Algoritmos de Engenharia

Programa de Pós-Graduação em

Engenharia Elétrica e de Computação

Centro de Tecnologia – UFRN

2. Projeto e Análise - Introdução

- Capítulo 4

O projeto de dividir para conquistar

- Essa é uma estratégia poderosa e recursiva para a solução de problemas, assim como dividir um grande projeto de engenharia em tarefas menores e gerenciáveis.
- **As três etapas principais**:
  - **Dividir**: Divida um problema grande e complexo em subproblemas menores e independentes.
  - **Conquistar**: resolver os subproblemas menores. Se eles ainda forem muito grandes, repita a etapa "Dividir" neles. Isso continua até que os problemas sejam triviais de resolver (o "caso base").
  - **Combinar**: Pegue as soluções dos subproblemas e mescle-as novamente para formar a solução para o grande problema original.

Modelagem de Desempenho: relações de recorrência

- **O que é uma recorrência?** É uma equação matemática que modela o desempenho de um algoritmo de dividir para conquistar. Pense nisso como uma especificação de desempenho.
- **A Forma Geral:**
  - `T(n) = a * T(n/b) + f(n)`
    - **T(n):** O tempo total (custo) para resolver um problema de tamanho n.
    - **a:** Em quantos subproblemas você divide o problema principal.
    - **n/b**: O tamanho de cada um desses subproblemas (por exemplo, n/2 significa metade do tamanho original).
    - **f(n)**: O custo do "trabalho" que você faz nas etapas Dividir e Combinar. Essa é a sobrecarga de gerenciar o processo.

Exemplo 1: Multiplicação padrão de matrizes

- **O problema:** multiplicar duas matrizes n x n. O método iterativo padrão é $\Theta(n^3)$.
- **A abordagem de dividir para conquistar**:
  - **Dividir**: Divida cada matriz n x n em quatro submatrizes n/2 x n/2.
  - **Conquistar**: Realize 8 multiplicações recursivas nessas submatrizes n/2 x n/2.
  - **Combinar**: Some os resultados. Esta etapa leva tempo Θ(n2).
  - **Modelo de desempenho**: T(n) = 8 * T(n/2) + Θ(n2)
  - **Resultado**: A solução ainda é $\Theta(n^3)$. Dividimos o problema, mas não ganhamos nenhuma eficiência assintótica.

Exemplo 2: Multiplicação de matrizes "inteligente" de Strassen

- **A sacada**: Podemos fazer uma troca? E se fizermos um trabalho mais "barato" para evitar fazer um trabalho "caro"?
- **Método de Strassen:**
  - Por meio de algumas manipulações algébricas inteligentes, Strassen encontrou uma maneira de multiplicar as quatro submatrizes usando apenas 7 multiplicações recursivas, mas exigiu 18 adições / subtrações.
  - O Trade-off: A adição de matrizes ($\Theta(n^2)$) é muito mais barata do que a multiplicação de matrizes ($\Theta(n^3)$). O método de Strassen aumenta o custo de "Combinar", mas reduz significativamente o número de etapas de "Conquistar“.
- **Modelo de desempenho**: $T(n) = 7 * T(n / 2) + \Theta(n2)$
- **Resultado**: A solução é $\Theta(n^{\log_2 7})$ que é aproximadamente $\Theta(n^{2.81})$. Esta é uma melhoria assintótica significativa e um exemplo clássico de engenhosidade algorítmica.

Resolvendo recorrências: o Método Mestre

- O **Método Mestre** é um "livro de receitas" ou "tabela de pesquisa" para resolver recorrências da forma T(n) = a * T(n/b) + f(n).
- Isso nos poupa de fazer derivações complexas à mão.
- **A ideia central**: Ele compara o custo das etapas de Dividir / Combinar (`f(n)`) com o custo de resolver os subproblemas do caso base (relacionados a $n^{\log_b a}$). O custo dominante determina o tempo de execução geral.

