Dividir e conquistar

Professor Gabriel Tostes

O que é?

É um algoritmo (ou uma técnica) que baseia-se na ideia de dividir recursivamente um problema em problemas menores que sejam facilmente resolvidos, e depois combinar as soluções deles.

Intuição para dividir e conquistar:

- Quando você vai dividir uma pizza, você corta ⅛ dela por vez ou você divide ela ao meio 4 vezes? (Você divide a pizza em duas até chegar ao problema menor de dividir um pedaço ao meio)
- Quando você está procurando uma página em um livro você vai página por página ou você abre em uma página aleatória e vê se a página procurada é menor ou maior que a aberta? (você divide o tamanho do livre em 2 e vê que só uma das partes contêm sua página procurada)

Exemplos de utilização

- Ordenação: **Merge Sort**(exemplo típico), Heap Sort, Quick Sort.
- **Busca binária** (procurar páginas em um livro)
- Estruturas de dados que envolvem árvores binárias como Binary Search Tree(árvore de busca binária!, é outra maneira de ver a busca binária), Segment Tree e Fenwick Tree.
- Calcular rapidamente uma exponenciação.
- Calcular raízes de uma equação

Métodos de Ordenação

Merge Sort

6 5 3 1 8 7 2 4

Merge Sort

Dos três listados anteriormente (Merge sort, QuickSort e Heap sort) o Merge Sort é o exemplo mais típico do método de divisão e conquista.

Ele funciona da seguinte maneira:

-

Se quisermos ordenar uma sequência $a_1 a_2 \ldots a_n$ podemos ordenar primeiro

$$a_1 a_2 \ldots a_{n/2} \quad \text{e} \quad a_{n/2} a_{1+n/2} \ldots a_n$$

e, a partir dessas duas sequências ordenadas, mesclá-las retirando em cada passo o menor termo geral (seja da primeira ou da segunda sequencia)

Merge Sort - Exemplo

Então vamos ordenar o vetor[]={1,6,2,11,5,3,12}

Se ordenarmos {1,6,2,11} e {5,3,12} teríamos v1={1,2,6,11} e v2={3,5,12} (claro, esse passo de ordenação é feito recursivamente. O caso base da ordenação é quanto o vetor tem tamanho 1, pois ele já é ordenado). Vamos então recriar o vetor[] a partir de v1 e v2:

	Passo 1	Passo 2	Passo 3	Passo 4	Passo 5	Passo 6	Passo 7
v1	{2,6,11}	{6,11}	{6,11}	{6,11}	{11}	{}	{}
v2	{3,5,12}	{3,5,12}	{5,12}	{12}	{12}	{12}	{}
vetor[]	{1}	{1,2}	{1,2,3}	{1,2,3,5}	{1,2,3,5,6}	{1,2,3,5,6,11}	{1,2,3,5,6,11,12}

Merge Sort - Complexidade

Se T(n) for a complexidade do algoritmo em função do tamanho ‘n’ do vetor. Então, para ordenar chamamos duas vezes a função merge-sort para dois vetores de tamanho n/2, e para mesclá-los usamos n passos.

Então $T(n)=2*T(n/2)+n$ …

$2*T(n/2)=4*T(n/4)+n$…

$2^{\log n}T(1)= 0 + n$ …

somando telescópicamente: $T(n)=n\log n$, ou seja, a complexidade é $O(n\log n)$

Merge sort - Implementação

Para facilitar a implementação, não é necessário criar um vetor e atualizar sempre o tamanho dele, retirando elementos. Basta apenas guardar a posição (pos) do primeiro elemento do vetor que não foi retirado. Quando ele for retirado simplesmente fazemos pos++;

Nessa implementação de Merge_sort usaremos a estrutura de dados vector da biblioteca <vector>. Caso não conheça entre em: http://www.cplusplus.com/reference/vector/vector/ Basicamente ele é um vetor mas com memória alocada dinamicamente. Ao contrário de vetores estáticos que declaramos o tamanho dele apenas uma vez como int a[30]. Declaramos o vector e cada vez que devemos colocar um novo elemento usamos vector.push_back(elemento) que aumenta o tamanho do vetor e coloca o elemento no novo espaço reservado.

