Resolução dos Exercícios

Exercícios C, D, E e G

Exercício C - a-Good String

Objetivo -> Contar quantas modificações precisam ser feitas para transformar a string dada em uma a-Good String.

(É garantido que o tamanho da string é uma potência de 2)

O que é uma a-Good String ?

aaaabbcd

aaaacdbb

bbcdaaaa

cdbbaaaa

Exercício C - a-Good String

Como fazer essas strings?

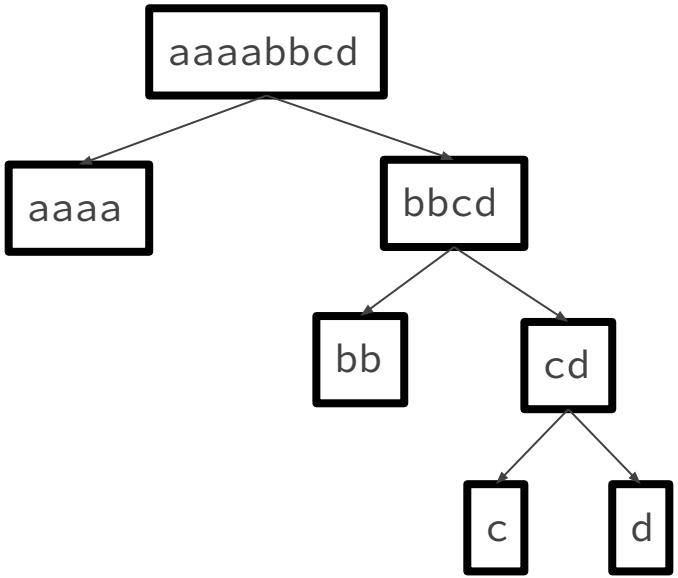
O problema fica mais fácil se eu dividir a string ao meio ?

Ex: aaaabbcd -> aaaa bbcd

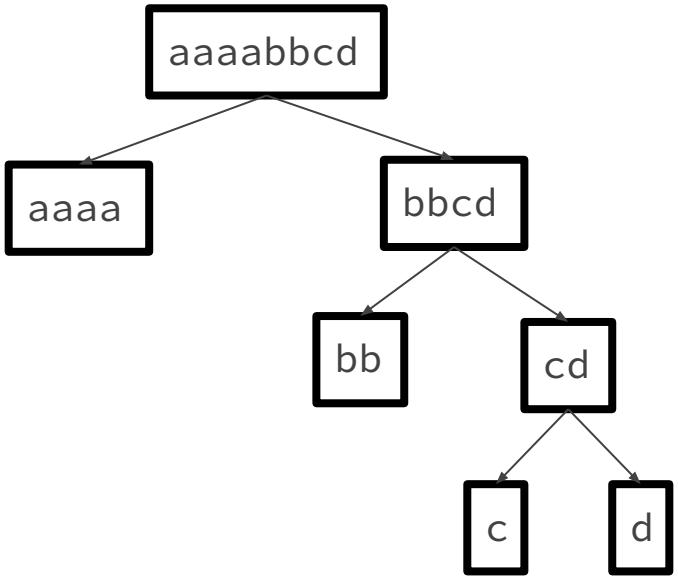
A string bbcd seria uma b-Good_String

Isto é, podemos dividir a string bbcd ao meio também :)

Exercício C - a-Good String

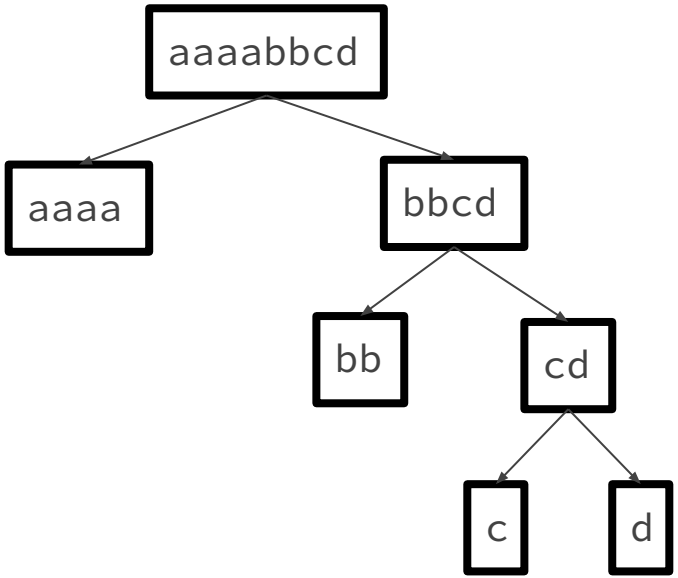


Exercício C - a-Good String



Sempre que dividirmos a string ao meio, temos uma good string da próxima letra em uma das metades :)

Exercício C - a-Good String



Ideia: Comparar a string do exercício com a string gerada pelo meu algoritmo

Exercício C - a-Good String

```
ll merge(string &s, char a, ll ini, ll fim){
    if(ini == fim){
        if(s[ini] != a)return 1;
        else return 0;
    }
    ll meio = (ini + fim)/2;
    ll res = 1e17;
    ll changes = 0;

    for(int i = ini; i <= meio; i++){
        if(s[i] != a)changes++;
    }

    changes += merge(s, a+1, meio+1, fim);
    res = min(res, changes);
    changes = 0;

    for(int i = meio+1; i <= fim; i++){
        if(s[i] != a)changes++;
    }
    changes += merge(s, a+1, ini, meio);
    res = min(res, changes);

    return res;

}
```

Exercício C - a-Good String

```
ll merge(string &s, char a, ll ini, ll fim){
    if(ini == fim){
        if(s[ini] != a)return 1;
        else return 0;
    }
    ll meio = (ini + fim)/2;
    ll res = 1e17;
    ll changes = 0;

    for(int i = ini; i <= meio; i++){
        if(s[i] != a)changes++;
    }

    changes += merge(s, a+1, meio+1, fim);
    res = min(res, changes);
    changes = 0;
```


```
for(int i = meio+1; i <= fim; i++){
    (s[i] != a)changes++;

s += merge(s, a+1, ini, meio);
nin(res, changes);

 res;
}
```

Exercício D - Expedition

- Um grupo de vaquinhas têm um caminhãozinho com o tanque furado, e este perde um litro de gasolina a cada metro de distância que ele anda.

