Cursos de Verão 2010 - IME/USP
Tópicos de Programação
Divisão e conquista
Prof. Alexandre da Silva Freire

1 Cálculo de potência

Considere o seguinte problema: dados dois números inteiros n e k, tal que $k \geq 0$, calcular n^k. Eis uma solução para o problema:

```
int potencia(int n, int k) {
   int i, total = 1;
   for(i = 0; i < k; i++) {
      total *= n;
   }
   return total;
}
```

O consumo de tempo do algoritmo acima é $\Theta(k)$. Este algoritmo é polinomial no tamanho da entrada? Antes de responder a esta pergunta, primeiro precisamos saber qual é o tamanho da entrada. Considerando que os números n e k estão codificados em base binária, temos que o tamanho da entrada é $\lg n + \lg k$. Como $2^{\lg k} = k$, conclui-se que o consumo de tempo do algoritmo `potencia` é exponencial. Isso não é nada bom! Vejamos uma outra solução:

```
   int potencia_rec(int n, int k) {
      int p;
(1)   if(k == 0) {
(2)      return 1;
      } else {
(3)       p = potencia_rec(n, k/2);
(4)       return (k%2 == 0)? (p*p) : (p*p*n);
      }
   }
```

Lema 1.1. *O algoritmo* `potencia_rec` *está correto.*

Lema 1.2. *O consumo de tempo do algoritmo* `potencia_rec` *é* $\Theta(\lg k)$.

As provas dos lemas 1.1 e 1.2 ficam como exercício (veja a lista 2).

2 Torres de Hanói

Torres de Hanói é um quebra-cabeça composto por uma base contendo três pinos, sendo que em um deles são empilhados alguns discos em ordem

decrescente de diâmetro (veja a figura 1). O objetivo é transferir todos os discos de um pino para outro qualquer, respeitando às seguintes regras: (1) só é permitido transferir um único disco por vez de uma torre para outra; (2) um disco maior não pode ser colocado sobre outro disco menor.

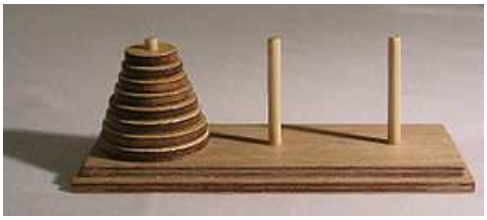

Figura 1: Torres de Hanói com 8 discos.

A seguir apresentamos um algoritmo para imprimir todos os movimentos necessários e suficientes para resolver o quebra-cabeça. O Algoritmo `hanoi` recebe como parâmetros um inteiro `n` e três caracteres representando os três pinos, sendo que o pino `inicial` contém `n` discos, o pino `final` corresponde ao pino que conterá os `n` discos após todos os movimentos serem executados, e o pino `auxiliar`, como o nome diz, será utilizado como pino auxiliar.

```
    void hanoi(char inicial, char final, char auxiliar, int n) {
       if(n == 1) {
          printf("%c -> %c\n", inicial, final);
       } else {
(2)       hanoi(inicial, auxiliar, final, n-1);
          printf("%c -> %c\n", inicial, final);
(4)       hanoi(auxiliar, final, inicial, n-1);
       }
    }
```

Lema 2.1. *O algoritmo* `hanoi` *está correto.*

Prova. Provaremos por indução em n. Suponha que $n = 1$. Neste caso o algoritmo "move" o disco que está no pino *inicial* para o pino *final* em um único movimento. Logo, para $n = 1$ o algoritmo está correto. Agora suponha que $n > 1$. Por hipótese de indução a chamada recursiva feita na linha (2) imprime todos os movimentos necessários e suficientes para mover os $n-1$ discos que estão no topo do pino *inicial* para o pino *auxiliar*, usando o pino *final* como auxiliar (antes da chamada o pino *final* está vazio e, portanto, podemos utilizá-lo como pino auxiliar). Depois da chamada recursiva da linha (2) o algoritmo "move" o único disco que está no pino *inicial* (o maior disco, no caso) para o pino *final*, que está vazio. Para completar o trabalho é feita a chamada recursiva da linha (4), que por hipótese de indução imprime todos os movimentos necessários e suficientes para mover os $n-1$ discos que estão no topo do pino *auxiliar* para o pino

final, usando o pino *inicial* como auxiliar (antes da chamada o pino *inicial* está vazio e, portanto, podemos utilizá-lo como pino auxiliar). Repare que todos estes movimentos são necessários, pois antes de mover o maior disco do pino *inicial* para o pino *final* é necessário que todos os outros discos estejam no pino *auxiliar* (para que o maior disco não seja empilhado sobre um disco menor). Portanto, para $n > 1$ o algoritmo está correto. ■

Seja $T(n)$ o consumo de tempo do algoritmo `hanoi`. Definiremos $T(n)$ através da seguinte recorrência:

$$T(n) = \begin{cases} 1, & \text{se } n = 1 \\ 2T(n-1) + 1, & \text{se } n > 1. \end{cases}$$

Lema 2.2. $T(n) = 2^n - 1$.

Prova. Provaremos por indução em n. Suponha que $n = 1$. Neste caso temos que $T(n) = 1 = 2^n - 1$. Agora suponha que $n > 1$. Por hipótese de indução vale que $T(n-1) = 2^{n-1} - 1$. Logo, $T(n) = 2T(n-1) + 1 = 2(2^{n-1} - 1) + 1 = 2^n - 1$. ■