Algoritmos em Strings

R. Rossetti, A. P. Rocha, J. Pascoal Faria

FEUP, MIEIC, CAL, 2013/2014

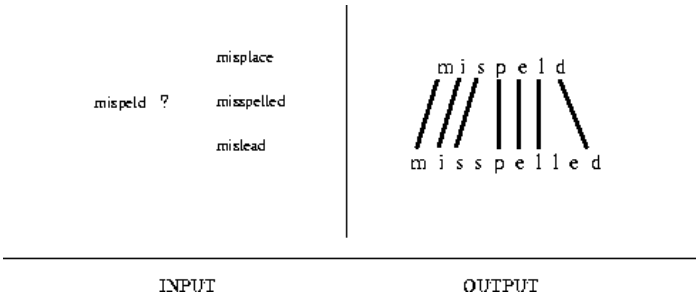
Índice

- Pesquisa exata (*string matching*)
- Pesquisa aproximada (*approximate string matching*)
- Outros problemas

Pesquisa aproximada

(*approximate string matching*)

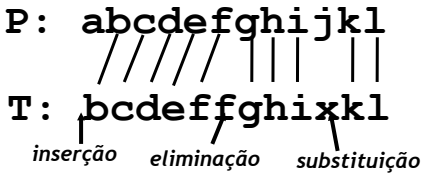
Problema



Input description: A text string *T* and a pattern string *P*. An edit cost bound *k*.
Problem description: Can we transform *T* to *P* using at most *k* insertions, deletions, and substitutions?
(Ou: qual é o grau de semelhança entre *P* e *T* ?)

Distância de edição entre duas strings

- A *distância de edição* entre *P* (*pattern string*) e *T* (*text string*) é o menor número de alterações necessárias para transformar *T* em *P*, em que as alterações podem ser:
  - substituir um carácter por outro
  - inserir um carácter
  - eliminar um carácter

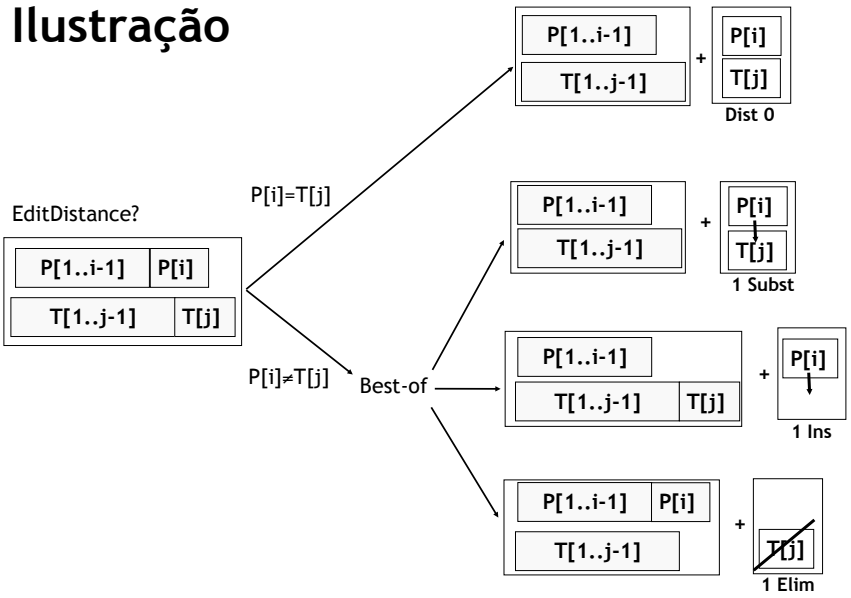


EditDistance(P,T)=3

Formulação recursiva

- $D[i,j] = EditDistance(P[1..i], T[1..j]),\ 0 \le i \le |P|,\ 0 \le j \le |T|$
- Caso recursivo ($i>0$ e $j>0$):
  - Se $P[i]=T[j]$, então $D[i, j] = D[i-1, j-1]$
  - Senão, escolhe-se a operação de edição que sai mais barata; isto é, $D[i,j]$ é o mínimo de:
    - $1 + D[i-1, j-1]$ (substituição de $T[j]$ por $P[i]$)
    - $1 + D[i-1, j]$ (inserção de $P[i]$ a seguir a $T[j]$)
    - $1 + D[i, j-1]$ (eliminação de $T[j]$)
- Condições fronteira:
  - $D[0, j] = j,\quad D[i,0] = i$ (porquê?)

Ilustração



Matriz de programação dinâmica

D[i,j]

T

P

			b	c	d	e	f	f	g	h	i	x	k	l
	0	1	2	3	4	5	6	7	8	9	10	11	12
a	1	1	2	3	4	5	6	7	8	9	10	11	12
b	2	1	2	3	4	5	6	7	8	9	10	11	12
c	3	2	1	2	3	4	5	6	7	8	9	10	11
d	4	3	2	1	2	3	4	5	6	7	8	9	10
e	5	4	3	2	1	2	3	4	5	6	7	8	9
f	6	5	4	3	2	1	2	3	4	5	6	7	8
g	7	6	5	4	3	2	2	2	3	4	5	6	7
h	8	7	6	5	4	3	3	3	2	3	4	5	6
i	9	8	7	6	5	4	4	4	3	2	3	4	5
j	10	9	8	7	6	5	5	5	4	3	3	4	5
k	11	10	9	8	7	6	6	6	5	4	4	3	4
l	12	11	10	9	8	7	7	7	6	5	5	4	3

Pseudo-código

Tempo e espaço: O(|P|.|T|)

```
EditDistance(P,T) {
  // inicialização
  for i = 0 to |P| do D[i,0] = i
  for j = 0 to |T| do D[0,j] = j
  // recorrência
  for i = 1 to |P| do
    for j = 1 to |T| do
       if P[i] == T[j] then D[i,j] = D[i-1,j-1]
       else D[i,j] = 1 + min(D[i-1,j-1],
                             D[i-1,j],
                             D[i,j-1])
  // finalização
  return D[|P|, |T|]
}
```


Optimização de espaço

Espaço: O(|T|)

```
EditDistance(P,T) {
  // inicialização
  for j = 0 to |T| do D[j] = j  // D[0,j]
  // recorrência
  for i = 1 to |P| do
    old = D[0] // guarda D[i-1,0]
    D[0] = i   // inicializa D[i, 0]
    for j = 1 to |T| do
       if P[i] == T[j] then new = old
       else new = 1 + min(old,
                          D[j],
                          D[j-1])
       old = D[j]
       D[j] = new
  // finalização
  return D[|T|]
}
```

Tem D[i-1,j-1]

Ainda tem valor anterior D[i-1,j]

Já tem valor da iteração corrente, i.e., D[i, j-1]

Outros problemas

- Sub-sequência comum mais comprida (*longest common subsequence*)
  - Formada por caracteres não necessariamente consecutivos
  - ABD ? ABCDEF (delete)
- Substring comum mais comprida (*longest common substring*)
  - Formada por caracteres consecutivos
  - ABAB (BAB) BABA (BA) ABBA -> {AB, BA} (tamanho 2)
- Compressão de texto com códigos de Huffman
- Criptografia

Referências e mais informação

- "Introduction to Algorithms", Second Edition, Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein, The MIT Press, 2001
  - Fonte consultada para o "matching" exato
- "The Algorithm Design Manual", Steven S. Skiena, Springer-Verlag, 1998
  - Fonte consultada para o "matching" aproximado
  - Discute como se usa o cálculo da distância de edição para encontrar num texto *T* a substring que faz o melhor "match" com um padrão de pesquisa *P*
- *Com base em slides de J. P. Faria*