- O objetivo é chegar até o ponto final para consertar o caminhão, abastecendo o mínimo número de vezes possíveis.

- São dadas a distância de todos os postos para a última cidade e quantidade de combustível que cada poço possui.

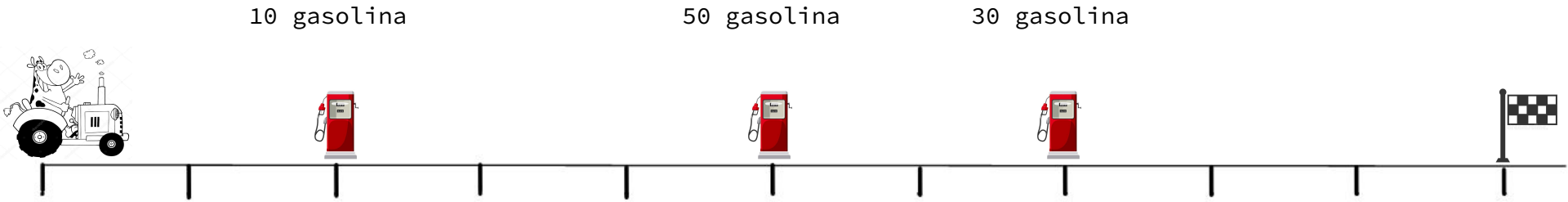
Exercício D - Expedition

- O objetivo é abastecer o mínimo de vezes possível, ou seja colocar combustível só quando precisar e do maior posto possível até o momento.

- Quando o combustível acabar precisamos olhar de todos os postos que já passamos, aquele que tem a maior quantidade de combustível para colocar.

Exercício D - Expedition

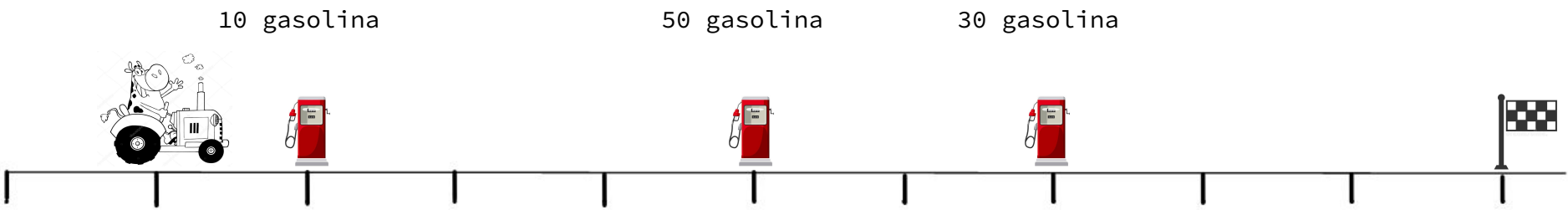
Combustível -> 80



Priority-Queue -> []

Exercício D - Expedition

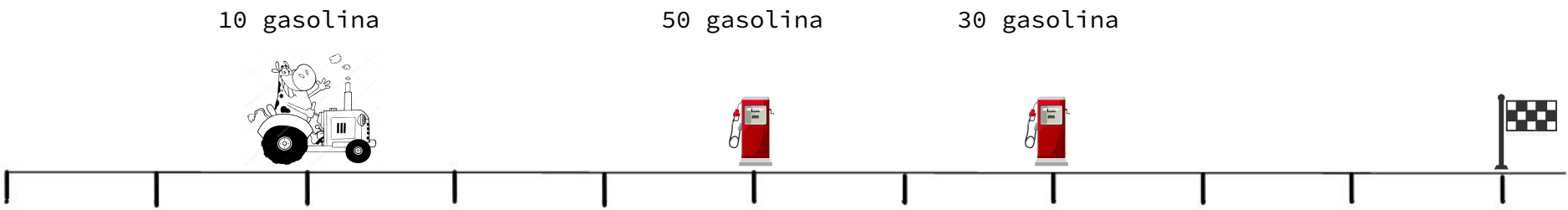
Combustível -> 70



Priority-Queue -> []

Exercício D - Expedition

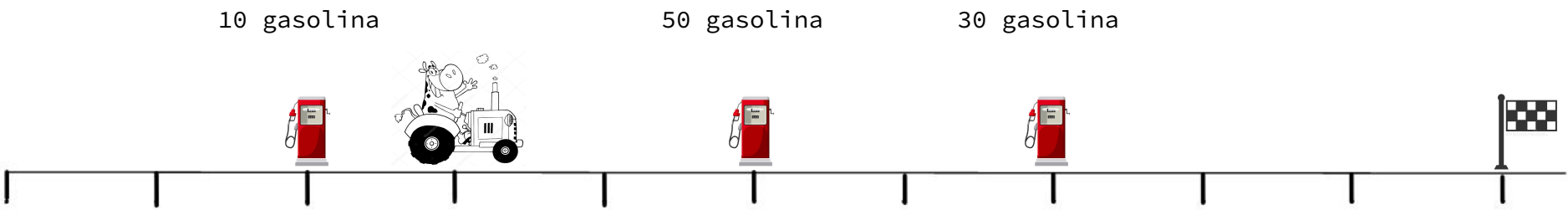
Combustível -> 60



Priority-Queue -> [10]

