Trabalho de Tópicos em Ciência da Computação: Programação Dinâmica - Análise de Problemas

Domingo, 04 de Setembro, 2016

Leonardo Mauro

Leonardo Mauro P. Moraes

Conteúdo

Problema 1

Distância de Edição (Edit Distance).

Problema

Dado duas *strings* (duas sequências de caracteres), compare-as de forma a retornar o número mínimo de operações necessárias para transformar uma *string* na outra [7]. Entende-se por operações a inserção, deleção ou substituição de um caractere [7, 9].

Complexidade - Solução

Algoritmo de Distância Levenshtein, que utiliza uma matriz [x + 1][y + 1], se bottom-up [7].
Complexidade: $O(|x| * |y|)$.

Código [4, 9]

Listing 1: Algoritmo da Distância de Levenshtein.

```
int levenshtein_topdown(string &x, int len_x, string &y, int len_y){
    int custo, custo_min;

    /* Caso base: strings vazias */
    if(len_x == 0) return len_y;
    if(len_y == 0) return len_x;

    /* Testa se o ultimo caractere das strings combinam */
    if(x[len_x-1] == y[len_y-1]) custo = 0;
    else custo = 1;

    /* Retorna o minimo da deleção do char x, char y, e dos dois */
    custo_min = min(levenshtein(x, len_x - 1, y, len_y    ) + 1,
                    levenshtein(x, len_x    , y, len_y - 1) + 1);
    custo_min = min(levenshtein(x, len_x - 1, y, len_y - 1) + custo, custo_min);

    return custo_min;
}

typedef struct {
    int val;
    char op;
    /* s - substituição, i - inserção, d - deleção, v - vazio */
} lev_data;

int levenshtein_bottomup(string &x, string &y){
    int custo, mat_min;
    char op;

    /* Matriz de operacoes */
    int x_len = x.size()+1, y_len = y.size()+1;
    vector< vector< lev_data > > mat (x_len, vector< lev_data > (y_len));

    fore(i, 0, x_len){ mat[i][0].val = i; mat[i][0].op = 'v'; }
```

```
35      fore(j, 0, y_len){ mat[0][j].val = j; mat[0][j].op = 'v'; }
36
37      fore(i, 1, x_len){
38          fore(j, 1, y_len){
39              /* Determina custo */
40              if(x[i-1] == y[j-1]) custo = 0;
41              else custo = 1;
42              /* Função minimo */
43              mat_min = mat[i-1][j].val + 1; op = 'i';
44              if(mat_min > mat[i][j-1].val + 1){
45                  mat_min = mat[i][j-1].val + 1; op = 'd';
46              }
47              if(mat_min > mat[i-1][j-1].val + custo){
48                  mat_min = mat[i-1][j-1].val + custo; op = 's';
49              }
50              /* Preenche matriz */
51              mat[i][j].val = mat_min;
52              mat[i][j].op = op;
53          }
54      }
55
56      return mat[x.size()][y.size()].val;
57  }
```

Problema 2

Soma de Subconjunto (Subset Sum).

Problema

Dado um conjunto de n números $vet[i]$ que a soma total é igual a M, e para algum $K \leq M$, se existe um subconjunto de números tais que a soma desse subconjunto dá exatamente K [6].

Complexidade - Solução

Algoritmo parte do pressuposto de que não há valores negativos no vetor de inteiros. Então dado o vetor $vet[]$ e o K, retorna $true$ ou $false$ para essa solução [6, 1].
Complexidade: $O(|K| * |n|)$.

Código [2]

Listing 2: Algoritmo Bottom Up Subset Sum.

```
1   bool subset_sum(vector<int> &vet, int sum){
2       /* Os valores subset[i][j] representa subset do
3          vet[0..j-1] que a soma é equivalente a i */
4       int n = vet.size();
5       bool subset[sum+1][n+1];
```

```
6
7       /* Caso base: nenhum elemento */
8       for(int j = 0; j < n+1; j++) subset[0][j] = true;
9
10      /* De mais valores false */
11      for(int i = 1; i < sum+1; i++) subset[i][0] = false;
12
13      /* Preenchimento bottomup */
14      for(int i = 1; i < sum+1; i++){
15          for(int j = 1; j < n+1; j++){
16              subset[i][j] = subset[i][j-1];
17              if (i >= vet[j-1])
18                  subset[i][j] = subset[i][j] || subset[i - vet[j-1]][j-1];
19          }
20      }
21
22      return subset[sum][n];
23  }
```

Problema 3

Tamanho da Maior Subsequência Comum (Longest Common Subsequence).

Problema

Dado duas *strings* encontrar o tamanho da maior subsequência comum entre elas (subsequência mais longa presente em ambas as *strings*) [5, 3].

Complexidade - Solução

Algoritmo recebe as duas *strings*, de tamanho m e n, então retorna o valor da maior subsequência entre elas [3].
Complexidade: $O(|m| * |n|)$.

Código [3]

Listing 3: Algoritmo Bottom Up Longest Common Subsequence.

```
1   int lcs(string &X, string &Y){
2       int m = X.size(), n = Y.size();
3       int L[m+1][n+1];
4
5       /* Construindo a L[m+1][n+1] */
6       for(int i=0; i<m+1; i++){
7           for(int j=0; j<n+1; j++){
8               /* Caso base: nenhum caractere */
9               if(i == 0 || j == 0) L[i][j] = 0;
10              /* Se forem iguais soma 1 com o resultado anterior */
```

```
11              else if(X[i-1] == Y[j-1]) L[i][j] = L[i-1][j-1] + 1;
12              /* Se não pega o máximo do X(i-1), Y(j-1) */
13              else L[i][j] = max(L[i-1][j], L[i][j-1]);
14          }
15      }
16
17      /* Resultado final: L[m][n] */
18      return L[m][n];
19  }
```

Referências

[1] Michael R Garey and David S Johnson. Computers and intractability: a guide to the theory of np-completeness. a series of books in the mathematical sciences. *WH Freeman and Company, New York, NY*, 25(27):141, 1979.

[2] GeeksforGeeks. Dynamic programming — set 25 (subset sum problem). *http://www.geeksforgeeks.org/dynamic-programming-subset-sum-problem/*, 2015. Online; Accessado: 28/08/2016.

[3] GeeksforGeeks. Dynamic programming — set 4 (longest common subsequence). *http://www.geeksforgeeks.org/dynamic-programming-set-4-longest-common-subsequence/*, 2015. Online; Accessado: 28/08/2016.

[4] F GONDIM. Algoritmo de comparação de strings para integração de esquemas de dados. *Trabalho de Conclusão de Curso (Graduação)*, 2006.

[5] ICS. Ics 161: Design and analysis of algorithms lecture notes for february 29, 1996. *https://www.ics.uci.edu/ eppstein/161/960229.html*, 1996. Online; Accessado: 28/08/2016.

[6] Marathoncode. Programação dinâmica - soma de subconjunto (subset sum). *http://marathoncode.blogspot.com.br/2013/01/soma-de-subconjunto-subset-sum.html*, 2013. Online; Accessado: 28/08/2016.

[7] Gonzalo Navarro. A guided tour to approximate string matching. *ACM computing surveys (CSUR)*, 33(1):31–88, 2001.

[8] Peter Norvig and Stuart Russell. *Inteligência Artificial, 3ª Edição*, volume 1. Elsevier Brasil, 2014.

[9] U.S. National Institute of Standards and Technology. Levenshtein distance. *www.nist.gov/dads/HTML/Levenshtein.html*, 2013. Online; Accessado: 27/08/2016.