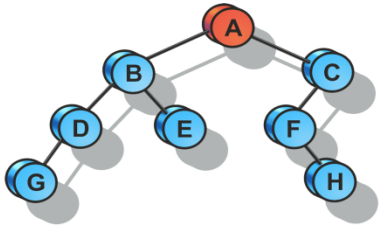
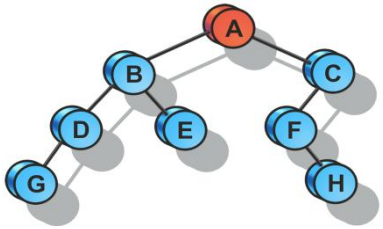
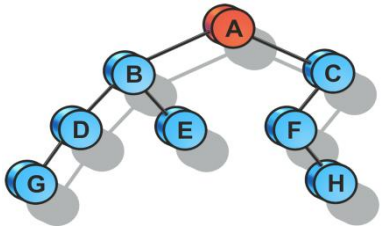


Поиск подстрок (часть 5). Алгоритм Бойера — Мура



Время работы алгоритма Бойера - Мура

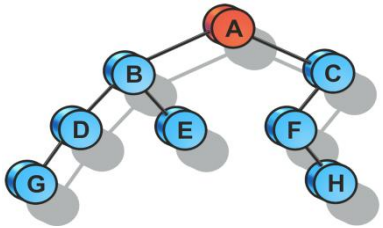
На «плохих» данных: O(NM)
На «хороших» данных: O(N / M)



Идея 1. Эвристика плохого символа

КОРАБЛИ ЛАВИРОВАЛИ

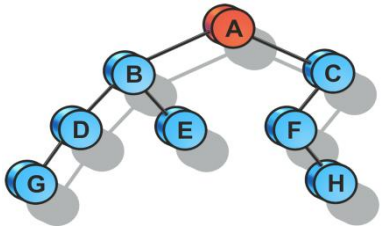
РОВ



Идея 1. Эвристика плохого символа

КОРАБЛИ ЛАВИРОВАЛИ

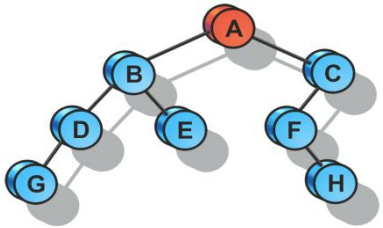
РОВ



Идея 1. Эвристика плохого символа

КОРАБЛИ ЛАВИРОВАЛИ

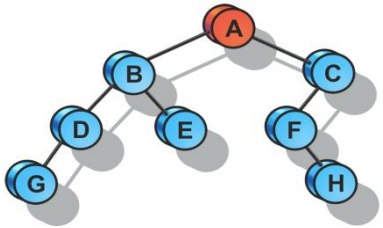
РОВ



Идея 1. Эвристика плохого символа

КОРАБЛИ ЛАВИРОВАЛИ

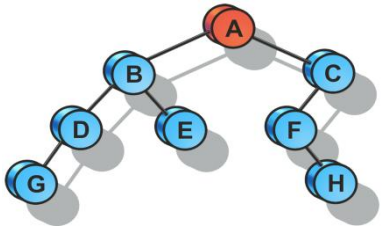
РОВ



Идея 1. Эвристика плохого символа

КОРАБЛИ ЛАВИРОВАЛИ

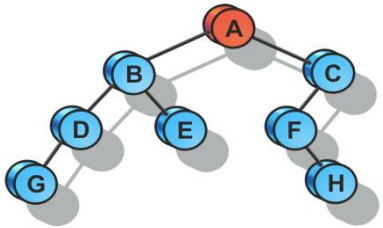
РОВ



Идея 1. Эвристика плохого символа

КОРАБЛИ ЛАВИРОВАЛИ

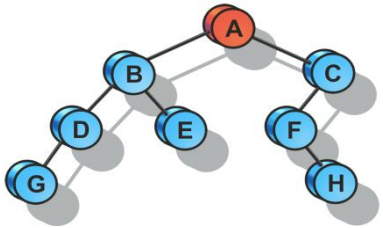
РОВ



Идея 1. Эвристика плохого символа

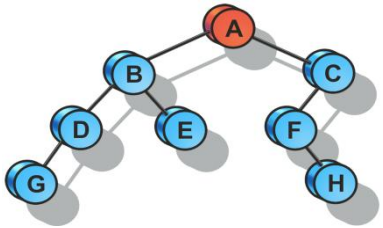
КОРАБЛИ ЛАВИРОВАЛИ

РОВ



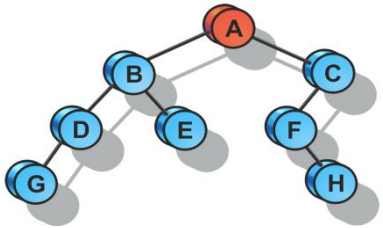
Код. Построение таблицы плохих символов

```
pos = [-1] * SYMBOLS
for i = 0..m - 1:
  pos[T[i]] = i
```