Exercício D - Expedition

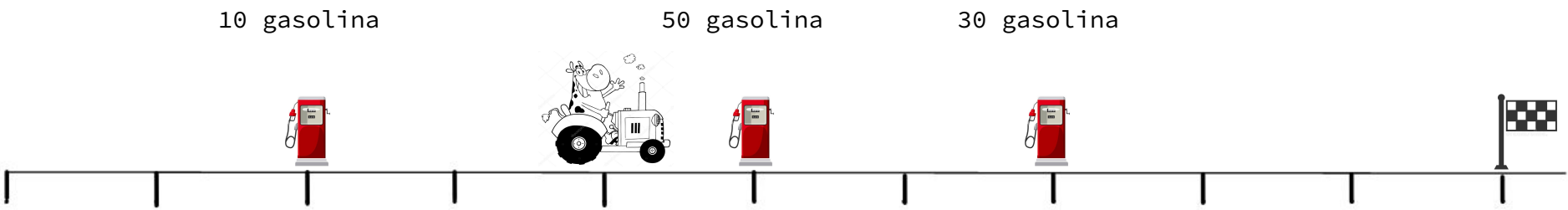
Combustível -> 50



Priority-Queue -> [10]

Exercício D - Expedition

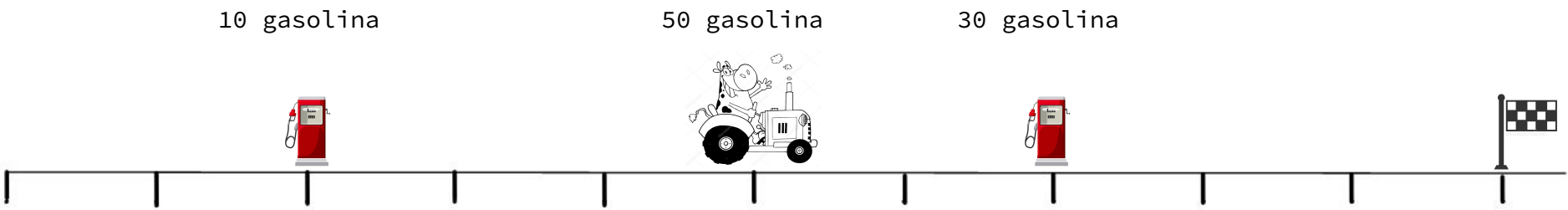
Combustível -> 40



Priority-Queue -> [10]

Exercício D - Expedition

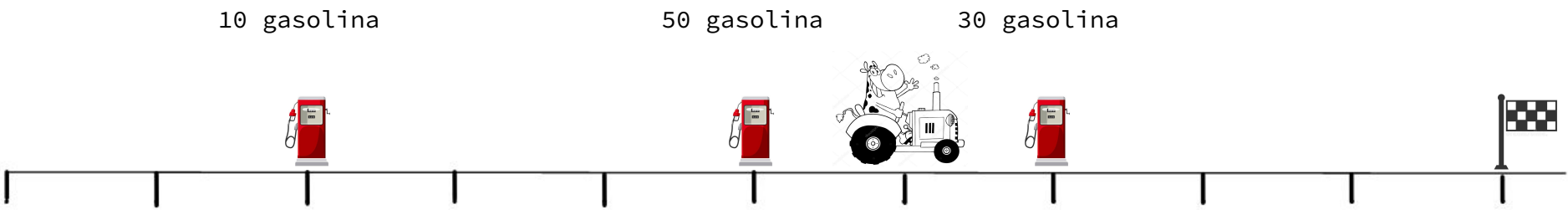
Combustível -> 30



Priority-Queue -> [50, 10]

Exercício D - Expedition

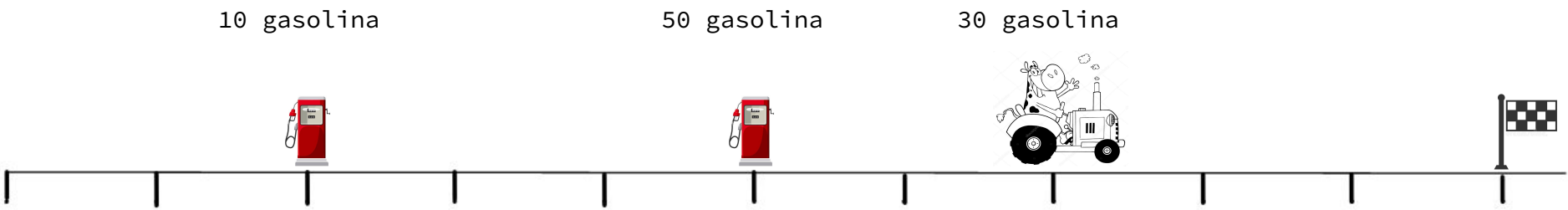
Combustível -> 20



Priority-Queue -> [50, 10]

Exercício D - Expedition

Combustível -> 10



Priority-Queue -> [50, 30, 10]

Exercício D - Expedition

Combustível -> 0

Se a gasolina acabar, vamos abastecer no posto que tinha mais gasolina que já passamos

Priority-Queue -> [50, 30, 10]

Exercício D - Expedition

Combustível -> 50

Se a gasolina acabar, vamos abastecer no posto que tinha mais gasolina que já passamos

Priority-Queue -> [30, 10]