Merge sort - implementação

Esse código abaixo lê um número n, depois lê um vetor de tamanho n e printa ele ordenado:

```cpp
#include<bits/stdc++.h> // declaramos todas as bibliotecas de c++ e c.
using namespace std; // deve ser colocado antes de algumas funções de c++
//basicamente escreve std:: antes de todas linhas do código.

vector < int > merge_sort(vector < int> a){
    if(a.size()==1) return a; // caso base, se o vector tiver tamanho 1 ele já está ordenado.
    vector < int > a1, a2, res; //declaramos a1(a1 ate a_n/2), a2(a_n/2 +1 até a_n)
//    e res para guardar o vetor a ordenado
    for(int i=0; i<a.size()/2; i++) a1.push_back(a[i]); //colocamos os elementos em a1
    for(int i=a.size()/2; i<a.size(); i++) a2.push_back(a[i]); //colocamos os elementos em a2
    a1=merge_sort(a1), a2=merge_sort(a2); //ordenamos a1 e a2 recursivamente
    int i=0, j=0;
    while(i<a1.size() || j<a2.size()){ //enquanto nao tivermos colocado todos os elementos dos dois vetores
        if(j==a2.size()) res.push_back(a1[i]), i++; //se já colocamos todos de a2 coloque o elemento de a1 e
        //atualize i (a posicao do menor elemento nao retirado de a1)
        else if(i==a1.size()) res.push_back(a2[j]), j++; // igual ao anterior mas invertendo a1 e a2
        else if(a1[i]<a2[j]) res.push_back(a1[i]), i++; // se o elemento de a1 é menor que o de a2
        // colocamos o elemento de a1 e atualizamos a posicao com i++
        else res.push_back(a2[j]), j++; // mesmo do anterior.
    }
    return res; //retorne o vetor a ordenado.
}

int main(){
    int n;
    cin >> n;
    vector < int > a(n);
    for(int i=0; i<n; i++) cin >> a[i];
    a=merge_sort(a);
    for(int i=0; i<n; i++) cout << a[i] << " ";
    cout << "\n";
}
```

Quick Sort

6 5 3 1 8 7 2 4

Quick Sort

O algoritmo para o Quick sort é:

- Escolha um elemento(x) do seu vetor para ser o pivô, em geral escolhemos o último elemento. Todos elementos < que x colocamos numa posição antes de x e todos >= colocamos depois de x.
- Repare que a posição de x nesse vetor já é a posição final dele no vetor ordenado.
- Chamamos recursivamente o Quick sort para o vetor com os elementos menores que x e para o vetor com os elementos >=x

QuickMerge Sort - Exemplo

Então vamos ordenar o vetor[]={1,6,2,11,5,3,12, **6**}

Então o vetor v1 com o s elementos < 6(pivô) é: {1,2,5,3} e o vetor v2 com os elementos >=6 é {6,11,12}. Repare que não colocamos o pivô em nenhum dos dois vetores.

Ordenando v1 e v2 recursivamente teríamos v1={1,2,3,5} e v2={6,11,12}.

Basta apenas recriar vetor[]=v1+pivô+v2={1,2,3,5,6,6,11,12}

QuickSort - Complexidade

- Para o caso médio do QuickSort é O(nlogn), como no Merge Sort, dado que a esperança de tamanho dos dois vetores criados é n/2. Todavia, no pior caso (no qual pegamos o maior elemento do vetor todas as vezes), devemos realizar n passos, com cada passo custando n. O algoritmo é $O(n^2)$. Um exemplo do pior caso é pegar um vetor já ordenado e o último elemento dele como pivô.

