Escola de Artes, Ciências e Humanidades (USP-Leste)
Algoritmos e Estruturas de Dados I - 2º Semestre de 2015
Prof. Alexandre da Silva Freire (afreire@ime.usp.br – sala 322K-I1)

Aula 18 - 16/10

Prova de corretude de algoritmos recursivos

A seguir, mostraremos alguns exemplos de provas de corretude de algoritmos recursivos.

```
/*
 * Recebe: um vetor "v" de inteiros e um inteiro "n" >= 0
 *
 * Devolve: a soma dos elementos contidos no intervalo v[0 .. n-1]
 */
int calculaSoma(int[] v, int n) {
   if(n == 0) {
      return 0;
   }

   return v[n-1] + calculaSoma(v, n-1);
}
```

Lema 1. *O algoritmo* `calculaSoma` *está correto.*

Demonstração. Provaremos o lema por indução em n. Para $n = 0$, o intervalo $v[0 \mathrel{..} n-1]$ é vazio e, portanto, a soma de seus elementos é 0. Logo, nesse caso o algoritmo devolve a resposta correta. Suponha que $n > 0$. Assumiremos, por hipótese de indução, que a chamada recursiva devolve o valor S, que corresponde à soma dos elementos contidos no intervalo $v[0 \mathrel{..} n-2]$. Nesse caso, o algoritmo devolve $S + v[n-1] = \sum_{i=0}^{n-1} v[i]$ e, portanto, está correto. $\square$

```
/*
 * Recebe: um vetor "v" de inteiros e dois inteiros "i" e "j"
 *
 * Devolve: a soma dos elementos contidos no intervalo v[i .. j]
 */
int calculaSoma2(int[] v, int i, int j) {
   if(i > j) {
      return 0;
   }

   if(i == j) {
      return v[i];
   }

   int m = (i + j)/2;
   return calculaSoma2(v, i, m) + calculaSoma2(v, m+1, j);
}
```

Lema 2. *O algoritmo* `calculaSoma2` *está correto.*

Demonstração. Provaremos o lema por indução em $n = j - i + 1$. Para $n \leq 0$ (ou seja, $i > j$), o intervalo $v[i \,..\, j]$ é vazio e, portanto, a soma de seus elementos é 0. Para $n = 1$ (ou seja, $i = j$), o intervalo $v[i \,..\, j]$ contém um único elemento e, portanto, a soma de seus elementos é $v[i]$. Logo, nesses dois casos o algoritmo devolve a resposta correta. Suponha que $n > 1$. Assumiremos, por hipótese de indução, que as chamadas recursivas devolvem os valores S_1 e S_2, que correspondem às somas dos elementos contidos nos intervalos $v[i \,..\, m]$ e $v[m + 1 \,..\, j]$, respectivamente, onde $m = \lfloor \frac{i+j}{2} \rfloor$ e, consequentemente, $m - i = \lceil \frac{n}{2} \rceil$ e $j - (m+1) = \lfloor \frac{n}{2} \rfloor$. Nesse caso, o algoritmo devolve

$$S_1 + S_2 = \sum_{k=i}^{m} v[k] + \sum_{k=m+1}^{j} v[k] = \sum_{k=i}^{j} v[k]$$

e, portanto, está correto. □

O algoritmo Mergesort

```
/*
 * Recebe: um vetor de inteiros "v" e os inteiros "p", "m" e "u",
 *         tais que p <= m < u, e assume que os intervalos
 *         v[p .. m] e v[m+1 .. u] estao ordenados
 *
 * O que faz: ordena o intervalo v[p .. u]
 */
void merge(int[] v, int p, int m, int u) {
   // OBS: assuma que "int[] aux" foi devidamente declarado e alocado
   for (int i = p; i <= u; i++) {
      aux[i] = v[i];
   }

   int i = p;
   int j = m + 1;
   int k = p;

   while (i <= m && j <= u) {
      if (aux[i] <= aux[j]) {
         v[k++] = aux[i++];
      } else {
         v[k++] = aux[j++];
      }
   }

   while (i <= m) {
      v[k++] = aux[i++];
   }

   while (j <= u) {
      v[k++] = aux[j++];
   }
}
```

Lema 3. *O algoritmo* `merge` *está correto.*

Como o algoritmo `merge` é iterativo, não provaremos o Lema 3.

```
/*
 * Recebe: um vetor de inteiros "v" e dois inteiros "p" e "u"
 *
 * O que faz: ordena o intervalo v[p .. u]
 */
void mergesort(int[] v, int p, int u) {
   if (p < u) {
      int m = (p + u) / 2;
      mergesort(v, p, m);
      mergesort(v, m + 1, u);
      merge(v, p, m, u);
   }
}
```

Lema 4. *O algoritmo* `mergesort` *está correto.*

Demonstração. Provaremos o lema por indução em $n = u - p + 1$. Para $n \leq 1$ (ou seja, $p \geq u$), o intervalo $v[u \mathrel{..} p]$ é vazio ou possui um único elemento e, portanto, já está ordenado. Logo, nesse caso o algoritmo está correto. Suponha que $n > 1$. Assumiremos, por hipótese de indução, que as chamadas recursivas ordenam os intervalos intervalos $v[p \mathrel{..} m]$ e $v[m{+}1 \mathrel{..} u]$, onde $m = \lfloor \frac{p+u}{2} \rfloor$ e, consequentemente, $m - p = \lceil \frac{n}{2} \rceil$ e $u - (m + 1) = \lfloor \frac{n}{2} \rfloor$. Pelo Lema 3, a chamada ao a algoritmo `merge` ordena o intervalo $v[p \mathrel{..} u]$ e, portanto, o algoritmo está correto. $\square$

Exercícios

Exercício 1. O que faz o algoritmo `misterio1`? Prove que sua resposta está correta.

```
boolean misterio1(int[] v, int n, int i) {
   if(n == 0) {
      return false;
   }
   return v[n-1] == i || misterio1(v, n-1, i);
}
```

Exercício 2. O que faz o algoritmo `misterio2`? Prove que sua resposta está correta.

```
int misterio2(int[] v, int p, int u, int i) {
   if(p > u) {
      return 0;
   }
   int m = (p + u) / 2;
   int c = 0;
   if(v[m] == i) {
      c++;
   }
   return c + misterio2(v, p, m-1, i) + misterio2(v, m+1, u, i);
}
```

Exercício 3. Uma *inversão* em um vetor v é um par $\{v[i], v[j]\}$, tal que $i < j$ e $v[i] > v[j]$. Repare que o número de inversões de um vetor com n elementos está entre 0 e $\frac{n^2-n}{2}$. Escreva um algoritmo recursivo para calcular o número de inversões em um vetor. Seu algoritmo pode alterar o vetor original, se necessário. O consumo de tempo de seu algoritmo deve ser $O(n \log n)$. Prove que seu algoritmo está correto.

Exercício 4. O que faz o algoritmo `misterio3`? Prove que sua resposta está correta.

```
void misterio3(int[] v, int p, int u) {
   if (p < u) {
      int i = fazAlgumaCoisa(v, p, u);
      misterio3(v, p, i-1);
      misterio3(v, i+1, u);
    }
}
```

```
int fazAlgumaCoisa(int[] v, int p, int u) {
   int i = p;

   for (int j = p; j < u; j++) {
      if (v[j] > v[u]) {
         troca(v, i++, j);
      }
   }

   troca(v, i, u);
   return i;
}
```

Observação: apenas descreva **o que faz** a subrotina `fazAlgumaCoisa`, não precisa demonstrar que ela está correta (assuma que a função `troca` faz a troca dos respectivos elementos).