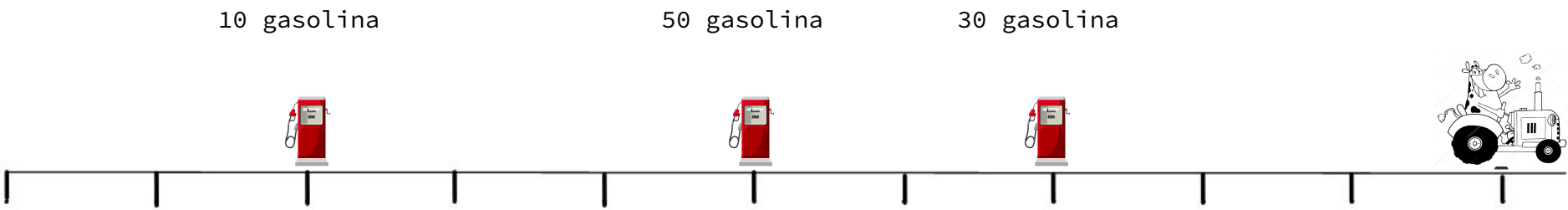
Exercício D - Expedition

Combustível -> 40

Priority-Queue -> [30, 10]

Exercício D - Expedition

Combustível -> 30



Priority-Queue -> [30, 10]

Exercício D - Expedition

Resposta = 1

Combustível -> 30

Priority-Queue -> [30, 10]

Exercício D - Expedition

```cpp
int main(){
    fast_io;
    ll t;
    cin >> t;
    while(t--){
        ll n, L, P; cin >> n;
        ll pos, fuel;
        vector<pair<ll,ll>> v;
        for(int i = 0; i < n; i++){
            cin >> pos >> fuel;
            v.push_back({pos, fuel});
        }
        priority_queue<ll> p;
        cin >> L >> P;
        for(int i = 0; i < n; i++){
            v[i].first = L - v[i].first;
        }
        sort(v.begin(), v.end());
        ll cont = 0;
        bool flag = true;
        for(int i = 0; i < n; i++){
            if(v[i].first <= P){
                p.push(v[i].second);
            }
            else{
                while(!p.empty() && P < v[i].first){
                    P += p.top();
                    p.pop();
                    cont++;
                }
                if(P < v[i].first){
                    flag = false;
                    break;
                }
                p.push(v[i].second);
            }
            if(P >= L)break;
        }
        while(P < L && !p.empty()){
            P += p.top();
            p.pop();
            cont++;
        }
        if(P<L || !flag)cout << "-1\n";
        else cout << cont << "\n";
    }
```

Exercício D - Expedition


```cpp
int main(){
    fast_io;
    ll t;
    cin >> t;
    while(t--){
        ll n, L, P; cin >> n;
        ll pos, fuel;
        vector<pair<ll,ll>> v;
        for(int i = 0; i < n; i++){
            cin >> pos >> fuel;
            v.push_back({pos, fuel});
        }
        priority_queue<ll> p;
        cin >> L >> P;
        for(int i = 0; i < n; i++){
            v[i].first = L - v[i].first;
        }
        sort(v.begin(), v.end());
        ll cont = 0;
        bool flag = true;
        for(int i = 0; i < n; i++){
            if(v[i].first <= P){
                p.push(v[i].second);
            }
            else{
                while(!p.empty() && P < v[i].first){
                    P += p.top();
                    p.pop();
                    cont++;
                }
                if(P < v[i].first){
                    flag = false;
                    break;
                }
                p.push(v[i].second);
            }
            if(P >= L)break;
        }
        while(P < L && !p.empty()){
            P += p.top();
            p.pop();
            cont++;
        }
        if(P<L || !flag)cout << "-1\n";
        else cout << cont << "\n";
    }
```

Exercício E - Collection Coins

- É dado uma quantidade de moedas n, tal que n >= 0 e n <= 10^4
- O número de casos de testes pode ir até 100

  Portanto, precisamos pensar em algo que funcione em tempo linear para esse exercício

- O rei pode pegar qualquer soma em valor do banco.
- O banco sempre irá pagar o valor pela maior moeda disponível enquanto, este for maior que ela.

Exercício E - Collection Coins

Ex:

```
Moedas -> 1 5 7 8 15 18 40

Valor que o rei quer -> 82

O banco irá usar duas moedas de 40 e duas de 2

40 + 40 + 1 + 1 = 82
```

Exercício E - Collection Coins

- Queremos maximizar o número de coins diferentes que é usada para fazer uma soma qualquer.

  Moedas -> 1 5 7 8 15 18 40

Exercício E - Collection Coins

```
Moedas -> 1 5 7 8 15 18 40
          1


Moedas utilizadas -> {1}
```

Exercício E - Collection Coins

Moedas -> 1 5 7 8 15 18 40

1 6

A soma 6 é maior que 7 ?

Não, então podemos usar a moeda 1 e 5.

Moedas utilizadas -> {1, 5}

Exercício E - Collection Coins

Moedas -> 1 5 7 8 15 18 40

1 6 13

A soma 13 é maior que 8 ?

Sim, se usarmos a moeda 7 a soma 13 irá ser feita pelo banco com apenas duas moedas: 5 + 8 = 13

Moedas utilizadas -> {1, 5}

Exercício E - Collection Coins

```
Moedas -> 1 5 7  8 15 18 40
          1 6   14
```

A soma 14 é maior que 15?

Não, então usamos a moeda 8

Moedas utilizadas -> {1, 5, 8}

Exercício E - Collection Coins

Moedas -> 1 5 7 8 15 18 40

1 6 14 29

A soma 29 é maior que 18?

Sim, então não usamos a moeda 15, pois a soma 29 pode ser gerada com 3 tipos de moedas.

18 + 8 + 1 + 1 + 1 = 29

Moedas utilizadas -> {1, 5, 8}

Exercício E - Collection Coins

Moedas -> 1 5 7 8 15 18 40

1 6 14 32

A soma 32 é maior que 18?

Não, então podemos usar a moeda 18 na nossa soma.

Moedas utilizadas -> {1, 5, 8, 18}

Exercício E - Collection Coins

Moedas -> 1 5 7 8 15 18 40

1 6 14 32 72

A última moeda sempre pode ser usada, porque não tem nenhuma moeda de valor maior que ela.

