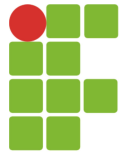
Aplicações da Busca Binária

Tópicos Especiais em Algoritmos

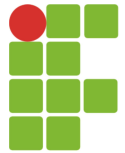
Prof. Daniel Saad Nogueira Nunes

IFB – Instituto Federal de Brasília,
Campus Taguatinga



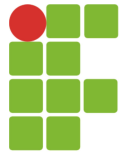
Sumário

1. Introdução
2. Busca binária
3. Método da bissecção
4. Analisando a resposta



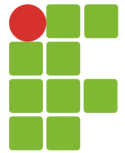
Sumário

1 Introdução



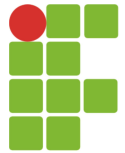
Divisão e conquista

- O paradigma de divisão e conquista para projetar algoritmos tem como fundamento dividir o problema em partes menores.
- As partes menores são resolvidas (conquistadas).
- Se necessário as partes conquistadas são combinadas para compor a solução completa do problema original.



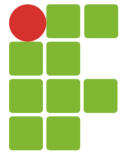
Divisão e conquista

- Exemplos de métodos e EDs baseados neste paradigma:
  - Quicksort.
  - Mergesort.
  - Binary Search Trees.
  - Segment Trees.
  - Fenwick Trees.
  - ...



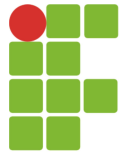
Divisão e conquista

- A busca binária (Binary Search) é um método de busca bem conhecido inspirado no paradigma de divisão e conquista.
- Apesar de ser um método ensinado no primeiro ou segundo semestre de um curso de Computação, poucos sabem utilizá-lo de maneiras menos óbvias.
- Veremos como utilizar a busca binária de formas pouco ortodoxas.



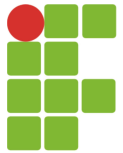
Sumário

2 Busca binária



Busca binária

- A maneira mais óbvia de utilizar a busca binária é sobre uma coleção de elementos ordenada e estática.
- Checamos o elemento do meio:
  - Se o elemento do meio corresponde àquilo que estamos buscando, paramos a busca.
  - Se o elemento do meio é maior do que aquilo que estamos buscando, descartamos a metade superior.
  - Se o elemento do meio é menor do que aquilo que estamos buscando, descartamos a metade inferior.
- $\Theta(\lg n)$ passos.

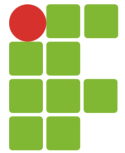


Busca Binária

Algorithm 1: BSEARCH(V, k)

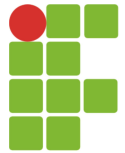
Input: $V[0, n-1]$,k
Output: j, se $k = V[j]$ e -1 caso não exista tal j.

```
1  l ← 0
2  r ← n − 1
3  while l ≤ r do
4      mid ← l + ⌊(r−l)/2⌋
5      if( k = V[mid] )
6          return mid
7      else if( k < V[mid] )
8          r ← mid − 1
9      else
10         l ← mid + 1
11 return −1
```



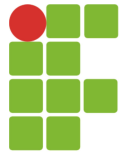
Busca binária

- A busca binária usual já encontra-se implementada nas bibliotecas das linguagens de programação.
- Ex: `algorithm::lowerbound` no C++.
- Ex: `Collections.binarySearch` no Java.
- No entanto, podemos aplicar a busca binária sem que a entrada seja um vetor de elementos, mas em outras estruturas de dados.



Sumário

2 Busca binária
- Busca binária em outras EDs



Busca binária em outra EDs

- Suponha o seguinte problema (Thailand ICPC National Contest 2009).
- Entradas:
  - Uma árvore com binária com pesos inteiros nos nós com até $N \leq 8 \cdot 10^4$ vértices. Qualquer caminho da raiz até uma folha tem pesos crescentes. A árvore não precisa estar balanceada.
  - Q ($Q \leq 2 \cdot 10^4$) consultas sobre vértices $v_1, \ldots . v_Q$ e inteiros $p_1, \ldots, p_Q$.
- Saída: o ancestral de v_i mais próximo a raiz com peso maior ou igual a p_i.

