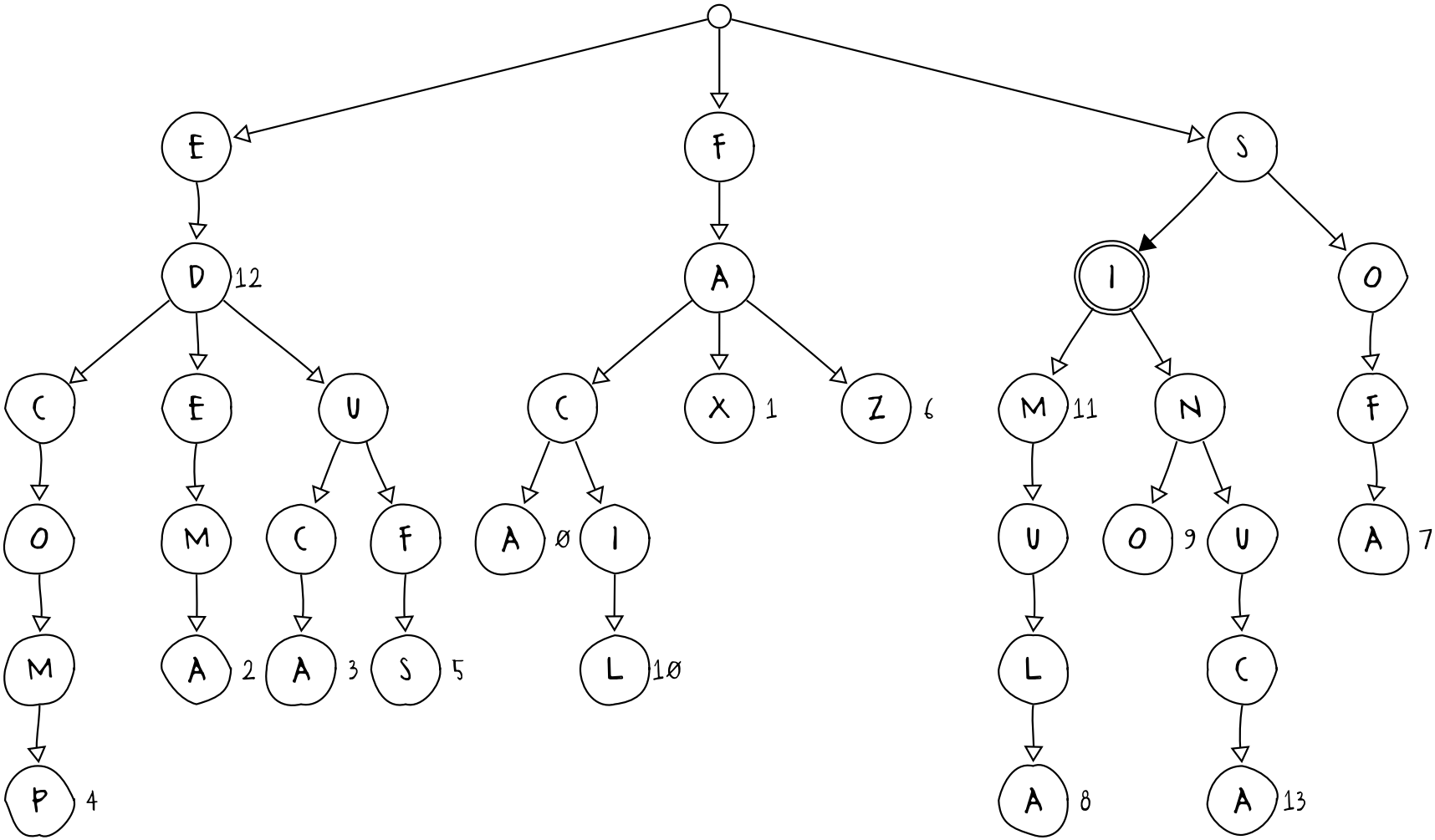
# Árvore de prefixo

- Operação de remoção
  - Parâmetro: sim



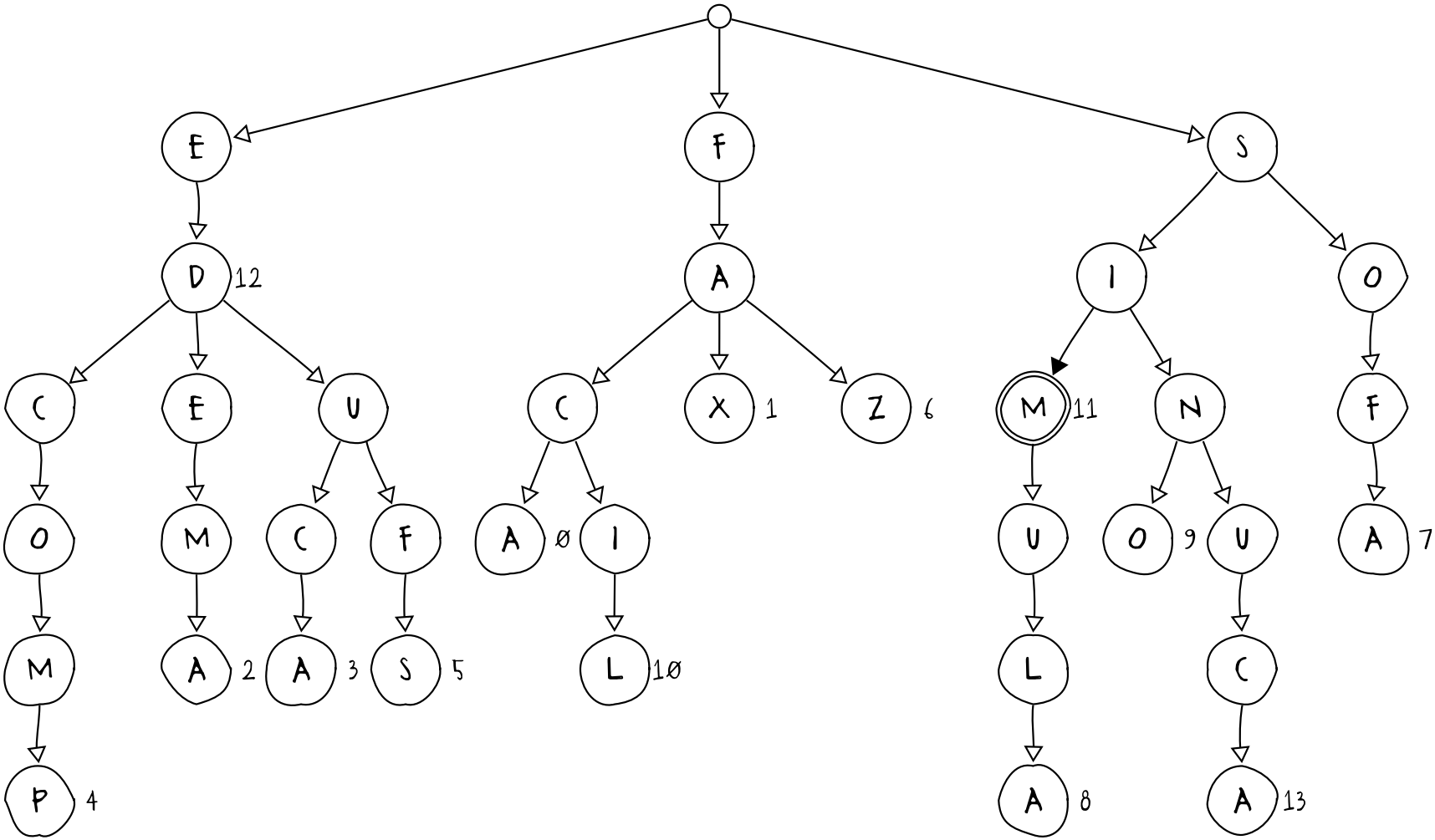
# Árvore de prefixo

- Operação de remoção
  - Parâmetro: sim



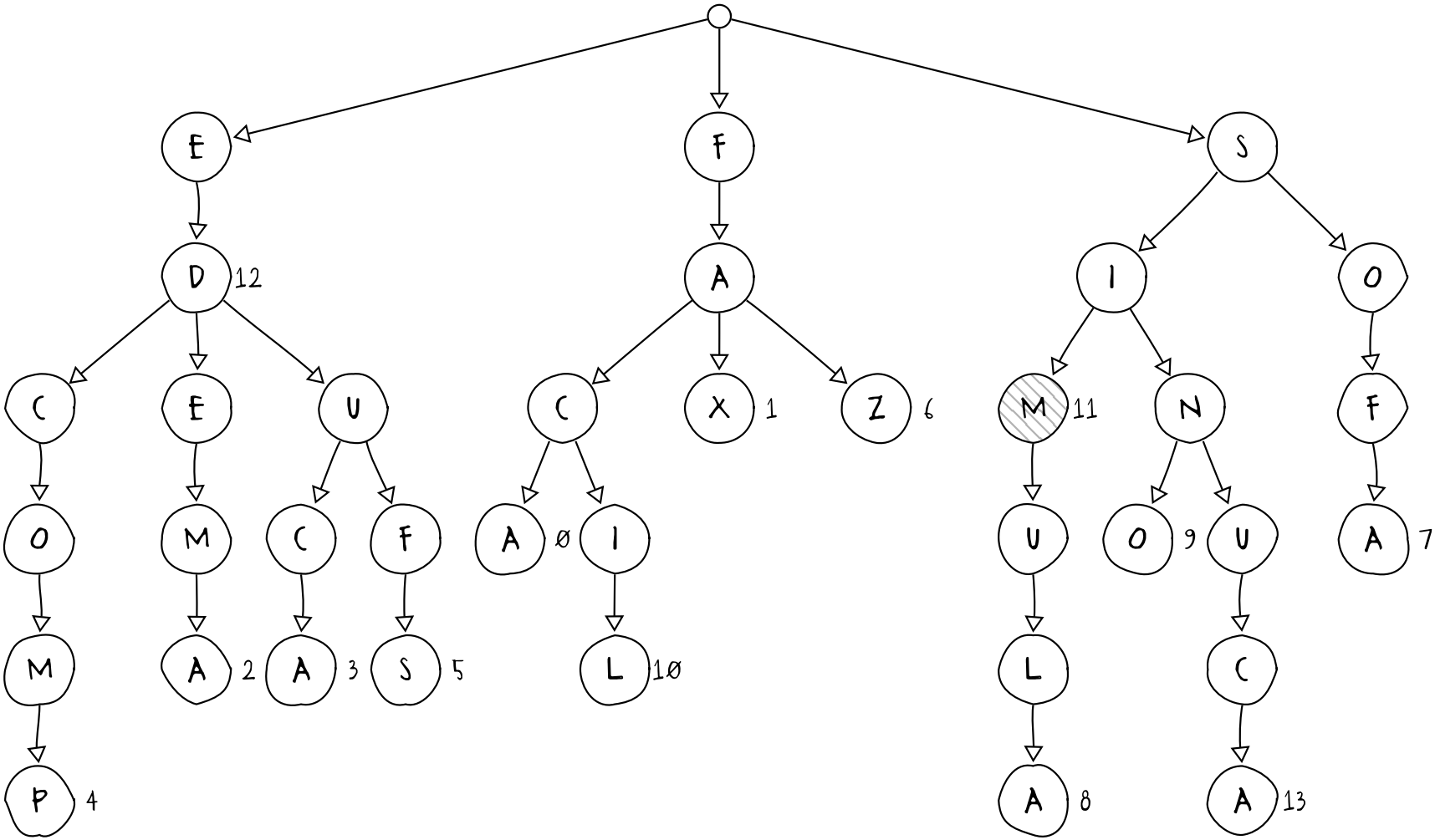
# Árvore de prefixo

- Operação de remoção
  - Parâmetro: sim



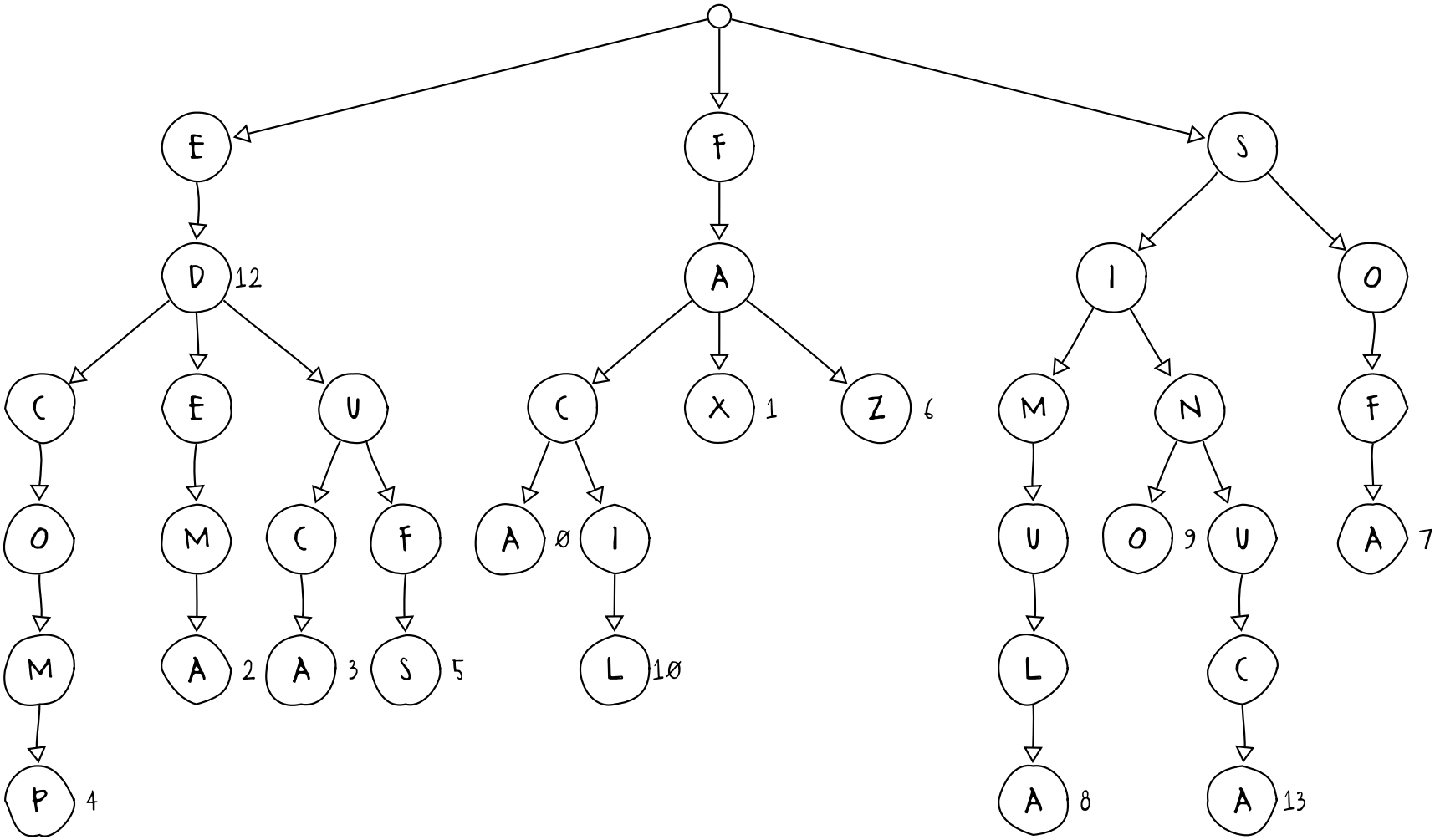
# Árvore de prefixo

- Operação de remoção
  - Parâmetro: sim



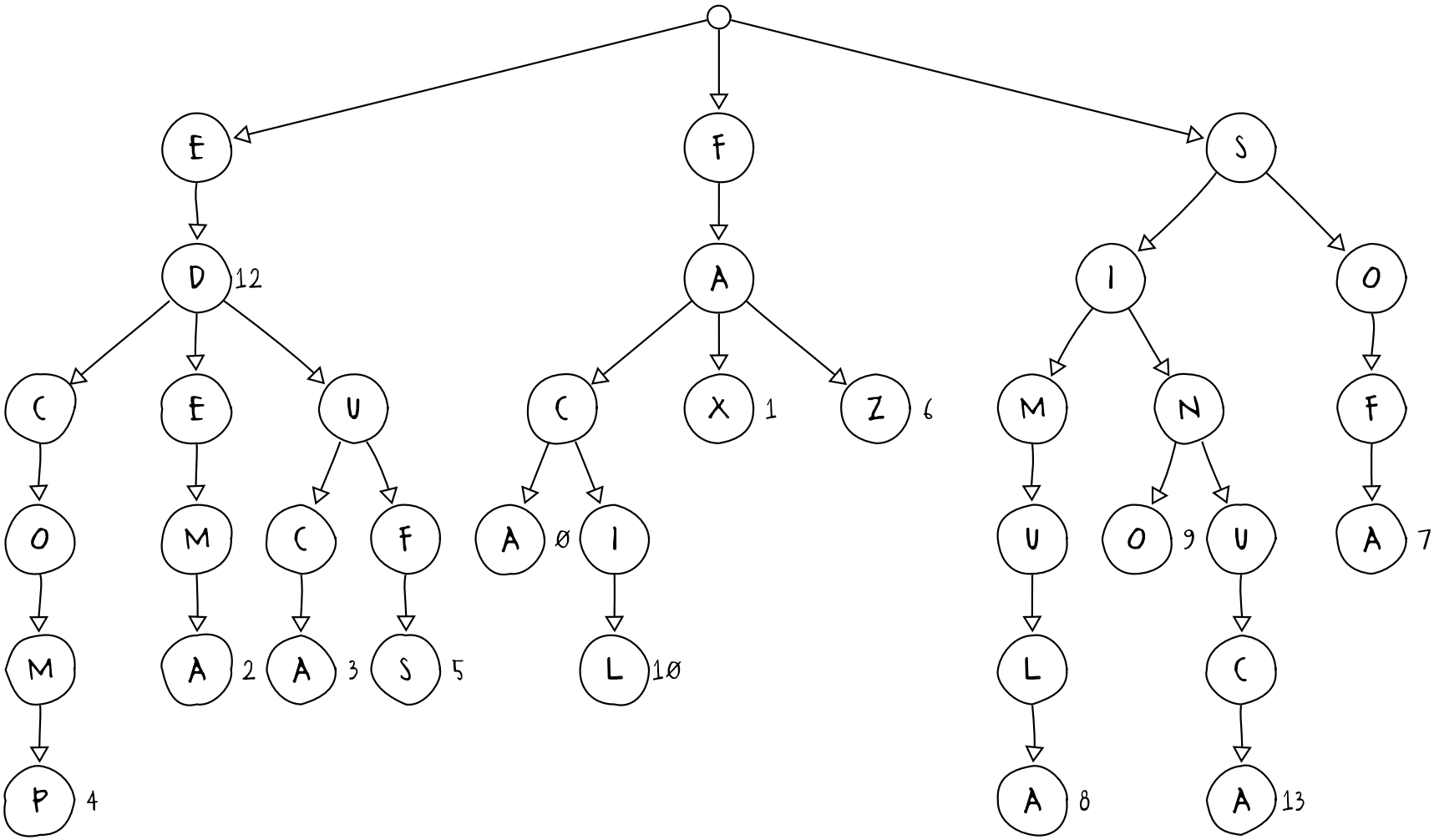
# Árvore de prefixo

- Operação de remoção
  - Parâmetro: sim



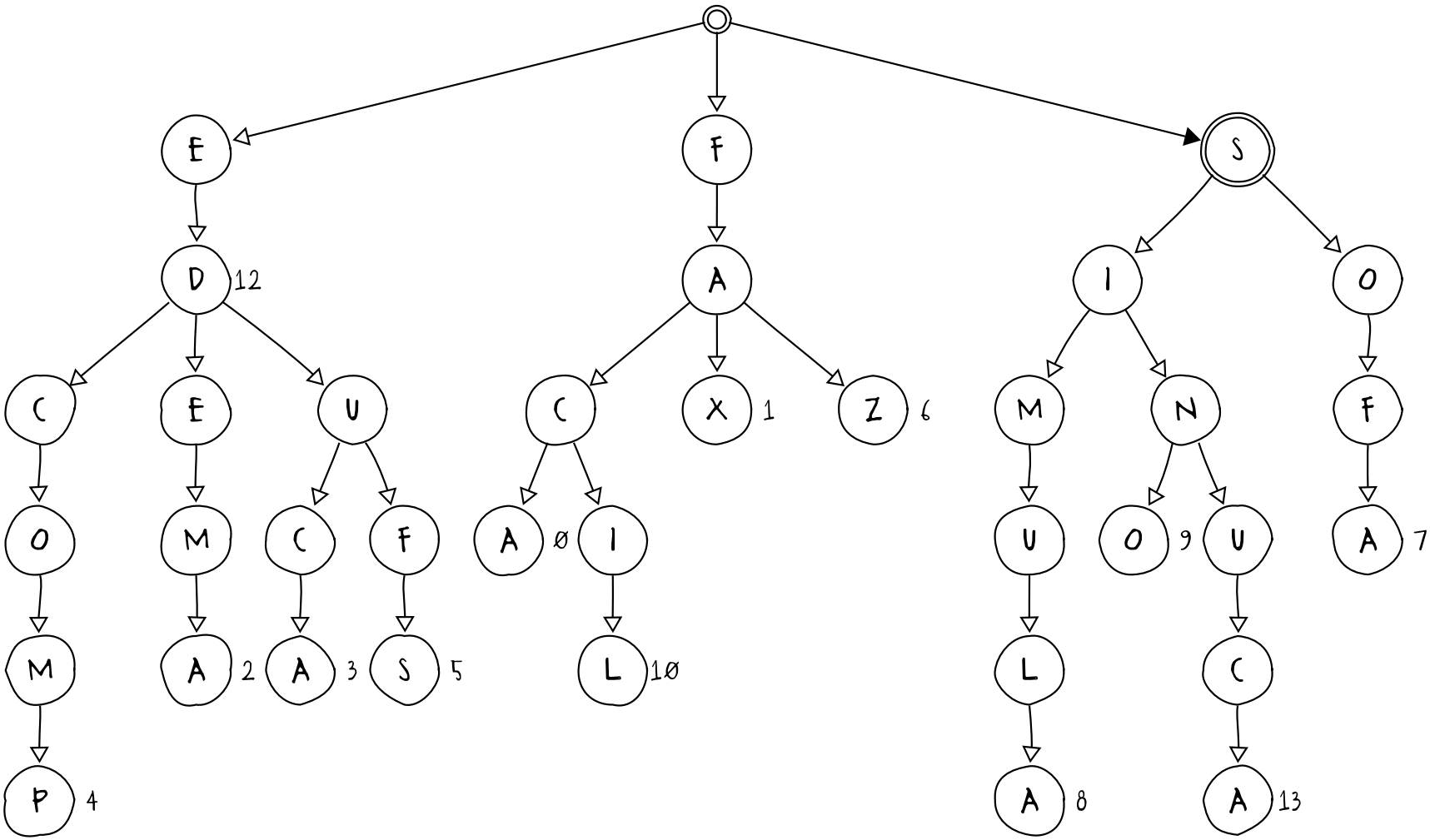
# Árvore de prefixo

- Operação de remoção
  - Parâmetro: simula