QuickSort - implementação

Esse código abaixo lê um número n, depois lê um vetor de tamanho n e printa ele ordenado:

```
#include<bits/stdc++.h> // declaramos todas as bibliotecas de c++ e c.
using namespace std; // deve ser colocado antes de algumas funções de c++
//basicamente escreve std:: antes de todas linhas do código.

vector < int > quick_sort(vector < int> a){
    if(a.size()==1 || a.size()==0) return a; // caso base, se o vector tiver tamanho 0 ou 1 ele já está ordenado.
    vector < int > a1, a2, res; //declaramos a1(< x), a2(>=x) e res para guardar o vetor a ordenado
    int x=a.size()-1; // a ultima posicao do vetor a.

    for(int i=0; i<a.size()-1; i++){ // colocamos os < a[x] em a1 e os >=a[x] em a2.
        if(a[i]<a[x]) a1.push_back(a[i]);
        else a2.push_back(a[i]);
    }

    a1=quick_sort(a1), a2=quick_sort(a2); //    a1=quick_sort(a1), a2=quick_sort(a2); //ordenamos a1 e a2 recursivamente

    for(int i=0; i<a1.size(); i++) res.push_back(a1[i]); // colocamos os elementos de a1 em res
    res.push_back(a[x]); // colocamos a[x] em res
    for(int i=0; i<a2.size(); i++) res.push_back(a2[i]); //colocamos os elementos de a2 em res.
    return res; //retorne o vetor a ordenado.
}

int main(){
    int n;
    cin >> n;
    vector < int > a(n);
    for(int i=0; i<n; i++) cin >> a[i];
    a=quick_sort(a);
    for(int i=0; i<n; i++) cout << a[i] << " ";
    cout << "\n";
}
```

Exponenciação

Em geral, calcular uma exponencial a^b em b passos é muito lento. Um jeito otimizado de resolver esse problema é reparar que: $a^b = (a^{b/2})^2$ se b for par e $a^b = a*(a^{(b-1)/2})^2$ se b for ímpar.

Exemplo: Se quisermos calcular 2^{501} então podemos seguir o seguinte caminho:

$2^{501}=2*(2^{250})^2$, $2^{250}=(2^{125})^2$, $2^{125}=2*(2^{62})^2$, $2^{62}=(2^{31})^2$, $2^{31}=2*(2^{15})^2$, $2^{15}=2*(2^7)^2$, $2^7=2*(2^3)^2$, $2^3=2*(2^1)^2$, $2^1=2*(2^0)^2$

Repare que fizemos isso em apenas em 9 passos, de fato, esse algoritmo tem complexidade O(logb).

*você reparou em quantos (e em quais) passos multiplicamos por dois? Se percorrermos os passos do mais simples para o mais complicado guardando ‘1’ se multiplicarmos por 2 e ‘0’ se não, teremos: 111110101, que é a representação binária de 501.

Exponenciação

Um bom uso desse método para calcular exponenciais é achar o N-ésimo termo de Fibonacci em complexidade O(logN).

