EFICIÊNCIA DO ALGORITMO

Matemática e algoritmos recursivos

Como analisar algoritmos recursivos usando um framework geral?
Começamos com o exemplo simples de calcular o fatorial de um número. Para isso, usamos uma função recursiva que se baseia na relação $F(n)=F(n-1)\times n$F(n)=F(n−1)×n.
Ao estabelecer uma relação de recorrência para o número de multiplicações necessárias para calcular $F(n)$F(n), descobrimos que é $M(n)=n$M(n)=n, uma solução linear.
Em seguida, exploramos outro problema clássico, a Torre de Hanói, que possui uma solução recursiva. Analisamos o número de movimentos necessários para resolver o que encontramos como $M(n)=2n-1$M(n)=2n−1, uma solução exponencial.

Finalmente, examinamos uma função que calcula o número de dígitos em uma representação binária de um número decimal. A análise dessa função resultou em $A(n)=\log_2 n$A(n)=log2n, uma solução logarítmica.
Esses exemplos ilustram como definir uma relação de recorrência para analisar algoritmos recursivos, resolver essa relação e, em seguida, determinar a eficiência do algoritmo em termos de tempo. Essas técnicas são essenciais para compreender a complexidade de algoritmos recursivos e são amplamente aplicáveis em diversos problemas.
Fibonacci é um grande exemplo de recursão.

Análise Empírica de algoritmos

É uma abordagem alternativa à análise matemática de sua eficiência. Ela envolve realizar experimentos práticos para medir o desempenho do algoritmo em termos de tempo de execução ou número de operações executadas.
Entenda o propósito do experimento
Decida a métrica de eficiência
Decida sobre as características da amostra de entrada
Prepare o programa
Gere uma amostra de entradas
Execute o algoritmo e registre os dados
Analise os dados

Escolher a amostra de entrada(número de operações executadas pelo algoritmo ou medir o tempo de execução), você precisa decidir o tamanho, a faixa e se os dados serão gerados de maneira aleatória ou seguindo algum padrão específico.
Os dados coletados podem ser apresentados em tabelas ou gráficos, o que pode ajudar a entender o desempenho do algoritmo de diferentes maneiras.
A análise empírica é uma ferramenta poderosa para entender o desempenho dos algoritmos na prática, complementando a análise matemática, que é mais teórica e abstrata.

não recursivos

1. Exemplo 2: Verificação de Elementos Únicos em uma Lista:
   O algoritmo verifica se todos os elementos em uma lista são distintos.
   O número de comparações no pior caso é aproximadamente $\frac{n^2}{2}$n222n2.
2. Exemplo 3: Multiplicação de Matrizes:
   O algoritmo calcula o produto de duas matrizes quadradas de ordem nn.
   O número de multiplicações é n^3n3.
3. Exemplo 4: Contagem de Dígitos Binários:
   O algoritmo conta o número de dígitos binários em uma representação binária de um número decimal.
   O número de vezes que a comparação $n>1$n>1 é executada é $\lfloor \log_2 n \rfloor +1$⌊log2n⌋+1.

Matemática e algoritmos não recursivos

Eemplo 1: Encontrando o Maior Elemento em uma Lista:
   O algoritmo verifica o maior elemento em uma lista de números.
   A operação básica é a comparação de elementos na lista.
   O número de comparações é $n-1$n−1, onde nn é o tamanho da lista.
2. Plano Geral para Analisar a Eficiência de Algoritmos Não Recursivos:
   Decidir sobre o parâmetro que indica o tamanho da entrada.
   Identificar a operação básica do algoritmo.
   Definir uma soma para contar o número de vezes que a operação básica é executada.
   Calcular a soma usando fórmulas e regras de manipulação de somas.

Notação Assintótica

Mostra como o tempo de execução cresce a medida que aumenta a entrada.(excluindo a linguagem de programação e a máquina que executa.)
Big O -> f ∈ O(g), dado g: ℕ->ℕ
O(g) = {f: ℕ->ℕ | Existem m ∈ ℕ* e c ∈ ℝ*+ tais como para todo n>=m, 0<=f(n)<= c.g(n)}(limite superior(teto))
Ω -> f ∈ Ω(g), dado g: ℕ->ℕ | {existem m ∈ ℕ* e c ∈ R*+ tais que, para todo n>m, 0<=c.g(n)<=f(n)}(limite inferior(piso))
TEOREMA: para f,g; f(n) = O(g(n))<=>g(n) = Ω
θ -> g:ℕ->ℕ, θ(g)= {f:ℕ->ℕ | existem m ∈ ℕ* e c1, c2 ∈ R* tais que, para todo n>= m, 0<= c1 .g(n) <= f(n)<= c2 .g(n)
TEOREMA: para f,g; f(n) = O(g(n)) <=>(f(n) = O(g(n)) e f(n) = Ω(g(n)))

Formas de eficiência

- Eficiência de tempo que se refere ao tempo de rapidez que o algoritmo é executado.
- Eficiência de espaço diz a respeito à quantidade de memória utilizada pelo algoritmo (além daquela requeriada para entradas e saídas).

Aplicação de framework de análise

- Para alguns algoritmos, precisamos distinguir entre as eficiências no pior caso, caso médio e melhor caso.
- O interesse principal do framework reside na ordem de crescimento do tempo de execução do algoritmo (unidades de memória extras consumidas) à medida que o tamanho da entrada tende ao infinito.
- A ordem de crescimento do tempo de execução do algoritmo para entradas grandes (já que as pequenas não são significativas), e é essencial para distinguir entre algoritmos eficientes e ineficientes.

Visualização de Algoritmo

O conceito de visualização de algoritmos adiciona uma dimensão visual ao estudo dos algoritmos. Pode ser estático, mostrando uma série de imagens que representam o progresso do algoritmo, ou dinâmico, criando uma apresentação contínua semelhante a um filme das operações do algoritmo.

A visualização de algoritmos tem duas aplicações principais: pesquisa e educação. Na pesquisa, pode revelar características desconhecidas dos algoritmos, enquanto na educação pode ajudar os alunos a compreender melhor os algoritmos. No entanto, sua eficácia na educação ainda é incerta e pode depender do nível de envolvimento dos alunos com a visualização.