Busca binária em outras EDs

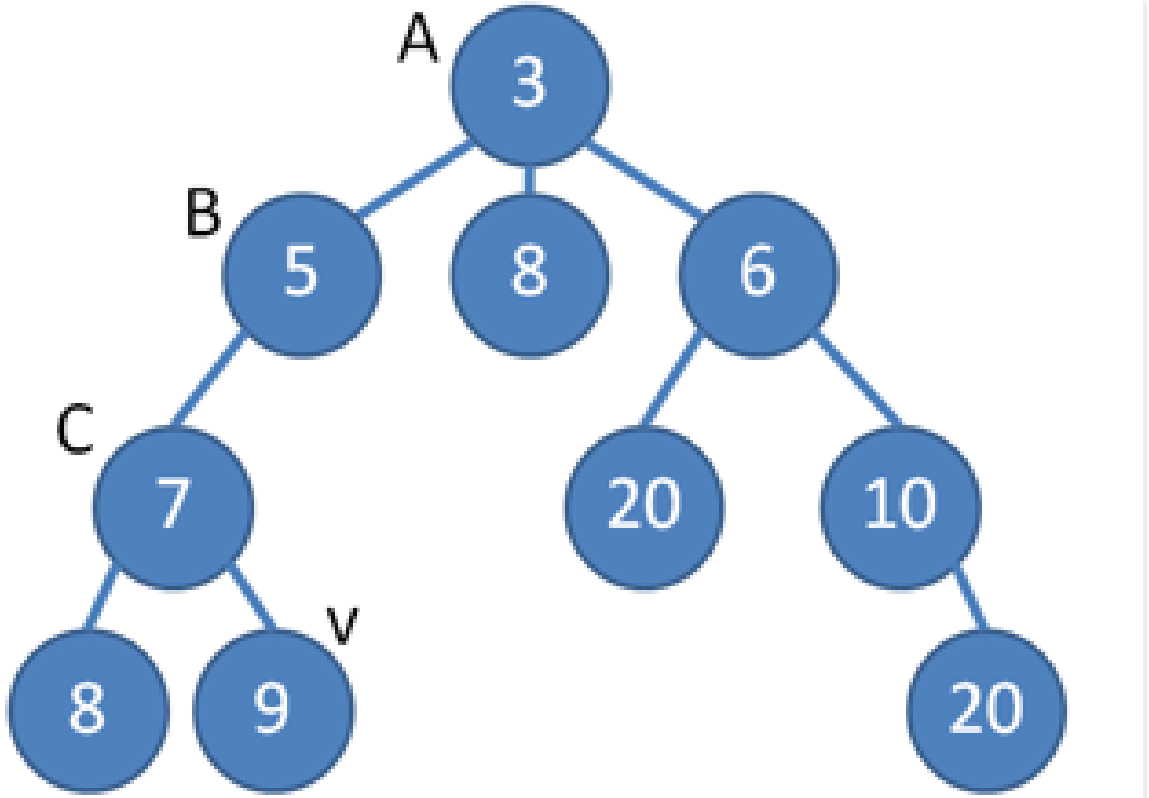


Figura: O que acontece se $p = 4$?