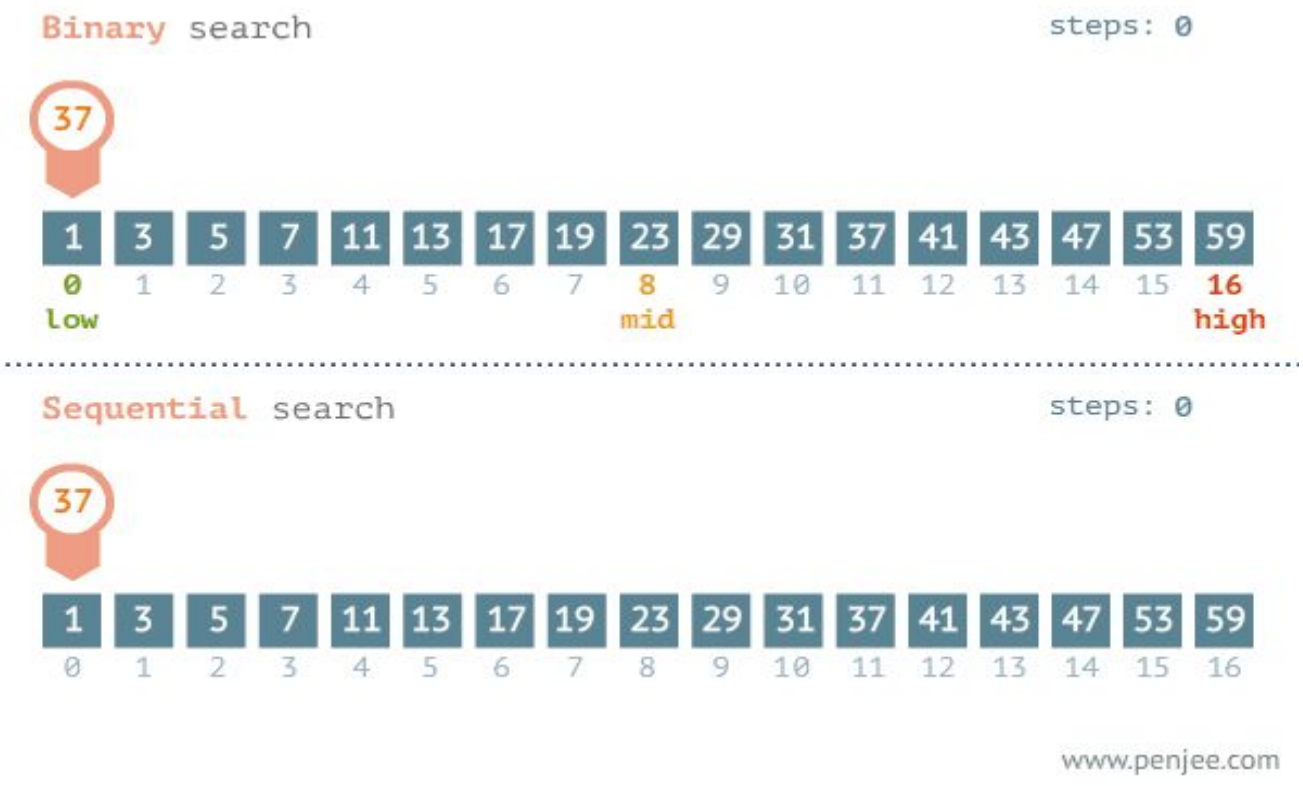
- Como fazer isso? Repare que:

$$\begin{pmatrix} F_{n+2} & F_{n+1} \\ F_{n+1} & F_n \end{pmatrix} = \begin{pmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{pmatrix} * \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$$

Então para calcular F_n basta calcular $\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^n$

- não se preocupe agora com a implementação desse algoritmo.

Busca binária



Busca binária

A busca binária basicamente é: Procure uma página em um livro!

Ela serve para procurar elementos em um vetor ordenado (como as páginas…..) e é feita da seguinte maneira:

- Suponha que queremos procurar o elemento y em um vetor.
- Pegue o elemento (x) do meio do vetor (posição mid=(inicio+fim)/2)
- Se x > y, continuamos a busca no vetor da posição inicio até a posição mid-1.
- Se x <= y, continuamos a busca da posição mid até posicao fim.
- Como em cada passo dividimos o tamanho do vetor por 2, em O(logN) acharemos o elemento y.

Busca Binária- Exemplo

Vamos resolver aqui o problema: Pão a metro, do último contest da bits:

https://www.urionlinejudge.com.br/judge/pt/problems/view/2329

Ideia inicial

Comece com um pão de tamanho 1 e veja quantos pedaços teríamos se o corte fosse de tamanho 1 (divida cada pão por 1). Se corte for >=n vamos para 2, 3, 4...

Fazemos isso até achar o primeiro i que não funciona, então o máximo corte é i-1.

Problema: Cada iteração tem complexidade O(n), e podemos fazer na pior das hipóteses 1e9 operações (tamanho máximo de um pedaço). Teríamos tempo limite excedido.

Busca binária - Exemplo

Solução:

A ideia é: Comece com um pão de tamanho em [inicio,fim]. Pegamos um pão com o tamanho médio mid=(1+(inicio+fim)/2), se o pão de tamanho médio serve para cortar pedaços o suficiente, reduzimos o intervalo para [mid,fim], se não serve reduzimos o intervalo para [inicio,mid-1]. Eventualmente o intervalo terá tamanho 1, e esse será o maior corte possível do pão.