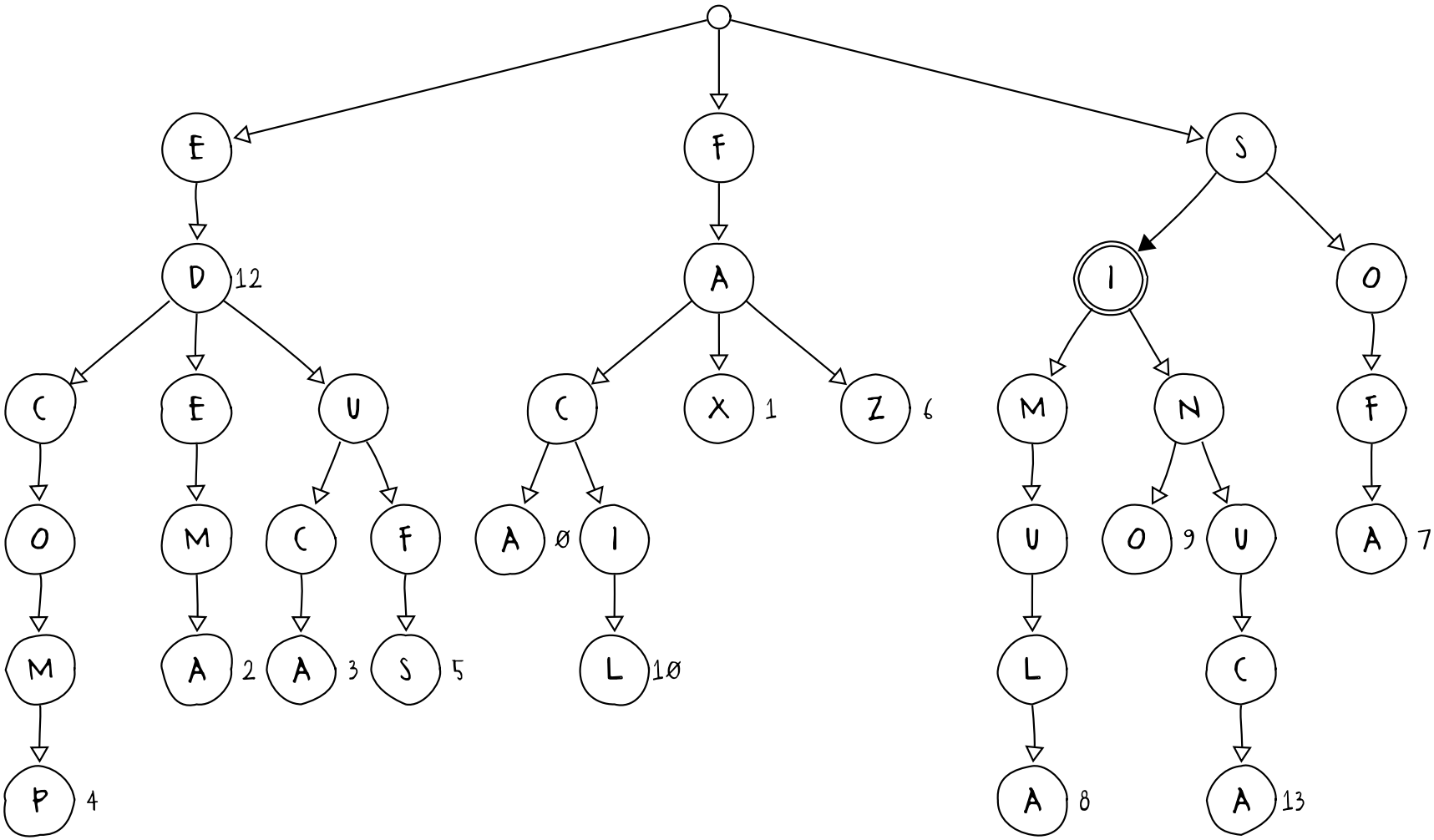
# Árvore de prefixo

- Operação de remoção
  - Parâmetro: simula



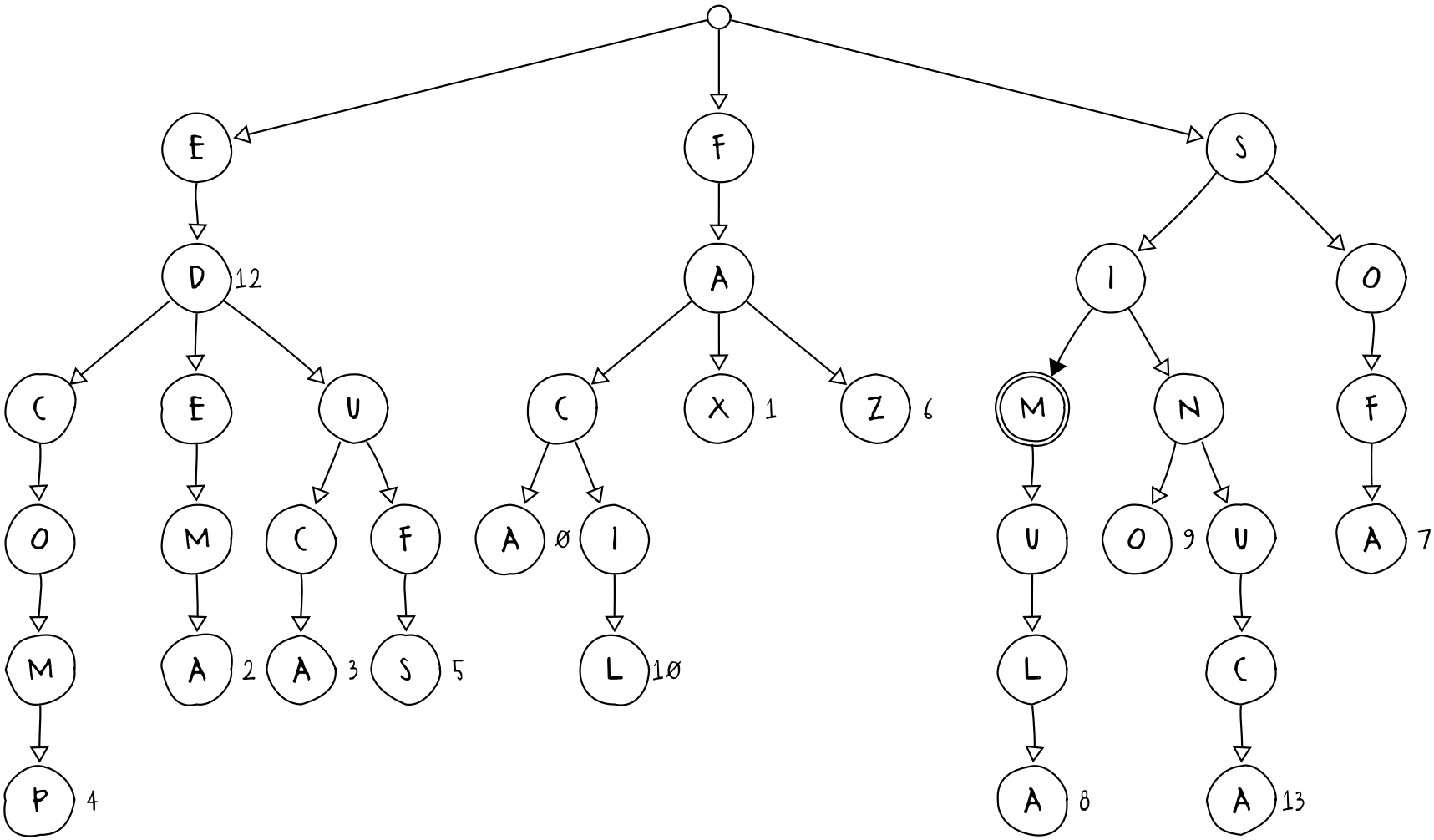
# Árvore de prefixo

- Operação de remoção
  - Parâmetro: simula



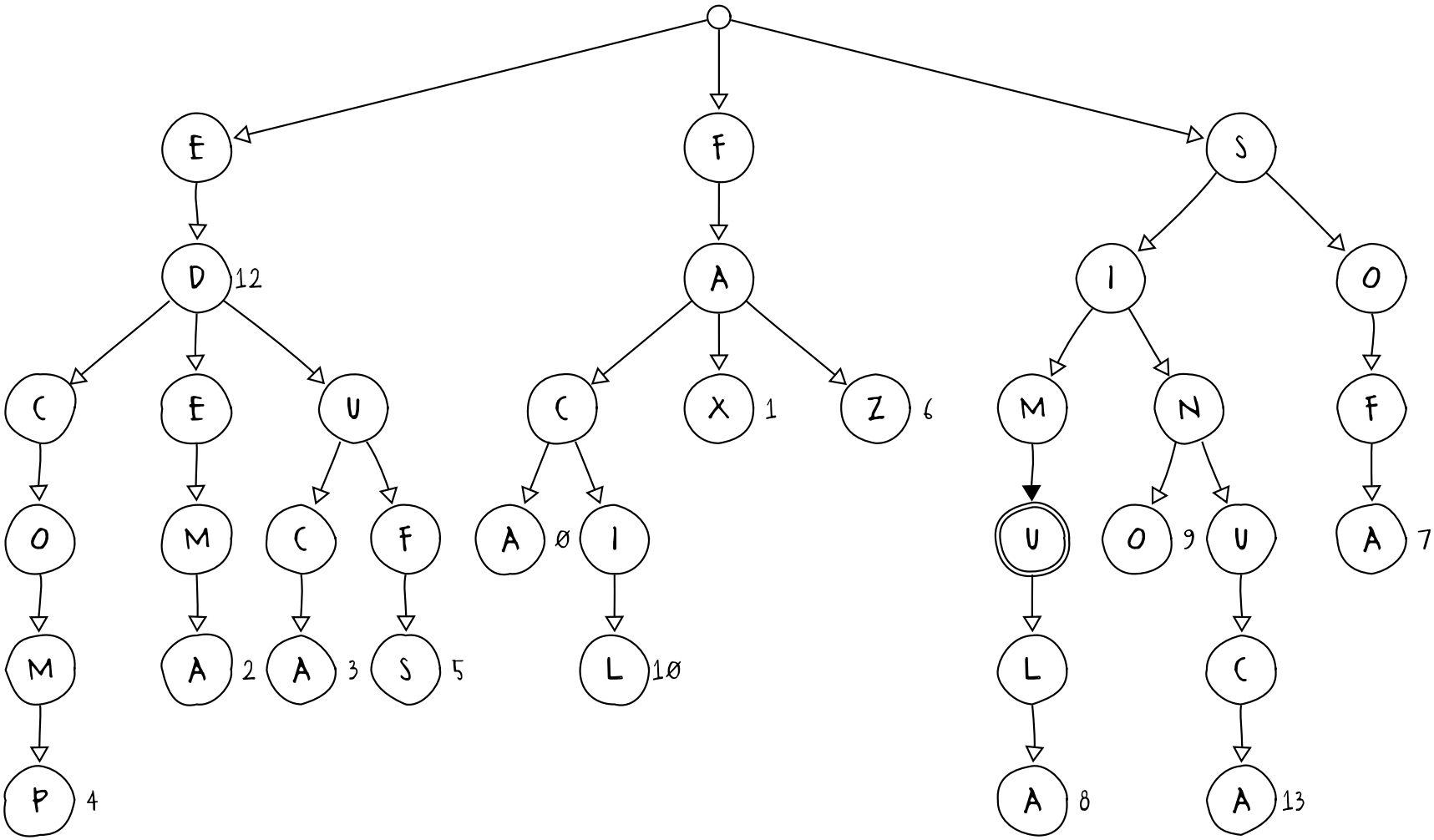
# Árvore de prefixo

- Operação de remoção
  - Parâmetro: simula



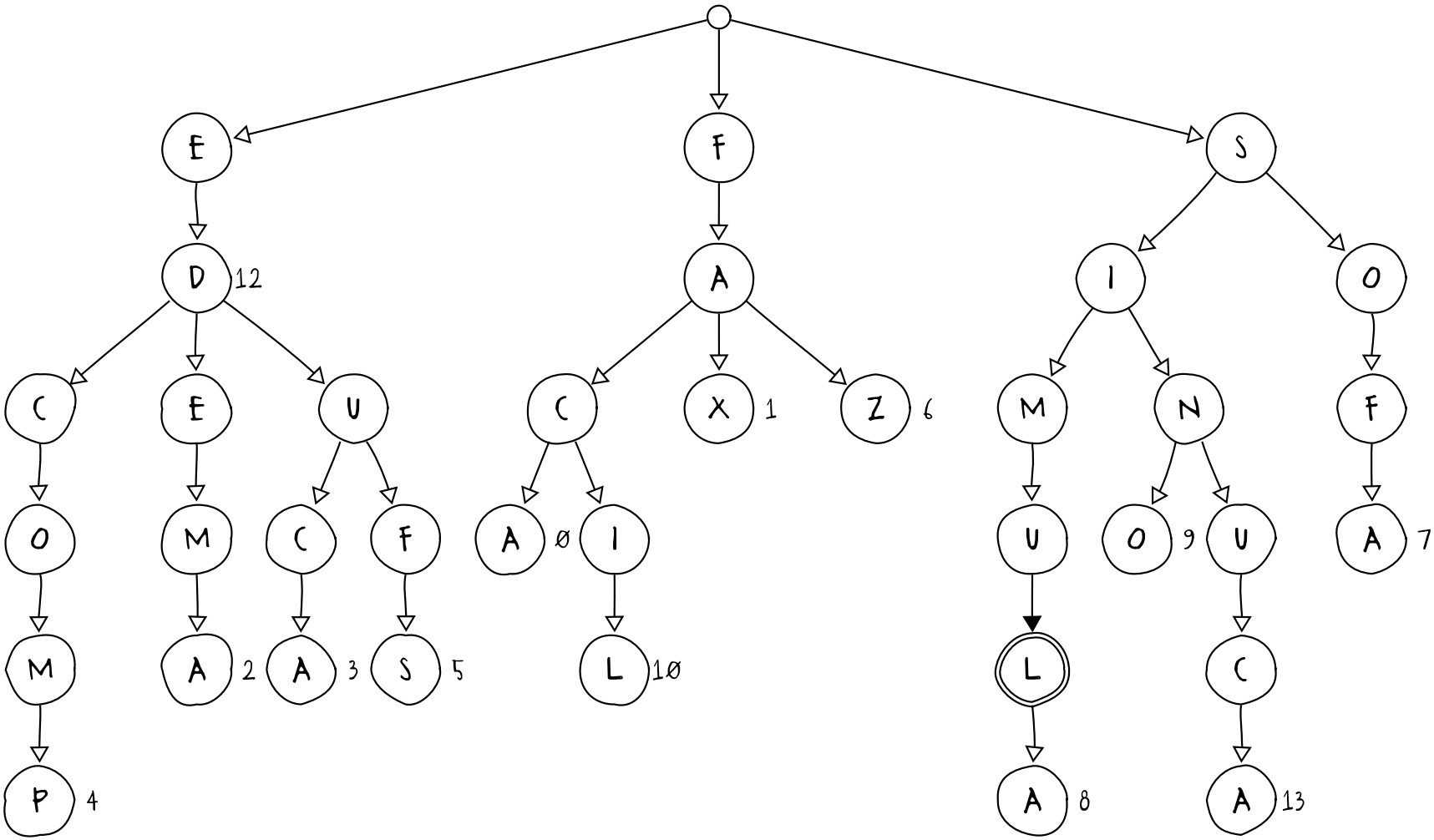
# Árvore de prefixo

- Operação de remoção
  - Parâmetro: simula



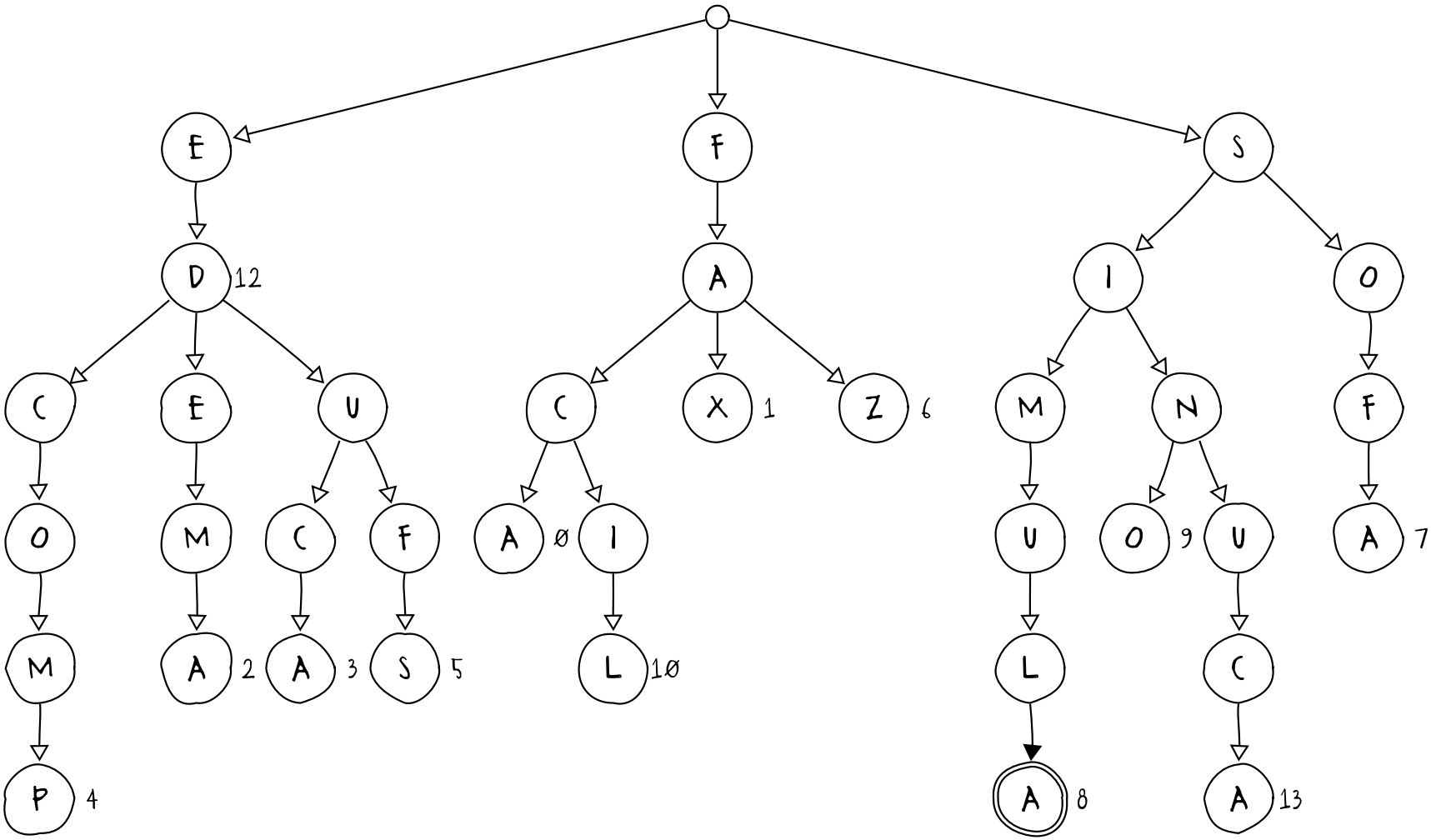
# Árvore de prefixo

- Operação de remoção
  - Parâmetro: simula



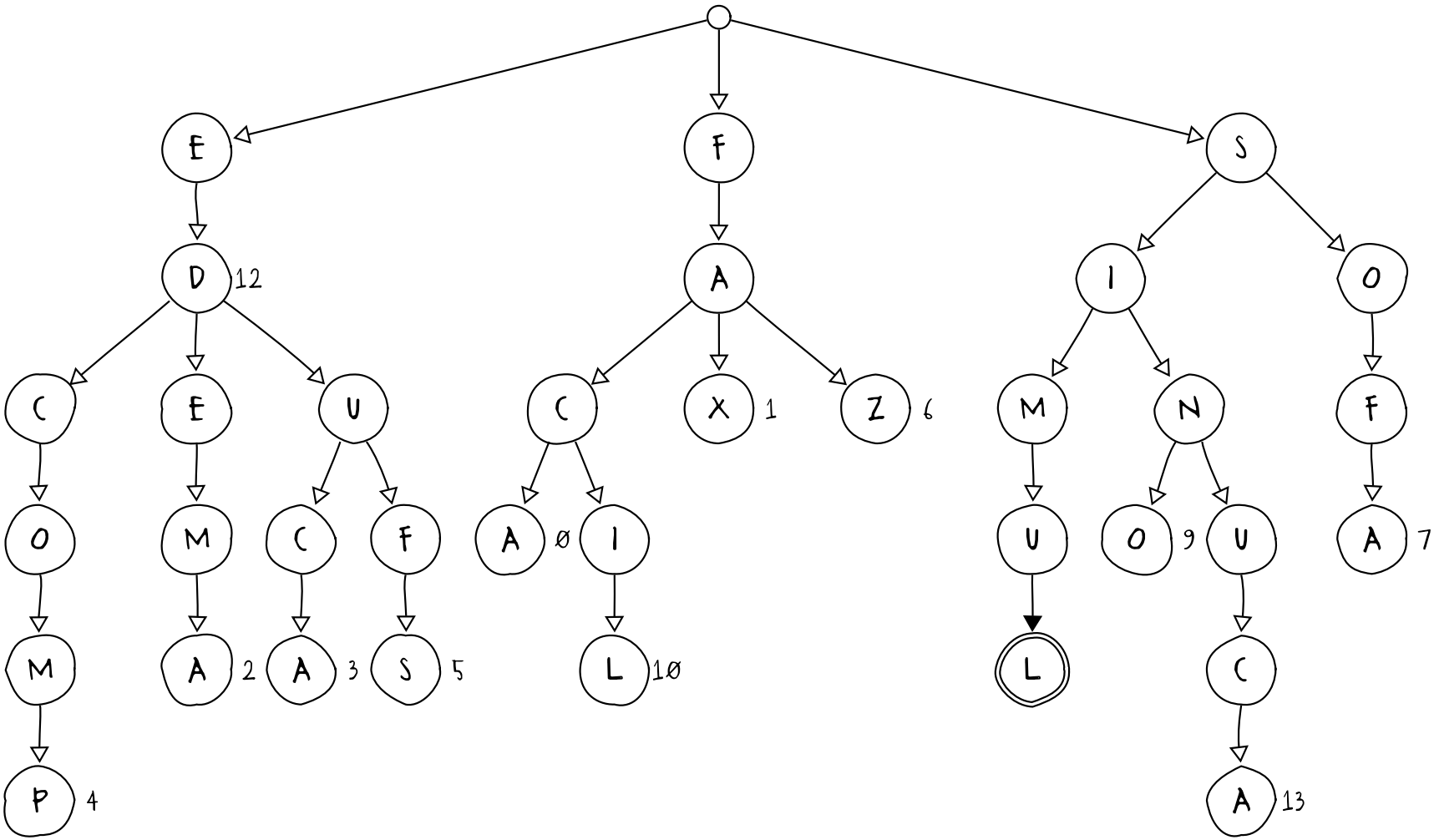
# Árvore de prefixo

- Operação de remoção
