Aula Prática 3

ASA 2024/2025

Somatórios

- $\sum_{k=1}^{n} k = \frac{n(n+1)}{2}$
- $\sum_{k=1}^{n}(a_k - a_{k-1}) = a_n - a_0$
- $\sum_{k=1}^{n}(a_k - a_{k+1}) = a_1 - a_{n+1}$
- $\sum_{k=0}^{\infty} x^k = \frac{1}{1-x}$, se $|x| < 1$
- $\sum_{k=1}^{n} \frac{1}{k} > \int_1^{n+1} \frac{1}{x} dx = log\,(n+1)$
- $\sum_{k=0}^{\infty} k\,x^k = \frac{1}{(1-x)^2}$, se $|x| < 1$
- $\sum_{k=1}^{n} k^2 = \frac{n(n+1)(2n+1)}{6}$
- $\sum_{k=0}^{n} x^k = \frac{1-x^{n+1}}{1-x}$

Teorema Mestre

Sejam $a \geq 1, b > 1$ constantes e $f(n)$ uma função.
Para $T(n)$ definido por $T(n) = aT(n/b) + f(n)$:

- Caso 1: $T(n) = \Theta(n^{log_b a})$, se $f(n) = O(n^{log_b a - \epsilon})$ para $\epsilon > 0$
- Caso 2: $T(n) = \Theta(n^{log_b a} log\, n)$, se $f(n) = \Theta(n^{log_b a})$
- Caso 3: $T(n) = \Theta(f(n))$, se $f(n) = \Omega(n^{log_b a + \epsilon})$ para $\epsilon > 0$ e se $af(n/b) \leq cf(n)$ para $c < 1$ e n suficientemente grande

Teorema Mestre (simplificado)

Sejam $a \geq 1, b > 1, d \geq 0$ constantes,
seja $T(n)$ definido por $T(n) = aT(n/b) + O(n^d)$.

$$T(n) = \begin{cases} O(n^{log_b a}) & if\ d < log_b a \\ O(n^d log\, n) & if\ d = log_b a \\ O(n^d) & if\ d > log_b a \end{cases}$$

Q1: Considere a função recursiva:

```
int sumxtoy(int x, int y) {
  int i, s;
  int c = 10;
  if (y - x < c) {
    s = 0;
    for (i = x; i <= y; i++)
      s += i;
    return s;
  } else {
    return sumxtoy(x,(x+y)/2) + sumxtoy((x+y)/2+1,y);
  }
}
```

- Determine o menor majorante assimptótico da função `sumxtoy` medido em função dos parâmetros da função.

Solução: Seja $n = y - x$. Vamos determinar a complexidade da função dada em termos de n. Há dois casos a considerar:

- Se $n < 10$:
$$T(n) = y - x + 1 = n + 1 = O(1)$$

- Se $n \geq 10$:
$$\begin{aligned} T(n) &= T((x+y)/2 - x) + T(y - ((x+y)/2 + 1)) \\ &= T((y-x)/2) + T((y-x)/2 - 1) \\ &\leq T(n/2) + T(n/2) \\ &= 2.T(n/2) \end{aligned}$$

Concluímos que:
$$T(n) = \begin{cases} 2T(n/2) & \text{if } n \geq 10 \\ O(1) & \text{otherwise} \end{cases}$$

Aplicando o Teorema Mestre com parâmetros: $a = 2, b = 2, d = 0$, concluímos que: $T(n) = O(n)$, dado que $log_b\ a = 1 > d$ e portanto $T(n) = O(n^{log_b\ a}) = O(n)$.

Q2 (EE 19/20): Considere a seguinte implementação naif de uma fila de prioridade mínima baseada em listas simplesmente ligadas. Uma fila de prioridade é guardada em memória como uma lista de nós, cada qual associado a uma prioridade `pri` e a um identificador `id`. A implementação é composta pelas funções:

- `Insert(Lst lst, Lst node)` que insere o nó `node` na lista `lst`;
- `Remove(Lst lst, int i)` que remove o nó com identificador `i` da lista `lst`;
- `ExtractQueue(Queue q)` que remove o nó com prioridade mínima da fila de prioridade `q`; e

