Processamento de Cadeias de Caracteres

prof. Fábio Luiz Usberti

MC621 - Programação Avançada II

Instituto de Computação - UNICAMP

Sumário

Introdução

- Serão discutidos problemas de processamento de strings que possuem soluções eficientes por programação dinâmica.
- Nessa categoria, dois problemas clássicos que devem estar no repertório de problemas dos programadores competitivos são:

1. Problema de alinhamento de strings.
2. Problema da máxima subsequência comum.

Alinhamento de strings

- O problema de alinhamento de strings pode ser aplicado para obter a distância de edição ou distância de Levenshtein entre duas strings.
- Considera-se como distância da edição de duas strings o número mínimo de alterações (remoções ou substituições de caracteres) que devem ser aplicadas na primeira string para se obter a segunda string.
- A distância de edição é aplicada, por exemplo, em corretores automáticos de texto.
- Se um usuário digita uma palavra incorretamente, um editor de texto poderá tentar reconhecer e sugerir uma palavra de um dicionário com a menor distância de edição com a palavra correta.

Alinhamento de strings

- O problema de alinhamento de strings é definido da seguinte forma: sejam P e Q duas strings de tamanhos m e n: escolha uma subsequência P' (da string P) e uma subsequência Q' (da string Q), tal que P' e Q' possuem o mesmo tamanho, de modo a maximizar uma função de pontuação de alinhamento.
- Como exemplo, considere a seguinte função de pontuação do alinhamento, $score(P[i], Q[i])$, para dois caracteres $P[i]$ e $Q[i]$:

1. Match: Caracteres $P[i]$ e $Q[i]$ são iguais ($score(P[i], Q[i]) = +2$).
2. Mismatch: Caracteres $P[i]$ e $Q[i]$ são diferentes ($score(P[i], Q[i]) = -1$).
3. Gap: Remova o caractere $Q[i]$ ($score(\_, Q[i]) = -1$).
4. Gap: Remova o caractere $P[i]$ ($score(P[i], \_) = -1$).

- Obs.: A função *score* pode ser alterada para ajustar a pontuação dependendo do problema que se deseja resolver. Por exemplo, como alterar a função *score* para obter a distância de edição entre duas strings?

Alinhamento de strings

Exemplo:

- No exemplo abaixo, o caractere _ em uma string denota a remoção do caractere correspondente da outra string.

```
P  =  TCTTCTTAATGCAGTCCTTTTAT  ->   P' = TCTTCTTAATGCAGTC--CTTTTAT
Q  =  TCTTATTCAGTCATCTCTT      ->   Q' = TCT-CTT-ATTCAGTCATCTCTT--
-------------------------------------------------------------------
                              pontuação = 222-222-22-22222--22-22-- = 26
```

Alinhamento de strings

- A solução do problema de alinhamento de strings por força-bruta, ou seja, tentar todos os possíveis alinhamentos $\Omega(2^{\min\{m,n\}})$, teria como resultado TLE mesmo para instâncias de tamanho médio.
- Uma técnica de programação dinâmica eficiente para esse problema é conhecida por algoritmo de Needleman-Wunsch's.

Alinhamento de strings

- Considere duas strings $P[1..n]$ e $Q[1..m]$ e defina $V(i,j)$ como a pontuação de alinhamento ótima (máxima) para os prefixos $P[1..i]$ e $Q[1..j]$.
- Subestrutura ótima: o alinhamento dos prefixos $P[1..i]$ e $Q[1..j]$ permite três possibilidades para seus últimos caracteres $P[i]$ e $Q[j]$:

1. Caracteres $P[i]$ e $Q[j]$ são alinhados.
2. Caractere $P[i]$ é removido.
3. Caractere $Q[j]$ é removido.

Alinhamento de strings

- Um alinhamento ótimo dos prefixos $P[1..i]$ e $Q[1..j]$ reside necessariamente em (pelo menos) uma das três possibilidades mencionadas para o par de caracteres $P[i]$ e $Q[j]$. Isso pode ser formalizado através da seguinte função recursiva:

$$V(i,j) = \max\left[V_1(i,j), V_2(i,j), V_3(i,j)\right]$$

Onde:

$V_1(i,j) = V(i-1,j-1) + score(P[i], Q[j])$ (caracteres $P[i]$ e $Q[j]$ são alinhados)
$V_2(i,j) = V(i-1,j) + score(P[i], \_)$ (caractere $P[i]$ é removido)
$V_3(i,j) = V(i,j-1) + score(\_, Q[j])$ (caractere $Q[j]$ é removido)

- Casos base:

$$V(0,0) = 0$$
$$V(i,0) = i \times score(P[i], \_)$$
$$V(0,j) = j \times score(\_, Q[j])$$

- É possível verificar na função recursiva acima a ocorrência de sobreposição de subproblemas.