Resposta -> 5 moedas diferentes

Moedas utilizadas -> {1, 5, 8, 18, 40}

Exercício E - Collection Coins

```
while(t--){
        ll n; cin >> n;
        vector<ll> v(n);
        for(int i = 0; i < n; i++){
            cin >> v[i];
        }
        cout << "Case #" << cas++ << ": ";

        if(n == 0){
            cout << "0\n";
            continue;
        }
        sort(v.begin(), v.end());
```

```
ll cont = 0, val = 0;
    for(int i = 0; i < n-1; i++){
        if(val + v[i] < v[i+1]){
            val += v[i];
            cont++;
        }
    }

    cout << cont + 1 << "\n";
}
```

Exercício E - Collection Coins

```
while(t--){
        ll n; cin >> n;
        vector<ll> v(n);
        for(int i = 0; i < n; i++)
            cin >> v[i];
        }
        cout << "Case #" << cas++

        if(n == 0){
            cout << "0\n";
            continue;
        }
        sort(v.begin(), v.end());
```


```
ll cont = 0, val = 0;
    for(int i = 0; i < n-1; i++){
        if(val + v[i] < v[i+1]){
            val += v[i];
            cont++;
        }


    cout << cont + 1 << "\n";
}
```

Exercício G - Code for 1

- Jon Snow e Sam, após uma árdua batalha contra os white-walkers, retornam para o castelo negro;

- Quando chegam, Sam decide viajar para a Cidadela, com o intuito de se tornar um maester;

- Para testar sua aptidão, eles recebem um problema, que deve ser resolvido para atingirem seu objetivo.

Exercício G - Code for 1

- Inicialmente, temos umas lista composta por **um único elemento** de **valor N**;

- Enquanto a lista não for composta apenas por **0’s** e **1’s**, eles escolhem um elemento da lista;

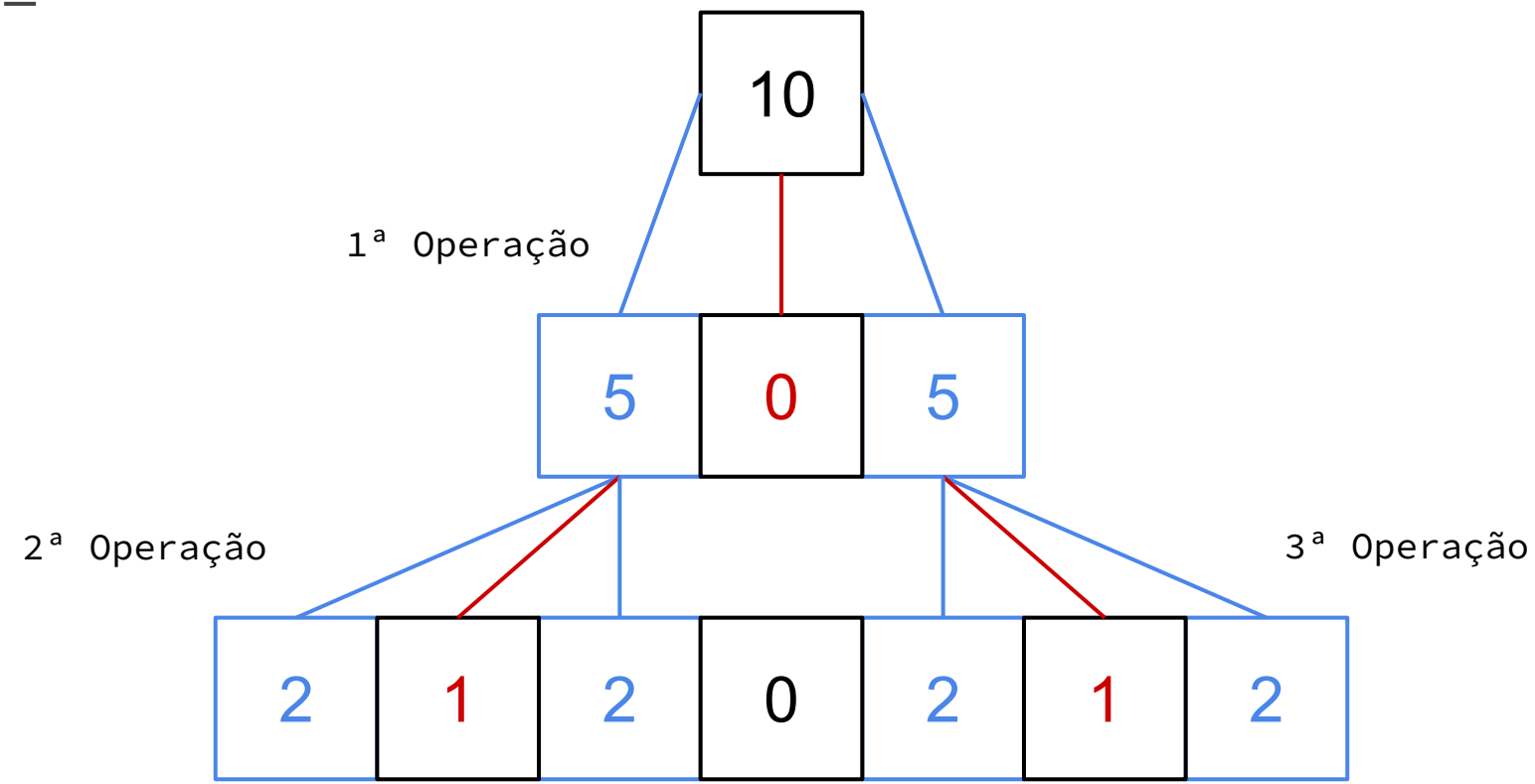
- **Inserimos** na lista, à **direita** e à **esquerda** do elemento escolhido, os resultados de sua **divisão por 2** e **substituímos o elemento** pelo **resto de sua divisão por 2.**