Os três casos do Método Mestre

- **Caso 1: Dominado por folhas**
  - Se o custo de resolver os subproblemas nas folhas da árvore de recursão cresce polinomialmente mais rápido do que o custo de divisão / combinação, **as folhas dominam**.
  - **Resultado**: O tempo de execução é determinado pelo número de folhas: $\Theta(n^{\log_b a})$.
- **Caso 2: Trabalho equilibrado**
  - Se o custo for distribuído uniformemente em todos os níveis da árvore, o trabalho será equilibrado.
  - **Resultado**: O tempo de execução é o custo em um nível vezes o número de níveis: $\Theta(n^{\log_b a} \log n)$.
- **Caso 3: Dominado pela raiz**
  - Se o custo da etapa de divisão / combinação crescer polinomialmente mais rápido do que o custo nas folhas, a etapa inicial de divisão / combinação domina.
  - **Resultado**: O tempo de execução é simplesmente o custo desse trabalho de nível superior: $\Theta(f(n))$.

A conclusão

- **Dividir e conquistar** é um padrão de projeto fundamental para lidar com problemas de grande escala, comum em áreas como processamento de sinal (FFT), gráficos e manipulação de grandes conjuntos de dados.
- **O desempenho não é de graça**: O ganho de eficiência depende inteiramente do trade-off entre o número de subproblemas (a), seu tamanho (n/b) e a sobrecarga de divisão e combinação (f(n)).
- **O Método Mestre** é um atalho poderoso para os analisarmos rapidamente a escalabilidade desses tipos de algoritmos recursivos sem nos perdermos nos detalhes matemáticos.

2. Projeto e Análise - Introdução

- Capítulo 5

Por que introduzir a aleatoriedade?

- **O mundo real é imprevisível**: muitas vezes não sabemos as características exatas dos dados de entrada que nossos algoritmos enfrentarão. Será resolvido? Aleatório? Ou um adversário fornecerá intencionalmente a pior contribuição possível?
- **Algoritmos determinísticos**: têm um comportamento fixo e previsível. Para uma determinada entrada, eles sempre fazem a mesma coisa. Isso pode torná-los vulneráveis a entradas de pior caso que causam baixo desempenho (por exemplo, $\Theta(n^2)$ para Quick Sort em uma matriz classificada).
- **Algoritmos aleatórios**: introduza aleatoriedade no próprio algoritmo. Ao fazer escolhas aleatórias, o comportamento do algoritmo muda a cada execução, mesmo com a mesma entrada.
- **A vantagem prática**: Um algoritmo aleatório não pode ser consistentemente derrotado por uma entrada "ruim" específica. Seu desempenho depende do acaso, não da estrutura da entrada, dando-nos um desempenho médio mais confiável.

Um exemplo motivador: o problema de contratação

- **Cenário**: Você precisa contratar um assistente. Uma agência envia n candidatos, um por dia. Você entrevista cada um deles e deve decidir imediatamente contratá-los ou rejeitá-los. Se você contratar alguém novo, deverá demitir o atual.
- **Objetivo**: Contratar o melhor candidato, mas minimizar o número de eventos dispendiosos de contratação/demissão.
- **Estratégia simples**: contrate o primeiro candidato. Então, para cada candidato subsequente, se ele for melhor do que o seu melhor atual, demita o antigo e contrate o novo.
- **A pergunta**: Quantas vezes esperamos contratar alguém?

Duas maneiras de analisar o problema

1. Análise probabilística (assumindo um mundo aleatório)
   - Analisamos nossa estratégia de contratação **determinística**.
   - Fazemos uma **suposição** crucial: os candidatos chegam em uma ordem completamente aleatória (qualquer uma das n! permutações é igualmente provável).
   - Calculamos o desempenho do **caso médio** com base nessa suposição.
2. Algoritmo Aleatório (Controlando a Aleatoriedade)
   - Mudamos nossa estratégia. A agência nos dá a lista de n candidatos antecipadamente.
   - Nosso primeiro passo: nós mesmos embaralhamos (permutamos) aleatoriamente a lista.
   - Em seguida, aplicamos a mesma lógica determinística de contratação à nossa lista embaralhada.
   - Calculamos o desempenho esperado. Esse resultado vale para qualquer lista inicial de candidatos porque introduzimos a aleatoriedade. Esta é uma garantia muito mais forte.

