Inserção:

Estrutura do nó:

```
typedef struct Node{
    struct Node *children[A_LENGTH];
    int value;
    int is_terminal;
}Trie;
```

Inserção:

Código:

```
Trie *insert(Trie *trie, int value, char *key){
    if(key[0] == '\0') return trie;

    if(!trie) trie = create();

    int iKey = (int) key[0];

    if(key[1]=='\0'){
        if(!trie->children[iKey])
            trie->children[iKey] = create();

        trie->children[iKey]->value = value;
        trie->children[iKey]->is_terminal = 1;
    }else
        trie->children[iKey] = insert(trie->children[iKey], value,
&(key[1]));


    return trie;
}
```

- Melhor caso da inserção:

No melhor caso, a inserção ocorrerá em uma chave cujo nó já existe. Nesse caso, a condição que checa se o nó não existe nunca é atendida.

7 linhas são executadas no caso base. E a cada caso recursivo, 4 linhas são executadas e a função novamente é chamada. A relação de recorrência que modela esse comportamento é:

$T(1) = 7$
$T(n) = 4 + T(n - 1)$

Resolvendo essa relação de recorrência, temos que

$T(n) = 4 + 4 + T(n - 2) = 4(n - 1) + T(1)$
$T(n) = 4n - 4 + 7 = 4n + 3$

Assim, dada a seguinte inequação

$c_1 n < 4n + 3 < c_2 n$, em que $c_1, c_2 > 0$
$c_1 < 4 + \frac{3}{n} < c_2$

Assumindo valores positivos para n (dada a natureza da nossa análise), temos que

$4 < 4 + \frac{3}{n} \Rightarrow c_1 < 4$

além disso, para $n > 3$,

$4 + \frac{3}{n} < 5$

nesse caso, c_2 pode assumir qualquer valor maior ou igual a 5.

Portanto, $4n + 3 = \Theta(n)$, e desse modo, a complexidade da implementação da inserção na TRIE no melhor caso é $\Theta(n)$

- Pior caso da inserção:

No pior caso, não existe nem mesmo algum nó cuja chave comece com a letra inicial da chave inserida. Nesse caso, a condição que checa se o nó não existe sempre é atendida.

$2\alpha_t + 14$ linhas são executadas no caso base, sendo α_t o tamanho do alfabeto. E a cada caso recursivo, $\alpha_t + 7$ linhas são executadas e a função novamente é chamada. A relação de recorrência que modela esse comportamento é:

$$T(1) = 2\alpha_t + 14$$
$$T(n) = \alpha_t + 7 + T(n - 1)$$

Resolvendo essa relação de recorrência, temos que

$$T(n) = \alpha_t + 7 + \alpha_t + 7 + T(n - 2) = (\alpha_t + 7)(n - 1) + T(1)$$
$$T(n) = (\alpha_t + 7)(n - 1) + 2\alpha_t + 14 = \alpha_t n - \alpha_t + 2\alpha_t + 7n - 7 + 14$$
$$T(n) = \alpha_t n + 7n + \alpha_t + 7$$

Assim, dada a seguinte inequação

$c_1 n < \alpha_t n + 7n + \alpha_t + 7 < c_2 n$, em que $c_1, c_2 > 0$

$$c_1 < \alpha_t + 7 + \frac{\alpha_t + 7}{n} < c_2$$

Assumindo valores positivos para n (dada a natureza da nossa análise), temos que

$$7 < \alpha_t + 7 + \frac{\alpha_t + 7}{n} \Rightarrow c_1 < 7$$

além disso, para $n > \alpha_t + 7$,

$$\alpha_t + 7 + \frac{\alpha_t + 7}{n} < \alpha_t + 7$$

nesse caso, c_2 pode assumir qualquer valor maior ou igual a $\alpha_t + 7$

Portanto, $\alpha_t n + 7n + \alpha_t + 7 = \Theta(n)$, e desse modo, a complexidade da implementação da inserção na TRIE no pior caso é $\Theta(n)$.

Remoção:

Código:

```
Trie *t_remove(Trie *trie, char *key){
    if(!trie) return trie;

    Trie *aux = NULL;

    if(key[0]=='\0'){
        trie->is_terminal = 0;
    }
    else {
        int iKey = (int) key[0];
        aux = t_remove(trie->children[iKey], &(key[1]));
        trie->children[iKey] = aux;
    }

    if(aux || trie->is_terminal) return trie;
```

```
    for(int i = 0; i<A_LENGTH; i++){
        if(trie->children[i]){
            return trie;
        }
    }

    free(trie->children);
    free(trie);

    return NULL;
}
```

- Melhor caso da remoção:

No melhor caso, o nó a ser removido não é uma folha, portanto não será retornado NULL no caso base, o que fará com a as outras chamadas nunca entrem no último for.