Exercício G - Code for 1

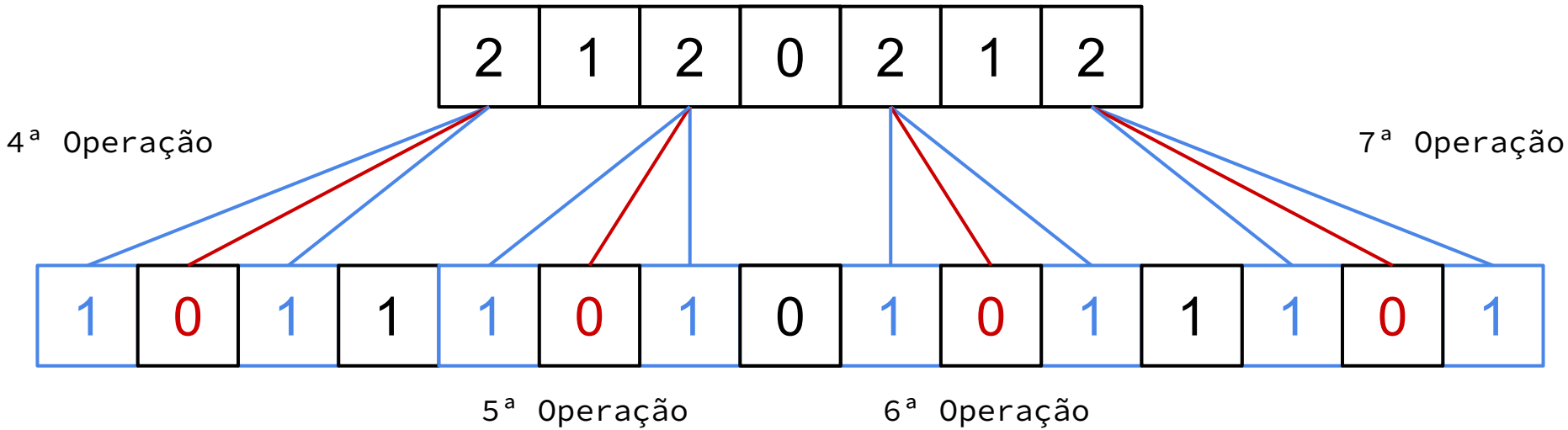
- Para dificultar a tarefa e torná-la um verdadeiro desafio capaz de decidir a dignidade de Sam em se tornar maester, uma última etapa é adicionada;

- Após montada a lista, dado um **intervalo [L : R]**, sendo **L** seu **limite inferior** e **R** o **limite superior**, os maesters querem saber a quantidade de **1's** contidos dentro dele.

Exercício G - Code for 1



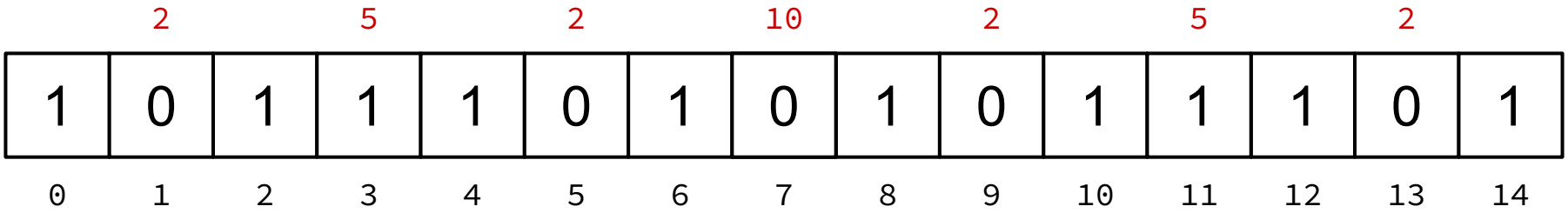
Exercício G - Code for 1



Exercício G - Code for 1

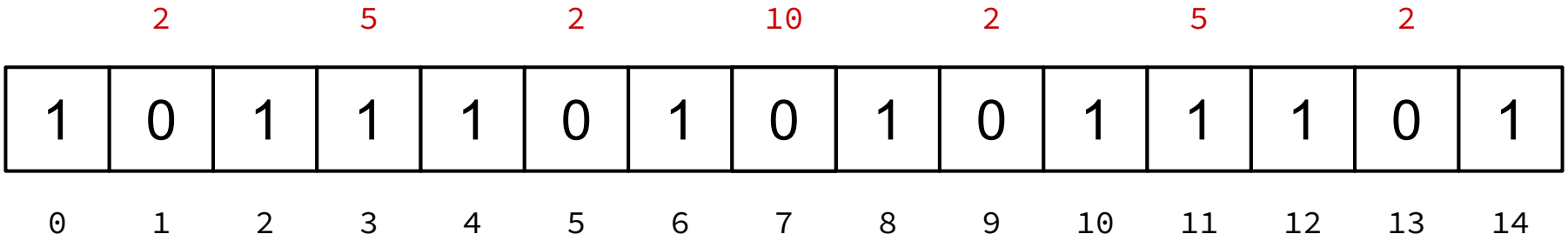
- Ao final do processo, podemos perceber que a quantidade de elementos em nossa lista é equivalente a $2^{log2(N)} - 1$, com log2(N) arredondado para cima;

- Ainda, indexando nosso vetor, percebemos que, dado um intervalo, podemos saber a posição do elemento central nele.



