MaratonaCIn

SELETIVA 2020

Strings

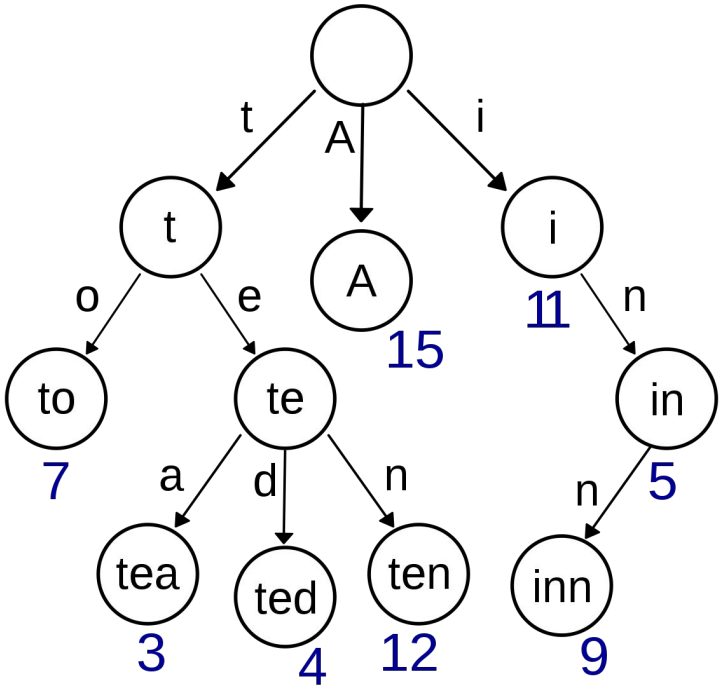
Aula 9

Trie

- Árvore de prefixos
- Substrings
- Como construir/representar

Trie

MaratonaCIn



Trie


```
int trie[ms][sigma], terminal[ms], z;

void init() {
  memset(trie[0], -1, sizeof trie[0]);
  z = 1;
}
```

Trie


```
void insert(string &p) {
  int cur = 0;
  for(int i = 0; i < p.size(); i++) {
    int id = get_id(p[i]);
    if(trie[cur][id] == -1) {
      memset(trie[z], -1, sizeof trie[z]);
      trie[cur][id] = z++;
    }
    cur = trie[cur][id];
  }
  terminal[cur]++;
}
```

Trie


```
int count(string &p) {
  int cur = 0;
  for(int i = 0; i < p.size(); i++) {
    int id = get_id(p[i]);
    if(trie[cur][id] == -1) {
      return false;
    }
    cur = trie[cur][id];
  }
  return terminal[cur];
}
```

Trie

MaratonaCIn

- Memória: O(SUM OF STRINGS * ALPHABET)
- Tempo: Inserção O(n), Queries O(n), Remoção O(n)
  - n sendo o tamanho da string

Rabin Karp - Hash

- Objetivo: transformar uma string para base N-ária
- Comparação -> O(1)
- “134379” = (‘1’*11^5) + (‘3’*11^4) + (‘4’*11^3) + (‘3’*11^2) + (‘7’*11^1) + (‘9’*11^0)
- “abcdex” = (‘a’*257^5) + (‘b’*257^4) + (‘c’*257^3) + (‘d’*257^2) + (‘e’*257^1) + (‘x’*257^0)
- Utiliza aritmética modular para lidar com problemas de overflow

Rabin Karp - Hash


```
void build(){
    pot[0] = 1;
    hash[0] = s[0];
    for(int i = 1; i < n; ++i){
        pot[i] = (pot[i-1] * base) % MOD;
        hash[i] = ((hash[i-1] * base) + s[i]) % MOD;
    }
}
```

Rabin Karp - Hash


```
ll getkey(int l, int r){ // (l, r)
  int res = \ hash[r];
    if(l > 0) res = ((res - pot[r - l + 1] * hash[l-1]) % MOD + MOD) % MOD;
    return res;
}
```