Isso resolverá o problema em O(nlog(1e9)) se pegarmos o intervalo inicial como [1,1e9] que passa dentro do tempo limite.

Busca Binária - Exemplo (implementação)

```cpp
#include<bits/stdc++.h>
using namespace std;
int n, k; // declaramos globalmente as variaveis para usarmos dentro de todas
// funções
int a[10500];

bool funciona(int m){ // funcão que retorna true se o corte de tamanho m
//serve e falso se não.
    int pedacos=0;
    for(int i=1; i<=k; i++){ //loop para calcular quantos pedaços teríamos
        pedacos+=a[i]/m;
    }
    if(pedacos>=n) return true; //se número de pedaçõs for >=n é true.
    else return false;
}

int main(){
    scanf("%d", &n);
    scanf("%d", &k);

    for(int i=1; i<=k; i++) scanf("%d", &a[i]);

    int inicio=1, fim=1e9; //inicialmente sabemos que o tamanho do pao
    //é >=1 e <=10000. Então começamos com [1,1e9]
    while(inicio!=fim){
        int mid=1+(inicio+fim)/2;
        if(funciona(mid)) inicio=mid;
        else fim=mid-1;
    }
    printf("%d\n", inicio);
}
```

Método da bissecção

- Encontrar raízes de uma função
- Imagine o seguinte problema:

  Você comprou um carro com um empréstimo e agora quer pagá-lo em ‘m’ meses com parcelas de ‘d’ reais. Suponha que o valor do carro é ‘v’ e o banco tenha uma taxa de juros de i%. Qual a quantidade d que você tem que pagar por mês?

Método da bissecção

Exemplo:
d= 576.19, m=2, v =1000 and i = 10%

- primeiro mês: 1000x 1.1 - 576.19 = 523.81
- segundo mês: 523.81x 1.1 - 576.19 = 0

Agora, nosso problema é descobrir d dados m,i e v

Método da bissecção

- Temos que escolher um intervalo [a .. b].

  a=0.01 (mínimo pagamento mensal)

  b=1100.00 (pagar o empréstimo em um mês)
- Escolher uma função f:

  f(n)= f(n-1)*i -d com 0<n<m+1

  IMPORTANTE: f(a)*f(b)<0

Método da bissecção

a	**b**	**d** $= \frac{a+b}{2}$	**status:** $f(d, m, v, i)$	**action**
0.01	1100.00	550.005	undershoot by 54.9895	increase d
550.005	1100.00	825.0025	overshoot by 522.50525	decrease d
550.005	825.0025	687.50375	overshoot by 233.757875	decrease d
550.005	687.50375	618.754375	overshoot by 89.384187	decrease d
550.005	618.754375	584.379688	overshoot by 17.197344	decrease d
550.005	584.379688	567.192344	undershoot by 18.896078	increase d
567.192344	584.379688	575.786016	undershoot by 0.849366	increase d
...	...	...	a **few** iterations later ...	...
...	...	576.190476	stop; error is now less than ϵ	answer = 576.19

PROBLEMAS

Problemas - Codeforces

Equivalent Strings - https://codeforces.com/problemset/problem/559/B

Code for 1 - https://codeforces.com/problemset/problem/768/B

Creative Snap - https://codeforces.com/problemset/problem/1111/C

D. Dr. Evil Underscores- https://codeforces.com/problemset/problem/1285/D

- os problemas do Codeforces tem solução no site.

Problema - Maratona SBC de Programação

Problema M - Ideia semelhante ao problema dos pães.

http://maratona.ime.usp.br/hist/2019/primfase19/provas/competicao/maratona.pdf

Problemas do livro Competitive Programming 3

UVa 10567 - Helping Fill Bates
https://uva.onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&category=24&page=show_problem&problem=1508

UVa 11057 - Exact Sum
https://uva.onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&category=24&page=show_problem&problem=1998

UVa 12192 - Grapevine
https://uva.onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&category=24&page=show_problem&problem=3344