INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
PARAÍBA
Campus João Pessoa

POP

CNPq
Conselho Nacional de Desenvolvimento Científico e Tecnológico

Projeto Olímpico de Programação

Divide and Conquer

Paulo Felipe paulofelipe.feitosa@gmail.com

Divisão e Conquista

- Divisão e conquista é um paradigma para solução de problemas que quando divididos em pequenas partes se tornam mais simples.

- Passos para solução:
  - **1.** Divida o problema original em subproblemas;
  - **2.** Encontre as soluções para cada subproblema - essas soluções devem ser fáceis;
  - **3.** Se necessário, combine as subsoluções para obter a solução completa para o problema principal.

Divisão e Conquista

- Divisão e Conquista é base para solução de vários problemas como:
  - Ordenação: Quick Sort, Merge Sort, Heap Sort;
  - Busca: Binary Search;
  - Estrutura de dados: Binary Search Tree, Heap, Segment Tree, Fenwick Tree;
  - Multiplicação de números: Algoritmo de Karatsuba e Ofman;
  - Estatística: Achar a mediana em um espaço amostral.

Divisão e Conquista

- (Aplicações na programação paralela) Problemas que utilizam esta técnica podem tirar proveito de máquinas com múltiplos processadores, pois, a fase de divisão em problemas menores proporciona uma divisão natural do trabalho. Cada um dos problemas menores obtidos pode ser calculados separadamente em um processador sem depender dos demais.

Busca Binária

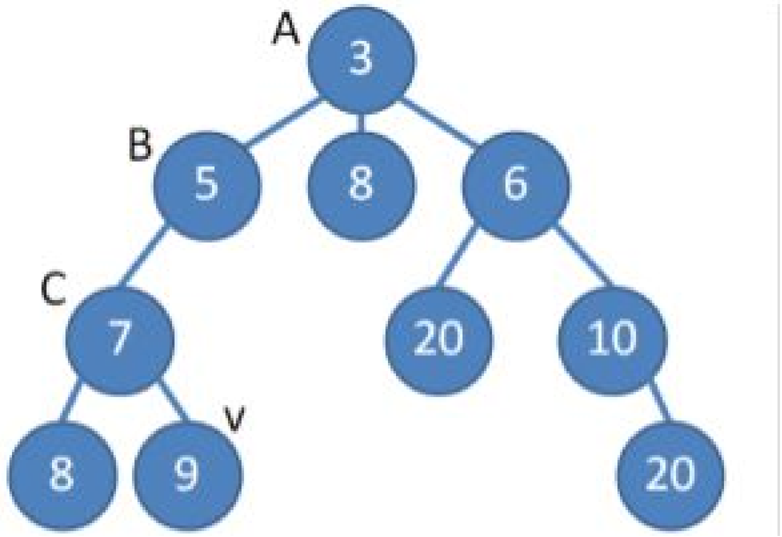
- Vamos estudar a Busca Binária para resolver problemas de uma forma não óbvia;

Busca Binária - 1º Problema

- ‘My Ancestor’ - Thailand ICPC National Contest 2009. Resumo do problema: Dada uma árvore com N <= 80K vértices, e estes vértices com valores que vão aumentando seu valor da raiz até os vértices folhas. Encontre um vértice ancestral (pai) que está contido no caminho da raiz até um vértice ***v*** que tem o valor de pelo menos P. Serão realizadas pelo menos Q <= 20K queries.

Busca Binária - 1º Problema

- Examine a figura:



- Se P = 4, então a resposta é o vértice marcado com 'B' e com o valor 5, pois, ele é um ancestral de ***v*** e está no caminho da raiz até o vértice ***v***. Se P = 7, então a resposta é C, se P >= 9, então não existe resposta.

Busca Binária - 1º Problema

- Vamos discutir soluções para este problema!!!
- O(N) para cada query, que tem como resposta O(Q*N).
  - Para cada query, eu subo na minha árvore do vértice ***v*** até a raiz procurando minha solução.
- O(Q*log(N)).
  - Primeiro vamos armazenar todas as ***queries***, depois iremos percorrer toda a árvore só uma vez, começando da raiz e através do algoritmo de pré-ordem (travessia em árvore). Vamos modificar esse algoritmo para produzir um vetor "caminho" da raiz até o "nó atual" na busca, sabemos que este vetor caminho está ordenado, pois, é uma propriedade deste problema.

Busca Binária - 1º Problema

- O(Q*log(N)) - Continuação.
  - Este algoritmo irá produzir, no caso da figura do slide 7, os seguintes vetores caminho: {{3}, {3, 5}, {3, 5, 7}, {3, 5, 7, 8}, {3, 5, 7, 9}, {3, 8}, {3, 6}, {3, 6, 20}, {3, 6, 10}, {3, 6, 10, 20}}.
  - Durante a travesia em pré-ordem, nós podemos encontrar um vértice que se encontra em uma ***query***, logo, podemos aplicar uma busca binária O(log N) para saber se existe algum vértice com um valor de pelo menos P.

Método da Bisseção - 2º Problema

- O Método da Bisseção (Cálculo Numérico) é um método de busca de raízes em uma função, também chamado de método da pesquisa binária, pois, utiliza do principio da busca binária.

Explicação

- Estamos interessados em algumas análises desse método, como, o número máximo de iterações necessárias (sem que algoritmo entre em **loop infinito**) para que a resposta do método esteja dentro de uma predeterminada faixa de erro.

Método da Bisseção - 2º Problema

- Sendo **n** o número de iterações máximas, logo, **n** é dado por:

$$n = \log_2\left(\frac{\epsilon_0}{\epsilon}\right) = \frac{\log \epsilon_0 - \log \epsilon}{\log 2},$$

sendo ϵ_0 o tamanho do intervalo inicial, isto é, $\epsilon_0 = b - a$.