Busca binária em outras EDs

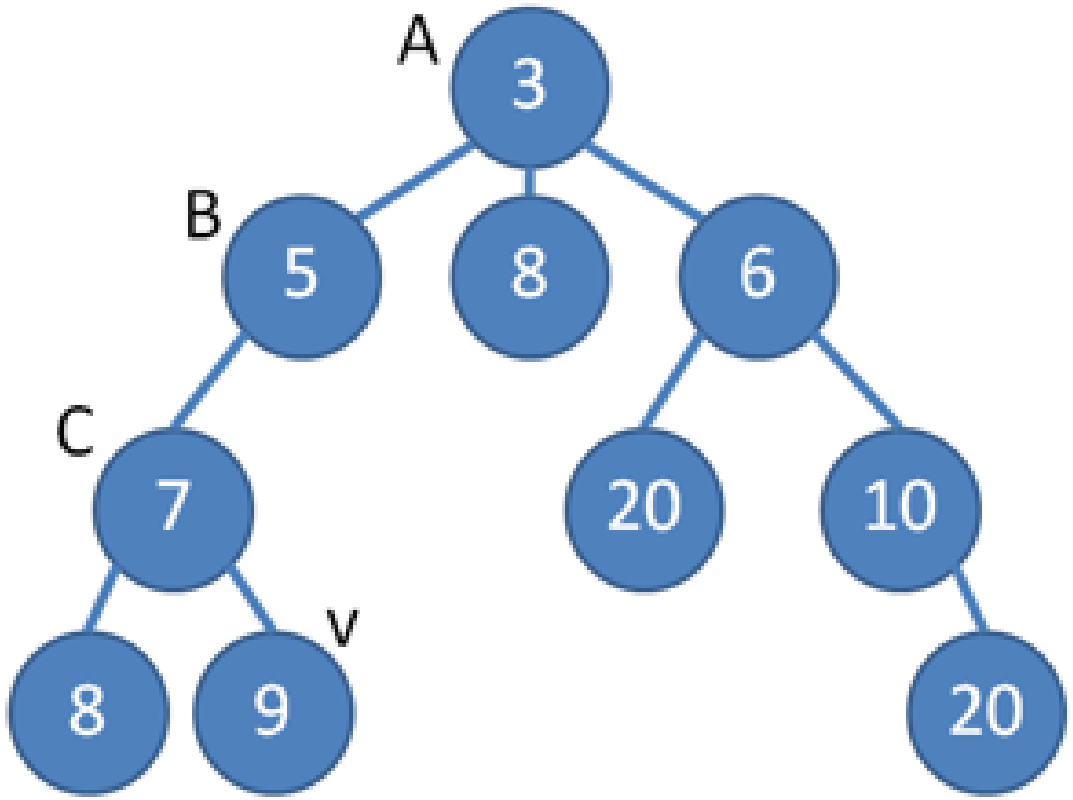


Figura: O que acontece se $p = 7$?

Busca binária em outras EDs

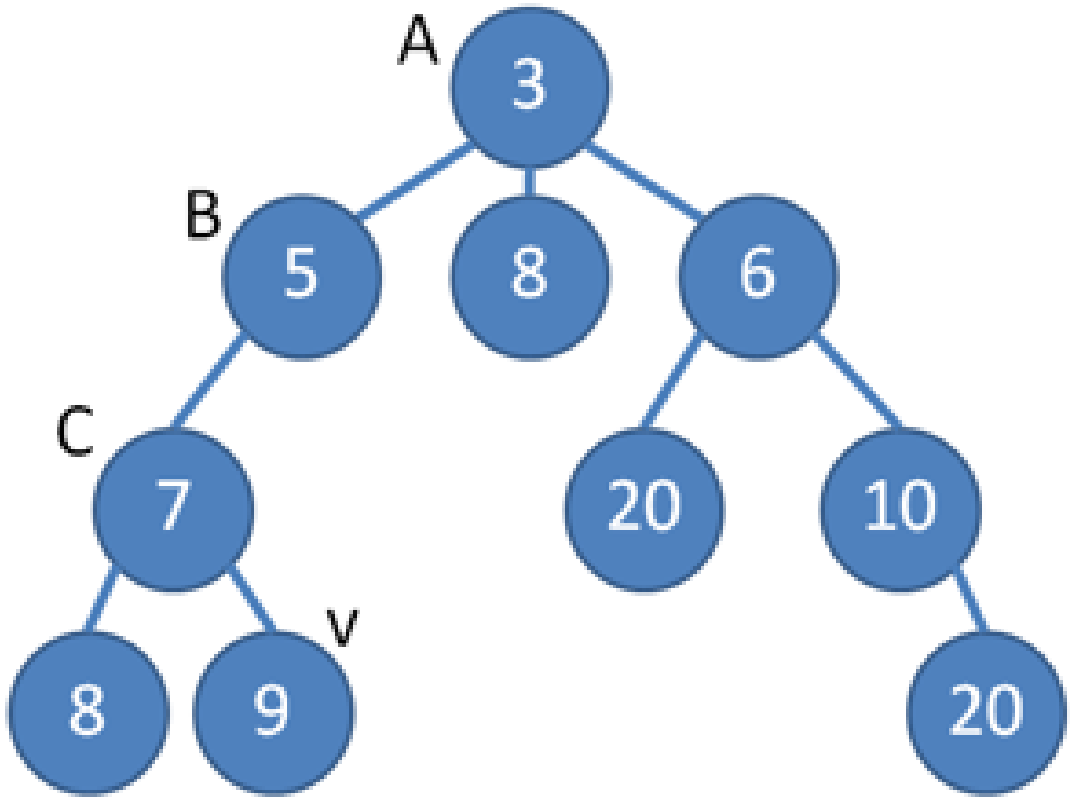
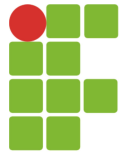


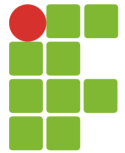
Figura: O que acontece se $p = 9$?