Uma ferramenta prática: variáveis aleatórias de indicadores

- **O conceito**: Uma maneira simples, mas poderosa, de contar eventos. Pense nisso como um botão liga / desliga.
- **Definição**: Uma variável indicadora I{A} é:
  - 1 se ocorrer o evento A
  - 0 se o evento A não ocorrer
- **A propriedade da chave**: O valor esperado de uma variável indicadora é simplesmente a probabilidade de o evento ocorrer. E[I{A}] = Pr{A}.
- **Por que é útil**: Permite-nos dividir um problema complexo em uma soma de eventos simples 0/1, tornando muito mais fácil calcular o resultado total esperado.

Analisando o problema de contratação

- Seja X o número total de vezes que contratamos alguém.
- Seja X_i uma variável indicadora onde X_i = 1 se contratarmos o candidato i e 0 caso contrário.
- O total de contratações X = X_1 + X_2 + ... + X_n.
- Por linearidade de expectativa, E [X] = E [X_1] + E [X_2] + ... + E [X_n].
- **Qual é a probabilidade de contratar o candidato i?**
  - Contratamos o candidato i somente se ele for o melhor entre os primeiros candidatos.
  - Como a ordem é aleatória, qualquer um desses primeiros i candidatos tem a mesma probabilidade de ser o melhor.
  - portanto, a probabilidade é 1/i.
- **O resultado**:
  - E[X] = 1/1 + 1/2 + 1/3 + ... + 1/n. Esta é a Série Harmônica.
  - E[X] ≈ ln(n).
- **Conclusão**: De n candidatos, esperamos contratar apenas cerca de ln(n) vezes. Esta é uma grande melhoria em relação ao pior caso de n contratações.

Como permutar aleatoriamente um vetor

- Uma necessidade comum de algoritmos aleatórios é embaralhar um vetor de entrada.
- **Um método simples e eficiente (embaralhamento de Fisher-Yates):**
  - Para i de 1 a n:
  - Escolha um inteiro aleatório j entre i e n (inclusive).
  - Troque os elementos em A[i] e A[j].
- Este laço simples é executado em tempo Θ(n) e garante que cada permutação seja igualmente provável.

A conclusão

- **Abrace a aleatoriedade**: a randomização é uma ferramenta poderosa para melhorar a confiabilidade de um algoritmo e protegê-lo das entradas do pior caso.
- **Probabilístico vs. Randomizado**:
  - A análise probabilística é uma forma de entender o desempenho com base em suposições sobre o mundo.
  - Algoritmos aleatórios são uma forma de garantir o desempenho, assumindo o controle da incerteza você mesmo.
- **Aplicações**: Essa abordagem é fundamental em muitas áreas, incluindo simulação (métodos de Monte Carlo), criptografia, aprendizado de máquina e roteamento de rede, onde as entradas geralmente são imprevisíveis.

Lista de Exercícios de Projeto e Análise

4. Mostre numericamente com suas implementações dos algoritmos de multiplicação de matrizes que o algoritmo de Strassen é mais rápido que o algoritmo convencional.
5. Escolha um algoritmo recorrente para aplicar um dos 4 métodos de resolução de recorrência descritos no capítulo 4 para medir o custo da recorrência do algoritmo escolhido. Compare o resultado com medições de tempo.
6. O problema de balanceamento de cargas busca atribuir tarefas de tamanhos diferentes a trabalhadores, de modo a minimizar a carga máxima que um trabalhador irá executar. Em um problema em que temos n tarefas e k trabalhadores ($n > k$), considere que o balanceador irá distribuir as n/k primeiras tarefas para o primeiro trabalhador, as n/k tarefas seguintes para o segundo trabalhador, e assim por diante. Mostre numericamente como permutar aleatoriamente os dados de entrada, que são as cargas de cada tarefa, pode influenciar na solução desse balanceador.