# Расстояние Левенштейна

Динамическое программирование – подчас частный случай – рекуррентные вычисления

Рекуррентная формула использует предыдущие шаги вычисления (члены ряда) для следующих

Числа Фибоначчи: 1, 1, 2, 3, 5, 8, 13, 24 …

Fi = {1, i=1 || 1, i=2 || fi-2+f-1, i>2}

Можем представить эти вычисления как шаблон в три клетки [1,1,2];[1,2,3]

Можно вычислять циклически или рекурсивно

Расстояние может быть:

* Между двумя числами
* В N-мерном пространстве
* Манхэттенское (гор. кварталов)
* Между строками

Расстояние Левенштейна (это базовый вид редакторского расстояния) – это минимальное количество редакций необходимое для превращения одной строки в другую

Редакторские операции:

* I (insert) - вставка
* D (delete) - удаление
* R (replace) – замена

У этих трёх операций Штраф 1

Добавим M (Match) – совпадение без штрафа

Взаимное выравнивание строк – это проблема

р а з в л е ч е н и я

у в л е ч е н и е

В данном случае выравнивание по левому краю нецелесообразно – 10 операций (RRRRRMRRRDD)

р а з в л е ч е н и я

у в л е ч е н и е

DDRMMMMMMMR – 4 операции

Используя рекуррентную формулу

D(s1[1...length(s1)], s2[1…length(s2)]) – расстояние между первой и второй строкой??? (Полные подстроки)

D(s1[1…i], s2[1…j])=

- Эти три строки – тривиальный случай

|  |  |
| --- | --- |
| Диаг. +0/1 | Вниз +1 |
| Вправо +1 | **.** |

Тривиальные случай:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 0(ноль) | С | К | А | Т |
| 0(ноль) | **0** | **1** | **2** | **3** | **4** |
| К | **1** |  |  |  |  |
| О | **2** |  |  |  |  |
| Т | **3** |  |  |  |  |

Дальнейший расчёт по формуле:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 0(ноль) | С | К | А | Т |
| 0(ноль) | 0 | 1 | 2 | 3 | 4 |
| К | 1 | **1** | **1** | **2** | **3** |
| О | 2 | **2** | **2** | **2** | **3** |
| Т | 3 | **3** | **3** | **3** | **2** |

D(к,с)=min(D(к,0)+1, D(0,с)+1,…)

D(0,0)+1=

При D(к,ск)= -> 1+0

D(к,ск)=min(D(s1[1…i],s2[1…j]) = ->1+1, 2+1, 1+0 = 1

*Применение 1:*

Компьютерная лингвистика:

* Авто замена
* Поисковая строка
* Yandex: “возможно вы имел ввиду…”

*Применение 2:*

Биоинформатика (кодируем молекулы буквами)

Если необходимо определить последовательность операций, которые привели к ответу, представим матрицу как карту высот. Надо спуститься на санках, фигурально выражаясь, из клетки с ответом в 0 в левом верхнем углу, проверяю легальность перемещения (могли быть совершённый ход при прямом проходе при заполнении матрицы) (налево и вверх мы можем ехать только по уменьшению)

2->2 диагональ => M совпд.

2->1 диагональ => R замена

1->1 диагональ => M совпд.

1->0 влево по горизонтали => I вставка

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Память программы | Адрес  Возврата | Состояние регимстров | Перем.  result | Память под аргументы | Локальные переменные |

Последние 5 столбцов – память метода

Рекурсивная реализация расстояния Левенштейна без кэширования промежуточных результатов будет работать долго при длинных строках

D(«кот», «скат»)

* D(«ко», «скат»)
  + D(«к», «скат»)
    - D(«к», «ска»)
  + D(«ко», «ска»
* D(«ко», «ска»)
  + D(«ко», «ска») повт.
  + D(«к», «ск»)
    - Много повторов
* D(«кот», «ска»
  + D(«ко», «ска») повт.
  + D(«ко», «ск»)
  + D(«ко», «ск»)
    - Много повторов

Менее трудоёмким может быть код, использующий рекурсию и кэш.

у н и |в е| р с и т е т

у н и |е в| р с и т е т

X (xchange) – перестановка

Или T – транспонирование

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **у** | **н** | **и** | **е** | **в** |
| **у** |  |  |  |  |  |
| **н** |  |  |  |  |  |
| **и** |  |  | ***0*** |  |  |
| **в** |  |  |  | ***1*** | ***1*** |
| **е** |  |  |  | ***1*** | ***?*** |