Код. Поиск подстроки

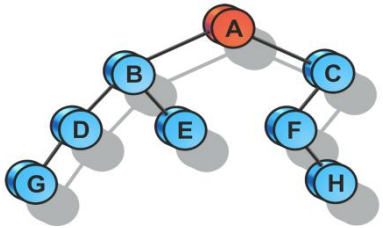
```
i = 0
while i <= n - m:
  j = m - 1
  while S[i + j] == T[j]:
    j--
    if j < 0:
      return i
  i += max(j - pos[S[i + j]], 1)
return -1
```



Идея 2. Эвристика хорошего суффикса

ЛЕДОКОЛ КОЛОЛ ЛЕД

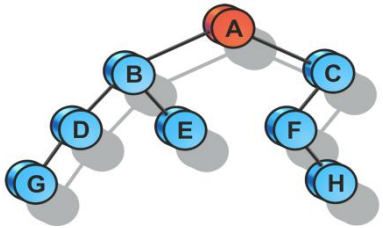
КОЛОКОЛ



Идея 2. Эвристика хорошего суффикса

ЛЕДОКОЛ КОЛОЛ ЛЕД

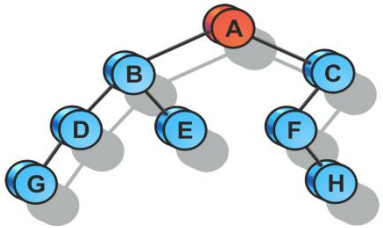
КОЛОКОЛ



Идея 2. Эвристика хорошего суффикса

ЛЕДОКОЛ КОЛОЛ ЛЕД

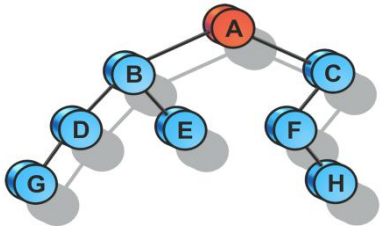
КОЛОКОЛ



Идея 2. Эвристика хорошего суффикса

ЛЕДОКОЛ КОЛОЛ ЛЕД

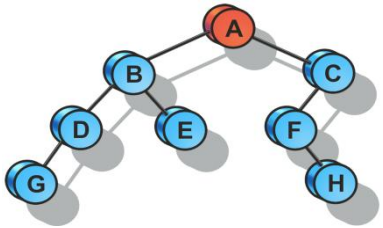
КОЛОКОЛ



Идея 2. Эвристика хорошего суффикса

ЛЕДОКОЛ КОЛОЛ ЛЕД

КОЛОКОЛ



Идея 2. Эвристика хорошего суффикса

ЛЕДОКОЛ КОЛОЛ ЛЕД

КОЛОКОЛ

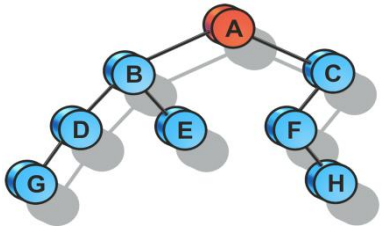


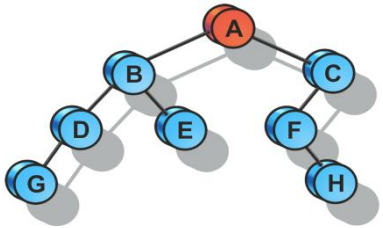
Таблица хороших суффиксов

К О Л О К О Л

– – – – 0 1 2

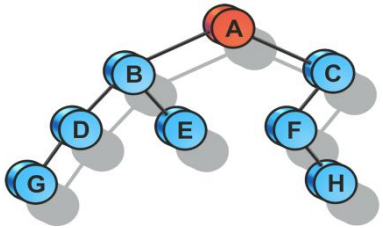
А Б Р А К А Д А Б Р А

– – – – – – – 0 1 2 7



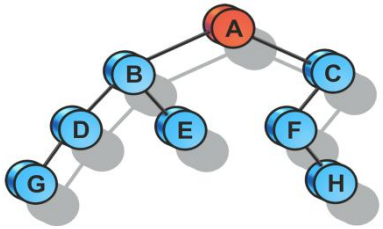
Код. Построение таблицы хороших суффиксов

```
suf = [-1] * m
for x = 0..m - 2
  i = x
  j = m - 1
  while i > 0 && T[i] == T[j]:
    suf[j] = i
    i--
    j--
```



Код. Поиск подстроки

```
i = m - 1
while i < n:
  j = m - 1
  while S[i] == T[j]:
    i--, j--
    if j == 0:
      return i
  i += j + 1 - suf[j + 1]
return -1
```



Совместное использование эвристик

```
i = m - 1
while i < n:
  j = m - 1
  while S[i] == T[j]:
    i--, j--
    if j == 0:
      return i
  i += max(j + 1 - suf[i + 1],
           j - pos[S[i + j]])
return -1
```