  - Parâmetro: simula



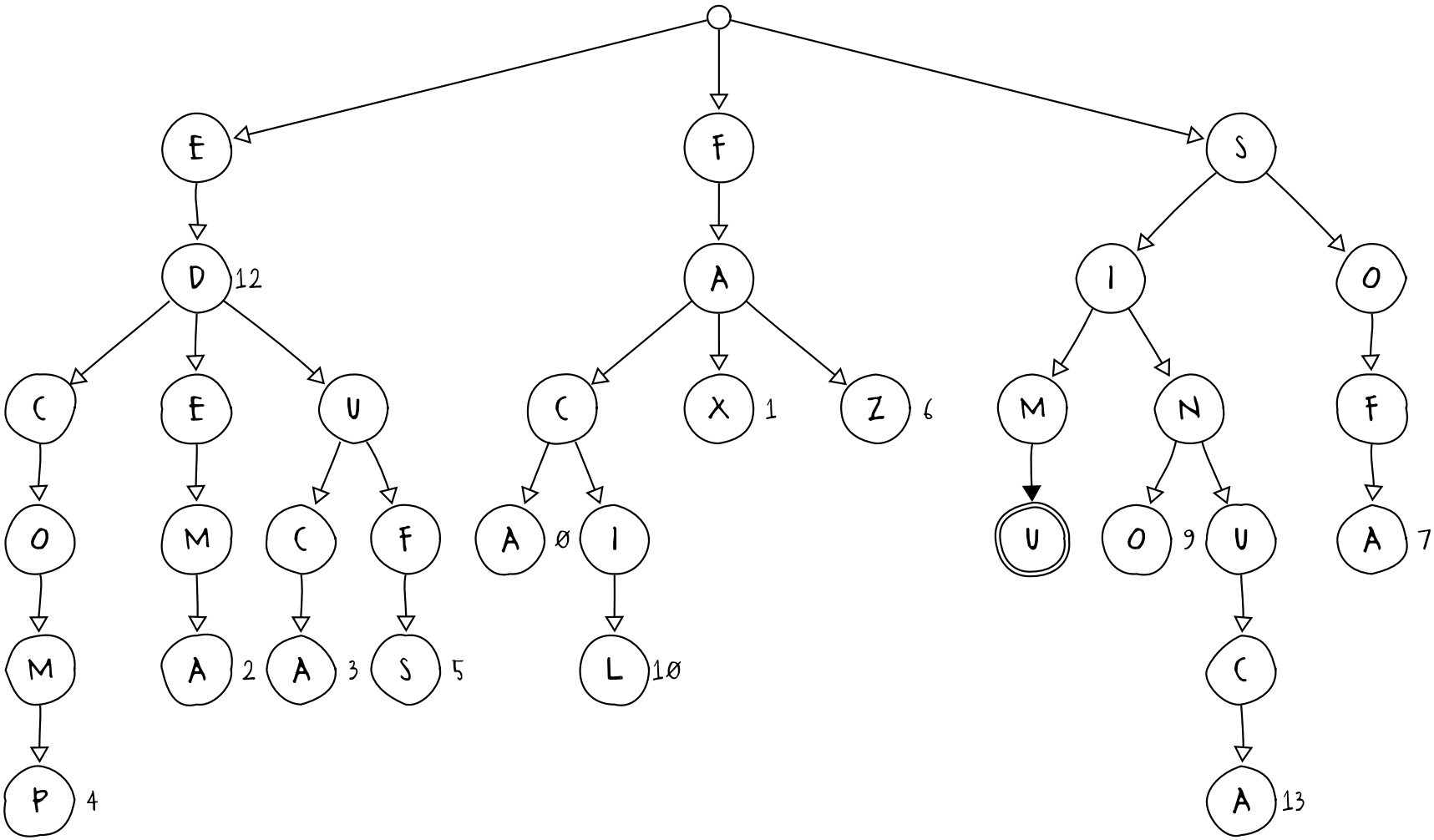
# Árvore de prefixo

- Operação de remoção
  - Parâmetro: simula



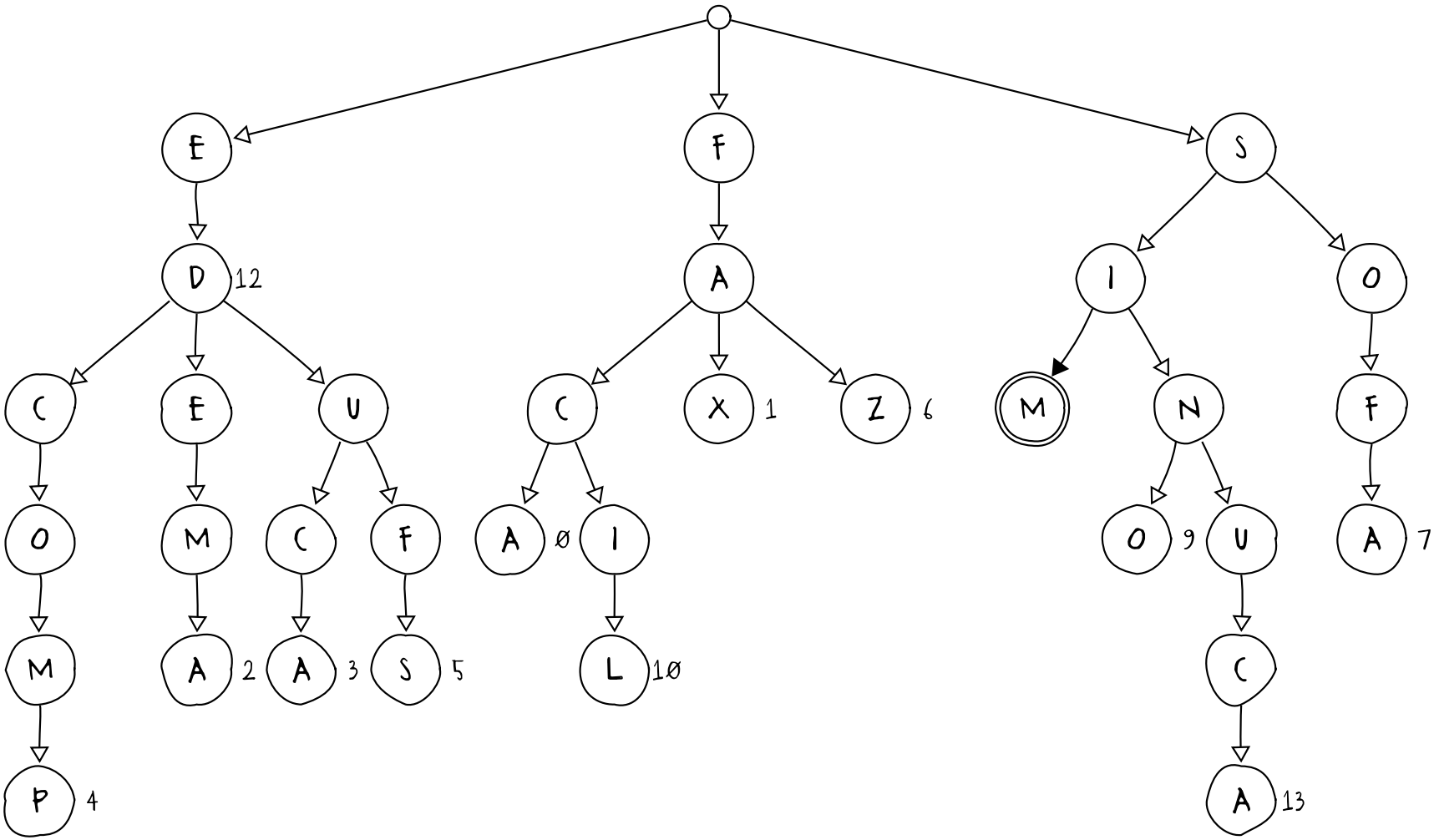
# Árvore de prefixo

- Operação de remoção
  - Parâmetro: simula



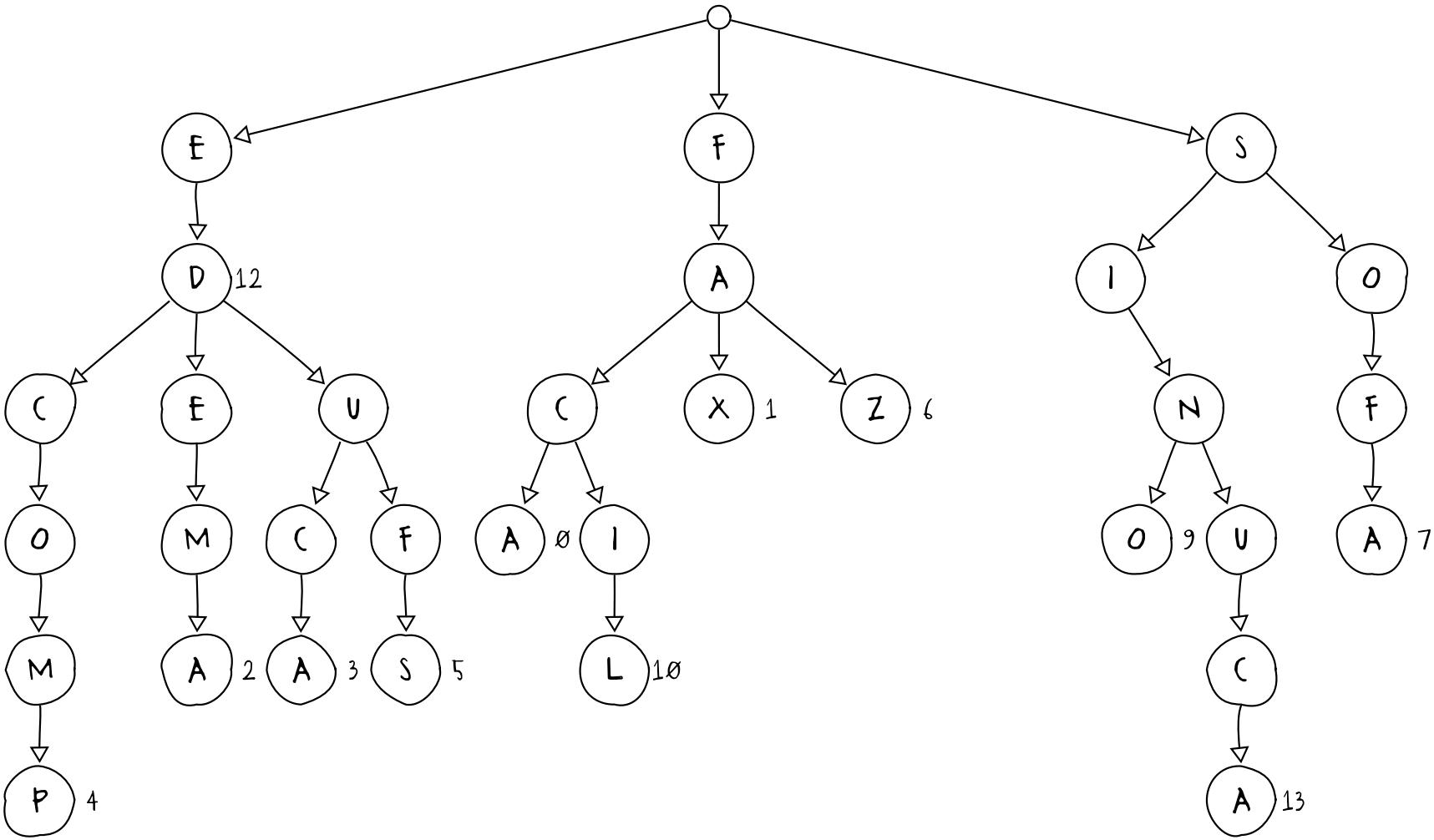
# Árvore de prefixo

- Operação de remoção
  - Parâmetro: simula



# Árvore de prefixo

- Operação de remoção
  - Parâmetro: simula



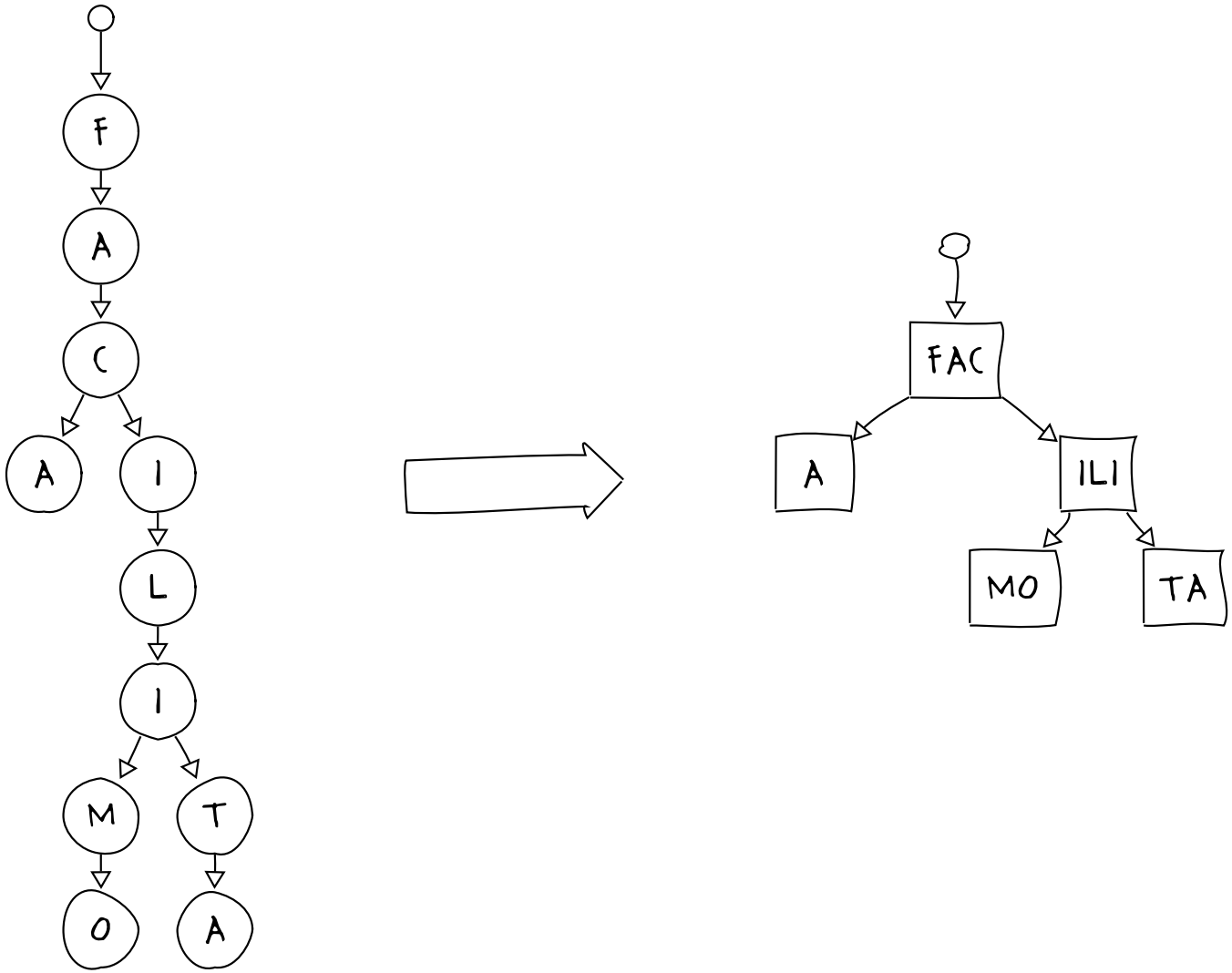
# Árvore de prefixo

- Implementação em C
  - Remoção de cadeia de caracteres

```
1  // Função de remoção de cadeia
2  no* remocao(no* x, char* p, uint32_t d) {