Busca binária em outras EDs

Abordagem força-bruta

- Para cada vértice v_i percorrer em direção a raiz até encontrar o vértice mais próximo da raiz com peso $\geq p_i$
- Como a árvore pode não ser balanceada temos tempo $O(N)$ por consulta.
- Tempo total $O(NQ)$.
- Resultado: TLE.

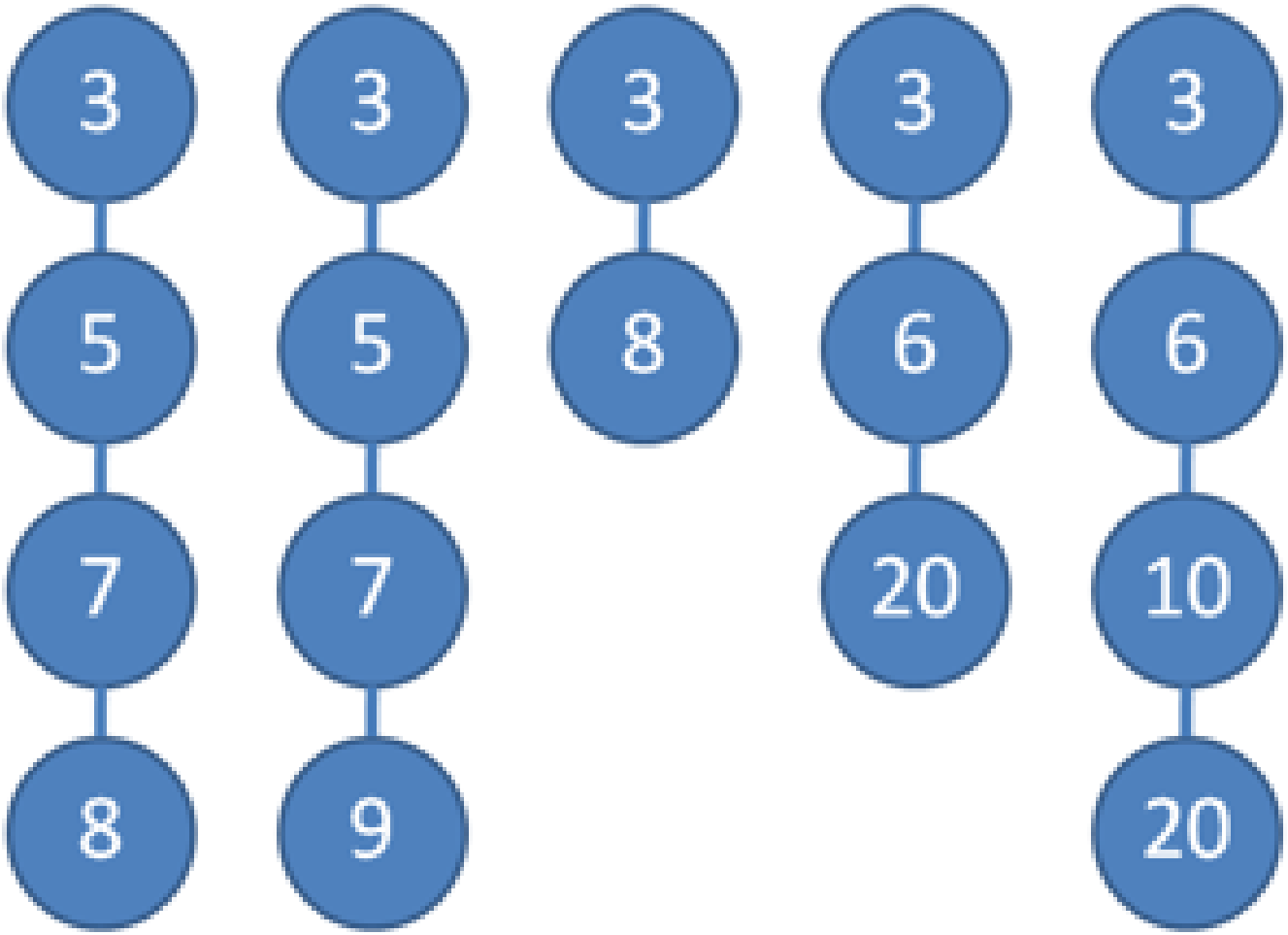


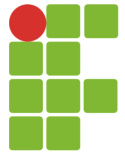
Busca binária em outras EDs

Solução aceitável

- Armazenar para cada caminho da raiz até uma folha um vetor. Gere todos os caminhos uma única vez.
- Este vetor estará em ordem crescente de acordo com a propriedade da entrada.
- Varra todas as consultas e marque os vértices que serão consultados.
- Para cada consulta, efetue uma busca binária para encontrar o ancestral mais próximo da raiz

Busca binária em outras EDs

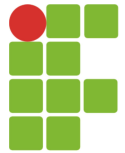




Busca binária em outras EDs

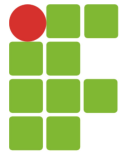
Solução aceitável

- Q consultas com tempo $\Theta(\lg N)$ em cada uma.
- $\Theta(Q \lg N)$.
- Resultado: AC.