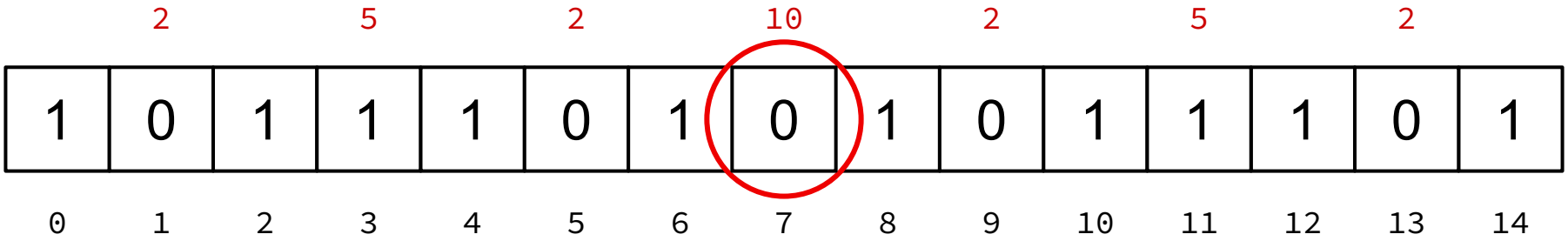
Exercício G - Code for 1

- Elemento na posição: **7**



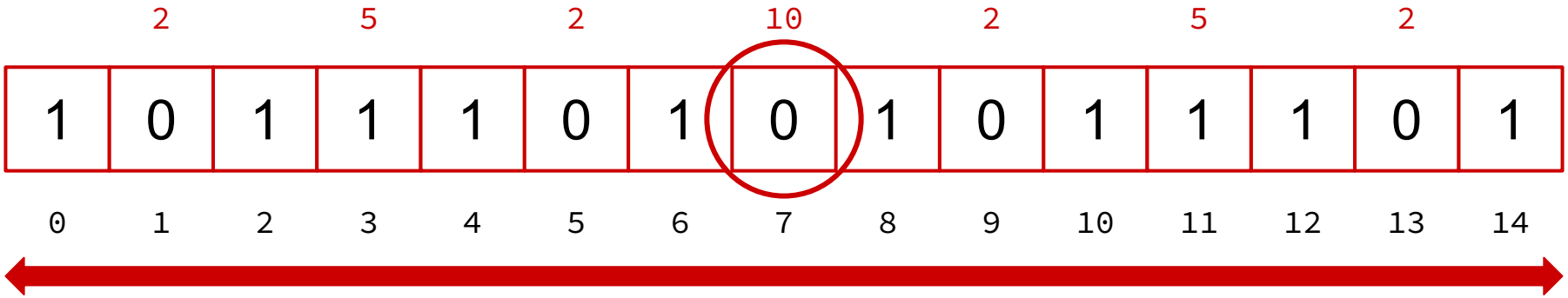
Exercício G - Code for 1

- Elemento na posição: **7**



Exercício G - Code for 1

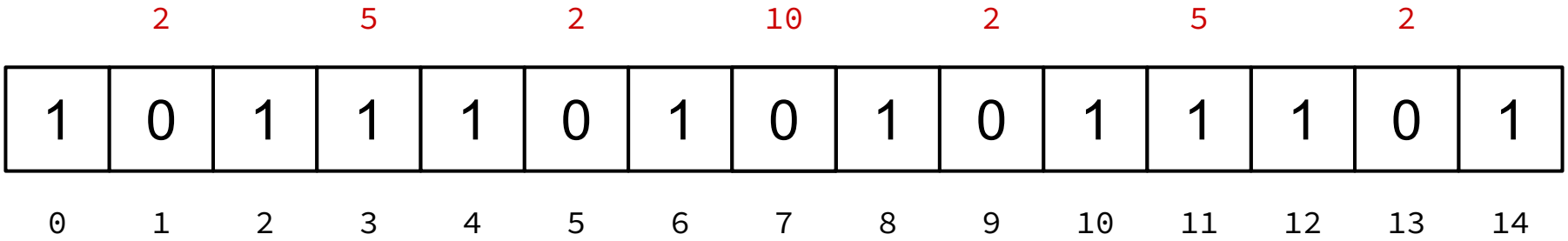
- Elemento na posição: **7**



- Elemento central do intervalo [0 : 14]
  - (0 + 14) / 2 = 7

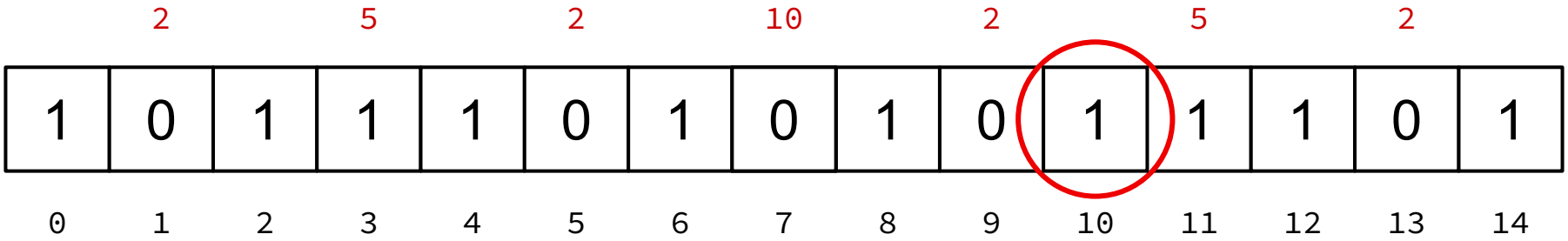
Exercício G - Code for 1

- Elemento na posição: **10**



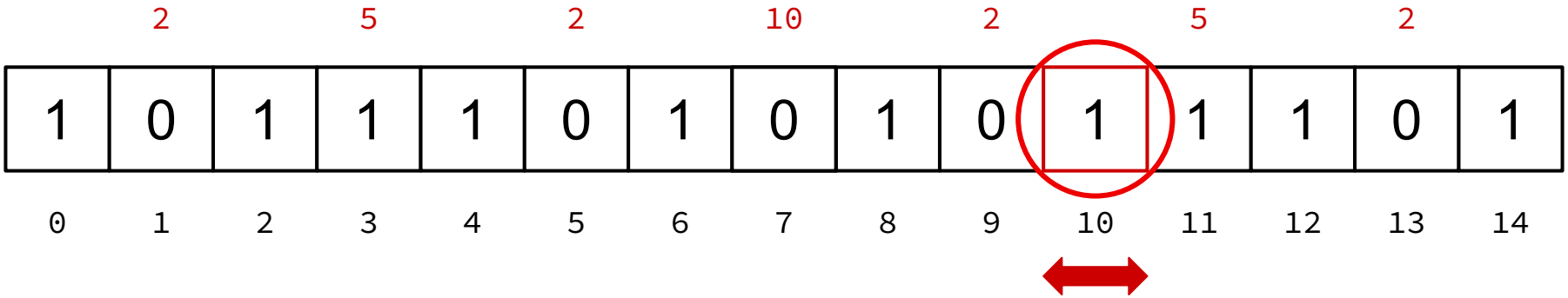
Exercício G - Code for 1