3      if(x != NULL) {
4          if(d == strlen(p)) {
5              free(x->V);
6              x->V = NULL;
7          }
8          else {
9              uint32_t i = indice(p, d);
10             x->P[i] = remocao(x->P[i], p, d + 1);
11             if(x->V == NULL && sem_filhos(x)) {
12                 free(x);
13                 x = NULL;
14             }
15         }
16     }
17     return x;
18 }
```

# Árvore de prefixo

- Análise de complexidade
  - Impacto de cadeias longas no espaço



# Árvore de prefixo

- Análise de complexidade
  - Considerando que $m$ é o tamanho médio das cadeias e que $n$ é a quantidade total de cadeias

| Aplicação | Exemplo | k | m | n |
|---|---|---|---|---|
| Placa | AB123CD | 36 | 7 | $\approx 4,6 \times 10^8$ |
| Telefonia | 987651234 | 10 | 9 | $\approx 10^9$ |
| Texto | aBcDefGHij | 52 | 10 | $\approx 1,45 \times 10^{17}$ |

# Árvore de prefixo

- Análise de complexidade
  - Considerando que $m$ é o tamanho médio das cadeias e que $n$ é a quantidade total de cadeias

| Aplicação | Exemplo | k | m | n |
|---|---|---|---|---|
| Placa | AB123CD | 36 | 7 | $\approx 4,6 \times 10^8$ |
| Telefonia | 987651234 | 10 | 9 | $\approx 10^9$ |
| Texto | aBcDefGHij | 52 | 10 | $\approx 1,45 \times 10^{17}$ |