Sumário

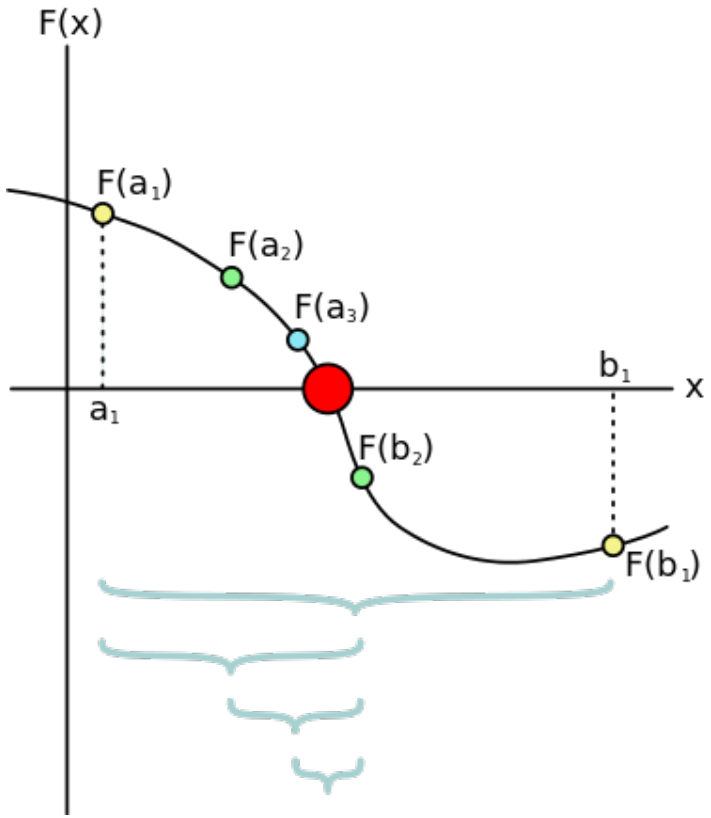
3 Método da bissecção

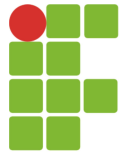


Método da bissecção

- A busca binária também pode ser encontrada no método da bissecção.
- Este método serve para encontrar raízes de uma função.

Método da bissecção

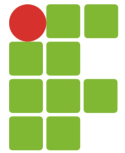




Método da bissecção

Exemplo

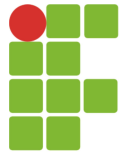
- Suponha que você quer comprar um carro através de um financiamento e que o valor da prestação por m meses é de d reais.
- O valor original do carro é v e o banco te cobra uma taxa de $i\%$ ao mês sobre o valor da quantia não paga.
- Qual o valor d que você deve pagar ao mês?



Método da bissecção

Exemplo

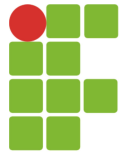
- Suponha que $d = 576.19$, $v = 1000$, $m = 2$ e $i = 10\%$.
- No primeiro mês, seu débito passa a ser $1000 \cdot 1.1 - 576.19 = 523.81$.
- No segundo mês, seu débito passa a ser $523.81 \cdot 1.1 - 576.19 \approx 0$.
- Como determinar que o valor $d = 576.19$?