Alinhamento de strings

	_	A	G	C	A	T	G	C
_	0	-1	-2	-3	-4	-5	-6	-7
A	-1							
C	-2		Base Cases					
A	-3							
A	-4							
T	-5							
C	-6							
C	-7							

	_	A	G	C	A	T	G	C
_	0	-1	-2	-3	-4	-5	-6	-7
A	-1	2	1	0	-1	-2	-3	-4
C	-2	1	1	3				
A	-3							
A	-4							
T	-5							
C	-6							
C	-7							

	_	A	G	C	A	T	G	C
_	0	-1	-2	-3	-4	-5	-6	-7
A	-1	2	1	0	-1	-2	-3	-4
C	-2	1	1	3	2	1	0	-1
A	-3	0	0	2	5	4	3	2
A	-4	-1	-1	1	4	4	3	2
T	-5	-2	-2	0	3	6	5	4
C	-6	-3	-3	0	2	5	5	7
C	-7	-4	-4	-1	1	4	4	7

Exemplo:

- Para o preenchimento da tabela de programação dinâmica, primeiramente os casos base são preenchidos.
- Em seguida, as células podem ser preenchidas linha a linha, de cima para baixo, da esquerda para direita.
- Note que para o preenchimento de uma célula é necessário consultar três células adjacentes: superior, esquerda e superior-esquerda.

Alinhamento de strings

	_	A	G	C	A	T	G	C
_	0	-1	-2	-3	-4	-5	-6	-7
A	-1							
C	-2		Base Cases					
A	-3							
A	-4							
T	-5							
C	-6							
C	-7							

	_	A	G	C	A	T	G	C
_	0	-1	-2	-3	-4	-5	-6	-7
A	-1	2	1	0	-1	-2	-3	-4
C	-2	1	1	3				
A	-3							
A	-4							
T	-5							
C	-6							
C	-7							

	_	A	G	C	A	T	G	C
_	0	-1	-2	-3	-4	-5	-6	-7
A	-1	2	1	0	-1	-2	-3	-4
C	-2	1	1	3	2	1	0	-1
A	-3	0	0	2	5	4	3	2
A	-4	-1	-1	1	4	4	3	2
T	-5	-2	-2	0	3	6	5	4
C	-6	-3	-3	0	2	5	5	7
C	-7	-4	-4	-1	1	4	4	7

Exemplo:

- Uma vez preenchida a tabela de programação dinâmica, o valor ótimo estará na célula do canto inferior direito.
- Para construir o alinhamento ótima, basta construir a trajetória a partir da célula do canto inferior direito até a célula do canto superior esquerdo.
- Um passo na vertical indica uma remoção de caractere da string A; um passo na horizontal indica uma remoção de caractere da string B; um passo na diagonal indica que os caracteres A e B estão alinhados.

Alinhamento de strings

	_	A	G	C	A	T	G	C
_	0	-1	-2	-3	-4	-5	-6	-7
A	-1							
C	-2		Base Cases					
A	-3							
A	-4							
T	-5							
C	-6							
C	-7							

	_	A	G	C	A	T	G	C
_	0	-1	-2	-3	-4	-5	-6	-7
A	-1	2	1	0	-1	-2	-3	-4
C	-2	1	1	3				
A	-3							
A	-4							
T	-5							
C	-6							
C	-7							

	_	A	G	C	A	T	G	C
_	0	-1	-2	-3	-4	-5	-6	-7
A	-1	2	1	0	-1	-2	-3	-4
C	-2	1	1	3	2	1	0	-1
A	-3	0	0	2	5	4	3	2
A	-4	-1	-1	1	4	4	3	2
T	-5	-2	-2	0	3	6	5	4
C	-6	-3	-3	0	2	5	5	7
C	-7	-4	-4	-1	1	4	4	7

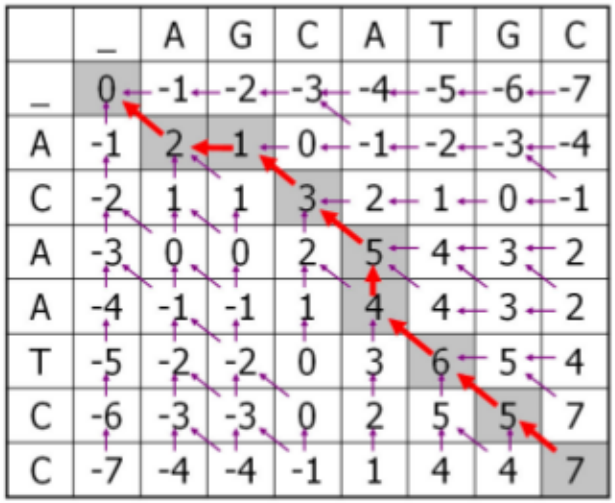
Solução ótima:

```
A   =   ACAATCC   ->   A_CAATCC
B   =   AGCATGC   ->   AGCA_TGC
-------------------------------
        pontuação = 2-22-2-2 = 7
```

Alinhamento de strings

	_	A	G	C	A	T	G	C
_	0	-1	-2	-3	-4	-5	-6	-7
A	-1							
C	-2		Base Cases					
A	-3							
A	-4							
T	-5							
C	-6							
C	-7							

	_	A	G	C	A	T	G	C
_	0	-1	-2	-3	-4	-5	-6	-7
A	-1	2	1	0	-1	-2	-3	-4
C	-2	1	1	3				
A	-3							
A	-4							
T	-5							
C	-6							
C	-7							



Complexidade:

- Para preencher cada célula da tabela de programação dinâmica basta avaliar três células vizinhas $\Theta(1)$. Portanto, a complexidade do algoritmo está limitada por $\Theta(mn)$.

