INF 02779 Análise de Algoritmos - 2002/2	11/25/02
Aula 3: Programação Dinâmica	
Instrutor: Berilhes Borges Garcia	Escriba: Rafael Gabeira Cola

DRAFT

1 Programação Dinâmica

Idéia: A razão por que o algoritmo é tão lento é que este recomputa o mesmo problema um grande número de vezes.
Por exemplo, na árvore para $F(5)$ nós temos duas computaçõesde $F(3)$. A segunda vez que nós obtemos $F(3)$, nós estamos desperdiçando esforço computacional.
Ao invés de resolver o mesmo problema repetidamente, por que nós não resolvemos cada subproblema uma única vez e, então, reutilizamos a solução gerada quando necessário?
Esta idéia simples pode conduzir a alguns algoritmos complicados, os quais nós estudaremos oportunamente, contudo, por agora, nosso emprego será bastante simples.
Algoritmo de Programação Dinâmica

Algoritmo 1. *Algoritmo2*
Fib (n)

1. *f[1] ← 1*
2. *f[2] ← 1*
3. *for i ← 3 to n*
4. *f[i] ← f[i-1]+f[i-2]*
5. *return f[n]*

Este algoritmo é interativo ao invés de recursivo.
As três primeiras são sempre executadas. A quarta linha é executada $(n-1)$ vezes (exceto para $n = 1$) enquanto que a quinta linha é executada $(n-2)$ vezes (exceto para $n = 1$).
Portanto $T(n) = n - 1 + n - 2 + 3 = 2n$ (exceto para $n = 1$, $T(1) = 4$).
Como um exemplo para $n = 45$ o Algoritmo2 requer 90 passos, aproximadamente 10 milhões de vezes mais rápido que o Algoritmo1!!
Dessa forma, mesmo que esta rotina não seja freqüentemente utilizada, esta diferença é tão marcante que acabarápor ser notado. Nós podemos concluir que o Algoritmo2 é muito melhor que o primeiro.

2 Complexidade de Espaço

Tempo de execução não é a única métrica de desempenho. Tempo de desenvolvimento e tamanho do código são importantes, mas nós não discutiremos estas métricas neste curso.
Por outro lado nós estaremos, algumas vezes, interessados na quantidade de memória utilizada por um programa.

Se um programa requer um tempo considerável para executar, este poderá executar, desde que nós tenhamos paciência. Por outro lado, se um programa demanda muita memória, pode ser que nós não consegamos executá-lo!!

3 Análise de Espaço

Novamente, a análise procede de forma diferente para programas interativos e recursivos.
Para programas interativos, usualmente apenas observe as declarações das variáveis e chamadas às funções de alocação de memória dinâmica, tal como malloc() no C.
Por exemplo, o Algoritmo2, declara somente um vetor de tamanho $n + 1$.
Análise da demanda de espaço de um programa recursivo é mais compicada.
O espaço utilizado em qualquer instante é o espaço total utilizado por todas as chamadas recursivas ativas naquele instante de tempo.
Cada chamada recursiva no Algoritmo1 requer espaço constante: variáveis locais, argumentos de funções e endereço de retorno.
As chamadas ativas em qualquer instante de tempo formam um caminho na árvore de recursão. O comprimento deste caminho é no máximo n, dessa forma o espaço demandado pelo algoritmo recursivo é novamente (algum fator constante vezes)n.
Nós abreviamos " algum fator constante vezes" por meio da notação "O" : $O(n)$.

4 Melhorando a Demanda por Espaço

O Algoritmo2 pode ser modificado de modo a utilizar menos espaço.
Idéia principal: cada passo através do loop utiliza somente dois valores de $F(n)$.
Ao invés de alocar um vetor de tamanho $n + 1$, simplesmente utilize duas variáveis.

Algoritmo 2. *Algoritmo3*
Fib(n)

1. *a ⟵ 1*
2. *b ⟵ 1*
3. *for i ⟵ 3 to n*
4. *c ⟵ a+b*
5. *a ⟵ b*
6. *b ⟵ c*
7. *return a*

As duas atribuições extras colocadas após a soma deslocam os valores para a próxima iteração.
O Algoritmo3 utiliza aproximadamente 4n linhas para computar F(n): mais lento que o Algoritmo2, porém utiliza muito menos espaço.
Números de Fibonacci

	Tempo	Espaço
Alg1	$O(2^n)$	$O(n)$
Alg2	$O(n)$	$O(n)$
Alg3	$O(n)$	$O(1)$

Nós podemos fazer melhor?