  - Espaço: $\Omega(kn)$ e $O(kmn)$

# Árvore de prefixo

- Análise de complexidade
  - Considerando que $m$ é o tamanho médio das cadeias e que $n$ é a quantidade total de cadeias

| Aplicação | Exemplo | k | m | n |
|---|---|---|---|---|
| Placa | AB123CD | 36 | 7 | $\approx 4,6 \times 10^8$ |
| Telefonia | 987651234 | 10 | 9 | $\approx 10^9$ |
| Texto | aBcDefGHij | 52 | 10 | $\approx 1,45 \times 10^{17}$ |

  - Espaço: $\Omega(kn)$ e $O(kmn)$
  - Tempo: $\Omega(1)$ e $O(m)$

# Exemplo

- Construa uma árvore de prefixo com $[a - z]$
  - Insira as cadeias *trivial*, *fax*, *nada*, *facil*, *fazenda*, *alfabeto*, *alfa*, *beta*, *faz*, *trilegal*
  - Realize a remoção das cadeias *faz* e *nada*
  - Verifique quais técnicas podem ser utilizadas para reduzir o espaço utilizado pelas árvores de prefixo, observando os seus respectivos impactos no desempenho

# Exercício

- A empresa de tecnologia Poxim Tech está desenvolvendo uma API para sugerir termos utilizados nas requisições em seus sistemas, considerando todas as palavras contidas em sua base de dados
  - Os termos são compostos por uma única palavra com letras minúsculas com até 20 caracteres
  - São consideradas como sugestões termos com até o dobro do tamanho do prefixo
    - Termo: abc
    - Sugestões: abab, abcdef, ac, ...

# Exercício

- Formato de arquivo de entrada
  - [#*Quantidade de termos*($n$)]
  - [$Termo_1$]
  - $\vdots$
  - [$Termo_n$]
  - [#*Número de requisições*($x$)]
  - [$Requisição_1$]
  - $\vdots$
  - [$Requisição_x$]

# Exercício

- Formato de arquivo de entrada

```
8
facil
simples
trivial
banal
bacana
banano
banda
facilimo
5
facin
ban
exemplo
trivial
bacaninha
```

# Exercício

- Formato de arquivo de saída
  - Os termos sugeridos possuem até o dobro do tamanho do prefixo do termo requisitado

```
1 facin:facil,facilimo
2 ban:banal,banano,banda
3 exemplo:-
4 trivial:trivial
5 bacaninha:bacana
```