- Com este o valor **n** conhecido, nós podemos resolver o Método da Bisseção (achar a raiz de uma função em um determinado intervalo) através de uma busca binária.

Método da Bisseção - 2º Problema

- Algoritmo:

Início

i = 1

Enquanto *(i <= N) && (|f(mid)| > ERRO ESTABELECIDO)*

mid = (a + b)/2

Se *Sinal(f(a)) == Sinal(f(mid))* ***então***

a = mid

Senão

b = mid

i = i + 1

Fim

Fim

Método da Bisseção - 2º Problema

- Você comprou um carro através de um empréstimo no banco e você deseja pagar este empréstimo mensalmente em parcelas de ***d*** reais em ***m*** meses. Suponha que o valor do carro seja de ***v*** reais e o banco cobra uma porcentagem ***i*** de juros em cima da dívida atual.
- Qual é o valor da parcela ***d*** que você deve pagar por mês sabendo que o método de cobrança de juros do banco é desse forma: Suponha ***d*** = 576.19, ***m*** = 2, ***v*** = 1000 e ***i*** = 10%, no primeiro mês seu débito será (1000 * 1.1) = 1100 e você irá pagar ***d***, logo o valor da sua dívida após pagar a primeira parcela será, 1100 - 576.19 = 523.81.

Método da Bisseção - 2º Problema

- No segundo mês a sua dívida será (523.81 * 1.1) = 576.191 e você irá pagar 576.19, logo, você irá dever aproximadamente 0 no segundo mês, liquidando sua dívida com o banco. Lembrando, que o valor mínimo que podemos pagar é 1 centavo, ou 0.01 reais.

- Pois bem, o problema é o seguinte se dermos o ***m***, ***v*** e ***i*** como eu encontro o valor ***d*** qual que no mês ***m*** eu liquido minha dívida com o banco, sem pagar mais do que o necessário, ou sem pagar menos do que o necessário? Em palavras:

$$f(d, m, v, i) \sim= 0$$

Método da Bisseção - 2º Problema

- Vamos aplicar o Método da Bisseção.
  - Ideias!?

- Primeiro temos que definir nosso intervalo de interesse, um **lower bound** e um **upper bound**, ou seja, o nosso **a** e o nosso **b**.

- Definimos nosso **ERRO** e encontramos o número **n** de iterações máximas (slide 11), normalmente ERRO entre [1e-9, 1e-15].

- Aplicamos o Algoritmo!!!

Binary Search - 3º Problema

- Pow, Paulo, mas esse método aí não é só para Cálculo Númerico não?
  - Solve It;
  - Through the Desert.

- Normalmente, vamos dar de cara com problemas em que dentro da busca binária fazemos a simulação do problema e retornamos o resultado para a busca, estrutura da solução do Problema: Through the Desert.

Binary Search - 3º Problema

- Solução:

```
#define EPS 1e-9  // this value is adjustable; 1e-9 is usually small enough
bool can(double f) {                    // details of this simulation is omitted
  // return true if the jeep can reach goal state with fuel tank capacity f
  // return false otherwise
}

// inside int main()
  // binary search the answer, then simulate
  double lo = 0.0, hi = 10000.0, mid = 0.0, ans = 0.0;
  while (fabs(hi - lo) > EPS) {         // when the answer is not found yet
    mid = (lo + hi) / 2.0;                          // try the middle value
    if (can(mid)) { ans = mid; hi = mid; } // save the value, then continue
    else          lo = mid;
  }

  printf("%.3lf\n", ans);     // after the loop is over, we have the answer
```

Merge Sort

- **Problema**: Rearranjar um vetor *A[p … r]* de modo que ele fique em ordem crescente.
- "**Divisão e conquista de verdade**": Merge Sort - Algoritmo de Ordenação. "Uma imagem fala mais do que mil palavras".

Merge Sort

- Complexidade **O(n log n)** para qualquer caso, diferentemente do Quick Sort.

Merge Sort

- **Algoritmo:**

```
MERGESORT (A, p, r)
se p < r então
    q ← ⌊(p+r)/2⌋
    MERGESORT (A, p, q)
    MERGESORT (A, q+1, r)
    INTERCALA (A, p, q, r)
```

- O algoritmo divide o vetor *A* em dois vetores *A'[p … q]* e *A''[q+1 … r]*, onde $q = (p+r)/2$.
- O caso base do Merge Sort, onde o subproblema se torna muito fácil de resolver, é quando $p == r$, pois, o vetor *A[p … r]* só tem um elemento, e ele já está ordenado.

Merge Sort

- Então, resolvido o subproblema, é necessário combinar as subsoluções do problema para formar a solução do problema principal.

- Após o retorno, será necessário juntar os dois vetores A' e A'' em um vetor só A, tal que, A esteja ordenado. Sabemos que A' e A'' já estão ordenados, como obter A de forma eficiente?
  - Algoritmos de ordenação?
  - Vamos à análise!

Merge Sort

- Função Intercala:

```
INTERCALA (A, p, q, r)
para i crescendo de p até q faça
    B[i] ← A[i]
para j crescendo de q+1 até r faça
    B[r+q+1–j] ← A[j]
i ← p
j ← r
para k crescendo de p até r faça
    se B[i] ≤ B[j]
        então A[k] ← B[i]
              i ← i+1
        senão A[k] ← B[j]
              j ← j–1
```

INSTITUTO FEDERAL DE
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
PARAÍBA
Campus João Pessoa

POP

CNPq
Conselho Nacional de Desenvolvimento Científico e Tecnológico

Projeto Olímpico de Programação

Divide and Conquer

Paulo Felipe paulofelipe.feitosa@gmail.com