Terceiro teste de Algoritmos e Estruturas de Dados

9 de Dezembro de 2019 **14h10m – 15h00m**

Responda a todas as perguntas no enunciado do teste. Justifique todas as suas respostas.
O teste é composto por 5 grupos de perguntas.

Nome: ____________________

N. Mec.: ________

4.0 $\boxed{\textbf{1:}}$ O algoritmo *merge sort* divide o *array* a ser ordenado ao meio, ordena (recursivamente) cada uma das duas partes, e depois junta-as. A sua complexidade computacional é $\boldsymbol{\Theta(n \log n)}$. Um aluno está convencido que se em vez de se dividir o array em duas partes se se dividir em três partes (todas mais ou menos do mesmo tamanho), então a complexidade computacional desta variante do *merge sort* será ainda mais baixa. Responda às seguintes perguntas:

1.0 **a)** Que estratégia algoritmica usa o *merge sort*?

2.0 **b)** O aluno tem razão? Justifique.

1.0 **c)** Indique uma desvantagem do *merge sort*, quando comparado com o *quicksort*.

Respostas:

(a) O *merge sort* utiliza a estratégia algorítmica de *divide and conquer*. Esta estratégia baseia-se em três etapas distintas:
(1) dividir o problema principal em vários problemas mais pequenos;
(2) resolver os sub-problemas à vez;
(3) compôr as suas soluções de modo a resolver o problema principal.

(b) A complexidade do *merge sort* normal é de O(n*log(n)), enquanto que a complexidade do *3-way merge sort* é de O(n*log3(n)). Se bem que a complexidade do *3-way merge sort* possa parecer menor comparada com a do *merge sort* original, o tempo que ele demora pode, na verdade, tornar-se superior ao original porque o número de comparações a realizar no *3-way merge sort* rapidamente se torna muito mais substancial do que o número de comparações a fazer no original. O aluno tem razão apenas para *arrays* relativamente pequenos.

(c) O *merge sort* requer mais espaço de memória do que o *quicksort*.

O *master theorem* afirma que se $\boldsymbol{T(n) = aT(n/b) + f(n)}$ então

- se $\boldsymbol{f(n) = O(n^{\log_b a - \epsilon})}$ para um $\boldsymbol{\epsilon > 0}$, então $\boldsymbol{T(n) = \Theta(n^{\log_b a})}$,
- se $\boldsymbol{f(n) = \Theta(n^{\log_b a})}$, então $\boldsymbol{T(n) = O(n^{\log_b a} \log n)}$,
- se $\boldsymbol{f(n) = \Omega(n^{\log_b a + \epsilon})}$ para um $\boldsymbol{\epsilon > 0}$ e se $\boldsymbol{af(\frac{n}{b}) \leq cf(n)}$ para $\boldsymbol{c < 1}$ e $\boldsymbol{n}$ suficientemente grande, então $\boldsymbol{T(n) = \Theta(f(n))}$.

6.0 **2:** Num tabuleiro de xadrez, pretende-se ir do canto inferior esquerdo $(\mathbf{0},\mathbf{0})$ para o canto superior direito $(\mathbf{7},\mathbf{7})$ fazendo movimentos apenas para a direita e para cima. Quando se está em $(\boldsymbol{x},\boldsymbol{y})$, o custo de ir para a direita é dado por $\boldsymbol{R_{xy}}$ e o custo de ir para cima é dado por $\boldsymbol{U_{xy}}$. O custo total é a soma dos custos dos **14** movimentos efetuados (**7** para a direita e **7** para cima). Sabe-se que $\mathbf{4} \leq \boldsymbol{R_{xy}} \leq \mathbf{10}$ e que $\mathbf{6} \leq \boldsymbol{U_{xy}} \leq \mathbf{20}$. No programa que foi usado para calcular o **custo mínimo** para ir de $(\mathbf{0},\mathbf{0})$ até $(\mathbf{7},\mathbf{7})$, os valores de $\boldsymbol{R_{xy}}$ e $\boldsymbol{U_{xy}}$ estão guardados nas matrizes

```
int R[7][8], U[8][7]; // initialized elsewhere
```

Responda às seguintes perguntas:

3.0 **a)** Complete o seguinte código, que resolve o problema usando a técnica *branch-and-bound.*

```
void go_to(int x,int y,int partial_cost,int *min_cost) {
  if(partial_cost + [4] * (7 - x) + 6 * ([7-y]) [<=] *min_cost) return;
  if(x < 7) go_to(x + 1,y,partial_cost + [R(x+1)y - Rxy],min_cost);
  if(y < 7) go_to(x,y + 1,partial_cost + [Ux(y+1) - Uxy],min_cost);
  if(x == 7 && y == 7) *min_cost = partial_cost;
}
int compute_min_cost_bb(void) {
  int min_cost = [4*7+6*7]; go_to(0,0,[0],&min_cost); return min_cost; }
```

3.0 **b)** Pretende-se também resolver este problema usando programação dinâmica. Para isso, o custo mínimo para ir de $(\mathbf{0},\mathbf{0})$ até $(\boldsymbol{x},\boldsymbol{y})$ é guardado na matriz

```
int C[8][8];
```

Complete o seguinte código. (No fim, estamos interessados no valor de C[7][7], mas, para o calcular, dá jeito conhecer os outros valores.)

```
int update_C(int x,int y,int C[8][8]) {
  if(x [<] 0 || y < 0) return 1000000000;
  if(C[x][y] [<] 0)          → C[x][y] = -1
                               (not yet known)
  {                                              custo de andar de (x-1,y)
                                                 para (x,y)
    int Cx = update_C(x - 1,y,C) + [Rxy - Rx-1y];
    int Cy = update_C(x,y - 1,C) + [Uxy - Uxy-1]; → custo de (x,y-1) para
                                                    (x,y)
    C[x][y] = (Cx < Cy) ? Cx : Cy;
  }
  return C[x][y]; }
int compute_min_cost_dp(void) {
  int C[8][8]; // -1 means not yet known
  for(int x = 0;x < 8;x++)                       not yet known
    for(int y = 0;y < [8];y++)
      C[x][y] = (x == 0 && y == 0) ? 0 : [-1];
  update_C([x],[y],C);
  return C[7][7]; }
```