Alinhamento de strings

```
int main() { // Needleman Wunsnch's algorithm
  char P[20] = "ACAATCC", Q[20] = "AGCATGC";
  int n = (int)strlen(P), m = (int)strlen(Q);
  int i, j, table[20][20];
  memset(table, 0, sizeof table);
  // insert/delete = -1 point
  for (i = 1; i <= n; i++) table[i][0] = i * -1;
  for (j = 1; j <= m; j++) table[0][j] = j * -1;

  for (i = 1; i <= n; i++)
    for (j = 1; j <= m; j++) {
      // match = 2 points, mismatch = -1 point
      table[i][j] = table[i - 1][j - 1] + (P[i - 1] == Q[j - 1] ? 2 : -1);
      // gap = -1 point
      table[i][j] = max(table[i][j], table[i - 1][j] - 1); // delete character from P
      table[i][j] = max(table[i][j], table[i][j - 1] - 1); // delete character from Q
    }
  printf("DP table:\n");
  for (i = 0; i <= n; i++) {
    for (j = 0; j <= m; j++)
      printf("%3d", table[i][j]);
    printf("\n");
  }
  printf("Maximum Alignment Score: %d\n", table[n][m]);
  return 0;
}
```

Máxima subsequência comum

- A máxima subsequência comum (LCS – "Longest Common Subsequence") de duas strings P e Q consiste em encontrar a maior subsequência de caracteres em P que seja comum a Q.
- Exemplo: $P = ACAATCC$ e $Q = AGCATGC$ apresentam um LCS de tamanho 5: $LCS(P, Q) = ACATC$.
- O problema LCS pode ser reduzido para o problema de alinhamento de strings redefinindo os custos de alinhamento $score(P[i], Q[i])$ da seguinte forma:

1. Caracteres $P[i]$ e $Q[i]$ formam par ($score(P[i], Q[i]) = +1$).
2. Caracteres $P[i]$ e $Q[i]$ não formam par ($score(P[i], Q[i]) = 0$).
3. Remova o caractere $Q[i]$ ($score(\_, Q[i]) = 0$).
4. Remova o caractere $P[i]$ ($score(P[i], \_) = 0$);

Palíndromo máximo

- Um palíndromo consiste em uma string que quando revertida resulta na mesma string.
- Muitas problemas envolvendo palíndromos podem ser resolvidos por técnicas de programação dinâmica.
- UVa 11151 - Longest Palindrome: Dada uma string com até $n = 1000$ caracteres, determine o comprimento do maior palíndromo que pode ser obtido após a remoção de 0 ou mais caracteres.
- Exemplos:

1. $ADAM \rightarrow ADA$
2. $MADAM \rightarrow MADAM$
3. $NEVERODDOREVENING \rightarrow NEVERODDOREVEN$
4. $RACEF1CARFAST \rightarrow RACECAR$

Palíndromo máximo

- Seja $len(l, r)$ o comprimento do maior palíndromo que pode ser gerado a partir da string $A[l..r]$.
- Casos base:

$$len(l, r) = \begin{cases} 0 & (l > r) \\ 1 & (l = r) \end{cases}$$

- Função recursiva de otimalidade:

$$len(l, r) = \begin{cases} 2 + len(l + 1, r - 1) & (A[l] = A[r]) \\ \max\left[len(l, r - 1), len(l + 1, r)\right] & (A[l] \neq A[r]) \end{cases}$$

- Complexidade: $O(n^2)$
- Obs: é possível reduzir esse problema para o problema de alinhamento de strings?

Referências

1. S. Halim e F. Halim. Competitive Programming 2, Second Edition Lulu (www.lulu.com), 2011. (IMECC – 005.1 H139c)
2. S. S. Skiena, M. A. Revilla. Programming Challenges: The Programming Contest Training Manual, Springer, 2003.
3. T.H. Cormen, C.E. Leiserson, R.L.Rivest e C. Stein. Introduction to Algorithms. 2nd Edition, McGraw-Hill, 2001. (no. chamada IMECC – 005.133 Ar64j 3.ed.)
4. U. Manber. Introduction to Algorithms: A Creative Approach. Addison-Wesley. 1989. (no. chamada IMECC – 005.133 Ec53t 2.ed.)