- DecreaseKey(Queue q, Lst node, int pri) que diminui a prioridade do nó `node` para `pri` na fila de prioridade q.

```
typedef struct Node {
   int id;
   int pri;
   struct Node* next;
} *Lst;

typedef struct QueueNode {
   Lst hd;
} *Queue;

int ExtractQueue(Queue q) {
  if (q->hd == NULL) return -1;

  Lst hd = q->hd;
  q->hd = hd->next;
  return hd->id;
}

Lst Remove(Lst lst, int id) {
   if (lst == NULL) return lst;

   if (lst->id == id) return lst->next;

   Lst prev = lst;
   Lst cur = lst->next;
   while (cur != NULL) {
      if (cur->id == id) {
         prev->next = cur->next;
         break;
      }
      prev = cur;
      cur = cur->next;
   }
   return lst;
}

Lst Insert (Lst lst, Lst node) {
  if (lst == NULL) return node;
  if (node->pri <= lst->pri) {
    node->next = lst;
    return node;
  } else {
          Lst ret = Insert(lst->next, node);
          lst->next = ret;
          return lst;
  }
}
```

```
void DecreaseKey (Queue q, Lst node, int pri) {
  Lst lst = Remove(q->hd, node->id);
  node->pri = pri;
  lst = Insert(lst, node);
  q->hd = lst;
}
```

Determine o menor majorante assimptótico para funções `Insert`, `Remove`, `ExtractQueue` e `DecreaseKey` em função do número de elementos, n, contidos na fila de prioridade ou lista que respectivamente recebem como argumento. Deve indicar para cada uma das funções a equação do tempo que expressa o número instruções executadas em função do tamanho do input (i.e. $T(n) = ...$).

Solução:

- $T_{EQ}(n) = O(1)$, which implies that: `ExtractQueue` $\in O(1)$.
- $T_R(n) = \sum_{i=1}^{n} O(1) = O(n)$, which implies that: `Remove` $\in O(n)$.
- $T_I(n) = O(1) + T_I(n-1) = \sum_{i=1}^{n} O(1) = O(n)$, which implies that: `Insert` $\in O(n)$.
- $T_{DK}(n) = O(1) + T_R(n) + T_I(n-1) = O(1) + O(n) + O(n-1) = O(n)$, which implies that: `DecreaseKey` $\in O(n)$.

Q3: Considere a função `findAllPairsWithSum`:

```
std::list<std::pair<int, int> >
  findAllPairsWithSum(std::vector<int>& arr, int v) {
    std::list<std::pair<int, int> > pairs;
    int size = arr.size();

    for (int i = 0; i < size; ++i) {
        for (int j = i + 1; j < size; ++j) {
            if (arr[i] + arr[j] == v) {
                pairs.push_back(std::make_pair(i, j));
            }
        }
    }

    return pairs;
}
```

- Admitindo que a função `push_back` tem complexidade assimptótica $O(1)$, determine o menor majorante assimptótico da função `findAllPairsWithSum` medido em função dos parâmetros da função.

Solução: Seja n igual ao número de elementos do array e $m = n - 1$. A função do tempo é definida como se segue:

$$\begin{aligned} T(n) &= \textstyle\sum_{i=1}^{m}(\sum_{j=(i+1)}^{m} O(1)) \\ &= O(\textstyle\sum_{i=1}^{m}\sum_{j=(i+1)}^{m} 1) \\ &= O(\textstyle\sum_{i=1}^{m} m - i) \\ &= O(m^2 - \textstyle\sum_{i=1}^{m} i) \\ &= O((m^2 - m)/2) \\ &= O(m^2) \\ &= O(n^2) \end{aligned}$$

Q4: Considere a função recursiva:

```
int f(int n, int m){
   if(n/m == 0)
      return 0;
   else
      return 1 + f(n/m, m);
}
```

- Determine o menor majorante assimptótico da função `f` medido em função dos parâmetros da função.

Solução:

$$\begin{aligned}
T(n) &= T(n/m) + O(1) \\
&= T(n/m^2) + 2.(O(1)) \\
&\vdots \\
&= T(n/m^k) + k.O(1) \\
&= T(n/m^k) + O(k)
\end{aligned}$$

Resolvendo $n/m^k = 1$ para k, obtemos $k = log_m\ n$. Concluímos portanto que: $T(n) = O(log_m\ n)$.