Nesse caso, no caso base são executadas 4 linhas, e no caso recursivo, 7 e a chamada recursiva. Portanto a relação de recorrência seria:

$T(1) = 4$
$T(n) = 7 + T(n - 1)$

Resolvendo essa relação de recorrência, temos que

$T(n) = 7 + 7 + T(n - 2) = 7(n - 1) + T(1)$
$T(n) = 7n - 7 + 4 = 7n - 3$

Assim, dada a seguinte inequação

$c_1 n < 7n - 3 < c_2 n$, em que $c_1, c_2 > 0$
$c_1 < 7 - \frac{3}{n} < c_2$

Assumindo valores positivos e inteiros para n (dada a natureza da nossa análise), temos que

$3 < 7n - 3 \Rightarrow c_1 \leq 3$

além disso, para $n > 3$,

$7 - \frac{3}{n} < 7$

nesse caso, c_2 pode assumir qualquer valor maior ou igual a 7.

Portanto, $7n - 3 = \Theta(n)$, e desse modo, a complexidade da implementação da remoção na TRIE no melhor caso é $\Theta(n)$

- Pior caso da remoção:

No pior caso, nenhum nó prefixo do objetivo é terminal, e o objetivo é uma folha. Nesse caso, o último for é sempre executado.

$\alpha_t + 6$ linhas são executadas no caso base. E a cada caso recursivo, $\alpha_t + 8$ linhas são executadas e a função novamente é chamada. A relação de recorrência que modela esse comportamento é:

$T(1) = \alpha_t + 6$
$T(n) = \alpha_t + 8 + T(n - 1)$

Resolvendo essa relação de recorrência, temos que

$T(n) = \alpha_t + 8 + \alpha_t + 8 + T(n - 2) = (\alpha_t + 8)(n - 1) + T(1)$
$T(n) = (\alpha_t + 8)(n - 1) + \alpha_t + 6 = \alpha_t n - \alpha_t + \alpha_t + 8n - 8 + 6$

$T(n) = \alpha_t n + 8n - 2$

Assim, dada a seguinte inequação

$c_1 n < \alpha_t n + 8n - 2 < c_2 n$, em que $c_1, c_2 > 0$

$c_1 < \alpha_t + 8 - \frac{2}{n} < c_2$

Assumindo valores positivos para n (dada a natureza da nossa análise), temos que

$6 < \alpha_t + 8 - \frac{2}{n} \Rightarrow c_1 < 6$ (considerando que o alfabeto não possua caracteres [no nosso caso, possui 128])

além disso, para $n > 2$,

$\alpha_t + 8 - \frac{2}{n} < \alpha_t + 8$

nesse caso, c_2 pode assumir qualquer valor maior ou igual a $\alpha_t + 8$

Portanto, $\alpha_t n + 8n - 2 = \Theta(n)$, e desse modo, a complexidade da implementação da remoção na TRIE no pior caso é $\Theta(n)$.

Busca:

Código:

```
Trie* search(Trie *trie, char *key){
    if(!trie) return NULL;

    if(key[0]=='\0'){
        return (trie->is_terminal)? trie : NULL;
    }
```

```
    int iKey = (int) key[0];
    return search(trie->children[iKey], &(key[1]));
}
```

- Melhor caso da busca:

Não existe na TRIE nenhuma chave que comece com o primeiro caractere da chave buscada.

Desse modo, para qualquer entrada, temos que será executada a função inteira uma vez, e na segunda chamada recursiva ela retornará NULL. Ou seja, para qualquer entrada, temos 6 linhas sendo executadas.

Podemos dizer que $6 = \Theta(k)$, ou seja, existem constantes tais que $6c1 < 6 < 6c_2$. Então a complexidade na implementação da busca na TRIE no melhor caso é $\Theta(k)$.

- Pior caso da busca:

A chave buscada existe.

Nesse caso, temos 3 linhas sendo executadas no caso base, e mais 3 linhas no caso recursivo, além da chamada recursiva. Então temos a seguinte relação de recorrência:

$T(1) = 3$
$T(n) = 3 + T(n - 1)$

Resolvendo essa relação de recorrência, temos que

$T(n) = 3 + 3 + T(n - 2) = 3(n - 1) + T(1)$
$T(n) = 3n - 3 + 3 = 3n$

Assim, dada a seguinte inequação

$c_1 n \ < \ 3n < c_2 n$, em que $c_1, c_2 > 0$
$c_1 < 3 < c_2$

Ou seja, c_1 pode assumir qualquer valor positivo menor que 3, e c_2 pode assumir qualquer valor maior que 3.

Portanto, $3n = \Theta(n)$, e desse modo, a complexidade da implementação da busca na TRIE no pior caso é $\Theta(n)$