Método da bissecção

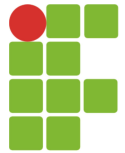
Exemplo

- No fim das contas queremos saber a raiz d tal que $f(d, m, v, i) \approx 0$.
- É a famosa tabela Price `https://pt.wikipedia.org/wiki/Tabela_Price`.



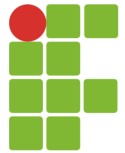
Método da bissecção

- Como resolver utilizando o método da bissecção?
- Primeiro temos que estabelecer um intervalo $[a, b]$ que contém a raiz da função f.
- Para o método da bisseção funcionar, é necessário que $f(a, m, v, i)$ e $f(b, m, v, i)$ tenham sinais opostos.
- Podemos escolher $a = 0.01$ (pagamos no mínimo 1 centavo por prestação) e $b = v \cdot (1 + i\%)$ (a maior prestação possível é aquela que pode ser quitada em 1 mês).



Método da bissecção

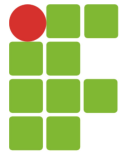
a	**b**	**d** $= \frac{a+b}{2}$	**status:** $f(d, m, v, i)$	**action**
0.01	1100.00	550.005	undershoot by 54.9895	increase d
550.005	1100.00	825.0025	overshoot by 522.50525	decrease d
550.005	825.0025	687.50375	overshoot by 233.757875	decrease d
550.005	687.50375	618.754375	overshoot by 89.384187	decrease 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



Método da bissecção

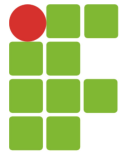
Algorithm 2: bissection(f, a, b)

```
1 i ← 0
2 while i < IT_MAX do
3 |   c ← a + ⌊(b−a)/2⌋
4 |   if( SIGN(f(c)) = SIGN(f(a)) )
5 |   |   a ← c
6 |   else
7 |   |   b ← c
8 |   i++
9 return c
```



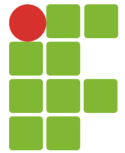
Método da bissecção

- Complexidade: $O(\frac{\lg(b-a)}{\epsilon})$.
- No exemplo, o método da bissecção leva $\lg \frac{1099.99}{\epsilon}$ tentativas.
- Se $\epsilon = 1e-9$, temos ≈ 40 iterações.
- Se $\epsilon = 1e-15$, temos ≈ 60 iterações.
- Também podemos parar a busca assim que $f(c) < \epsilon$.



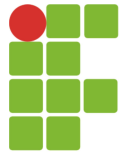
Sumário

4 Analisando a resposta



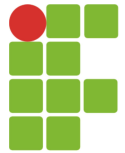
Busca Binária sobre a Resposta

- Peguemos o problema UVa 11935.
- Basicamente temos um jipe com tanque cheio inicialmente, cuja capacidade é conhecida, que faz 100km com n litros.
- Durante esta jornada, vários eventos podem ocorrer:
  - Consumo de combustível: a taxa 100km/litro é informada no trecho a ser percorrida (pode variar dependendo do trecho).
  - Posto de combustível: enche o tanque.
  - Vazamento: adiciona ao consumo atual 1 litro a cada 100km. Múltiplos vazamentos acumulam.
  - Mecânico: conserta todos os vazamentos.
  - Objetivo: chegou ao final da jornada.



Busca Binária sobre a Resposta

- Da descrição do problema, temos que dar como resposta a capacidade mínima do tanque para que o jipe complete a viagem.
- Esta capacidade está em $[0.000, 10000.000]$.
- 10^9 possibilidades.
- Simular em todas elas: TLE.
- Solução: simular algumas apenas. Aquelas dadas pela busca binária.



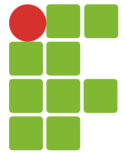
Busca Binária sobre a Resposta

Algorithm 3: BSEARCH-ANSWER(a, b)

Input: a e b com $[a, b]$ sendo o intervalo de possíveis respostas.

Output: Menor capacidade do tanque para completar a viagem.

```
1  l ← a
2  r ← b
3  while |l − r| > ε do
4      m ← l + ⌊(r−l)/2⌋
5      if( SIMULATE(m) )
6          ans ← m
7          r ← m
8      else
9          l ← m
10 return ans
```



Analisando a resposta

- Mais uma vez a condição de parada poderia ser substituída pelo número de iterações.