- Elemento na posição: **10**



Exercício G - Code for 1

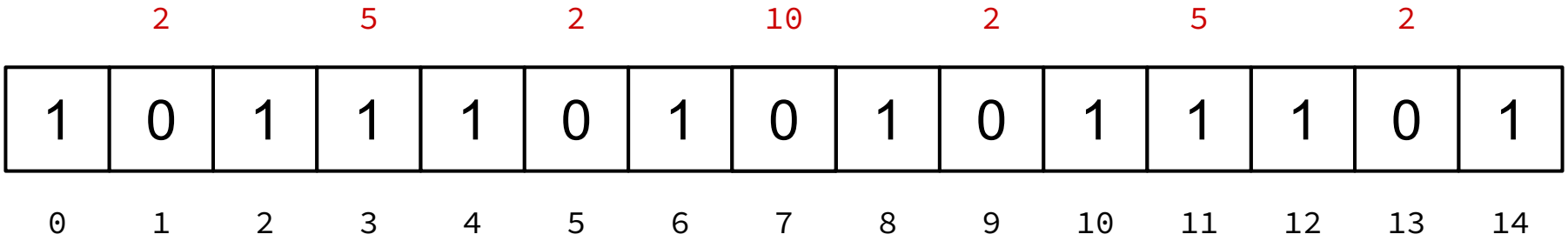
- Elemento na posição: **10**



- Elemento central do intervalo [10 : 10]
  - (10 + 10) / 2 = 10

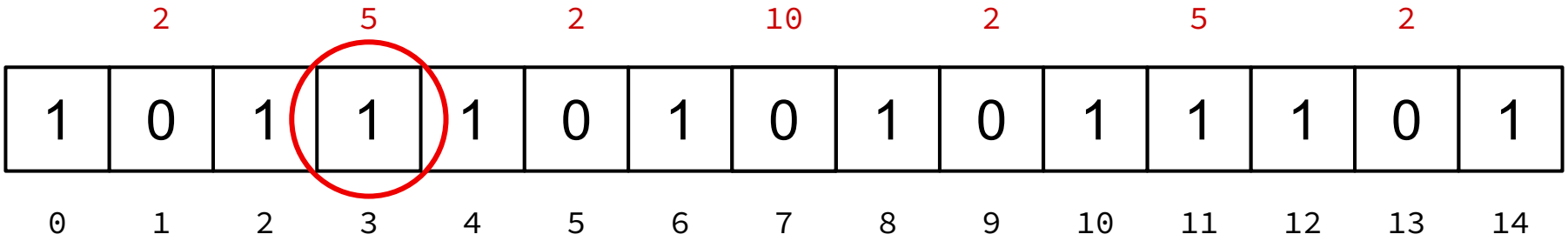
Exercício G - Code for 1

- Elemento na posição: **3**



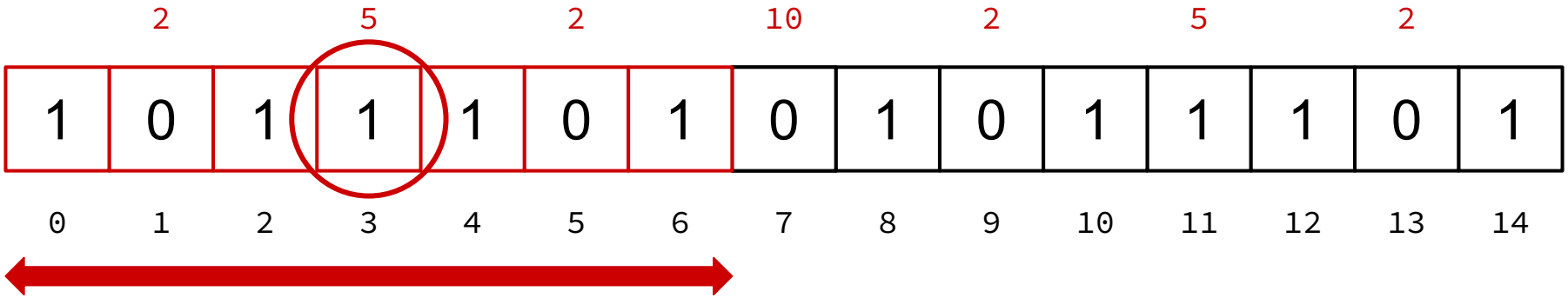
Exercício G - Code for 1

- Elemento na posição: **3**



Exercício G - Code for 1

- Elemento na posição: **3**



- Elemento central do intervalo [0 : 6]
  - (0 + 6) / 2 = 3

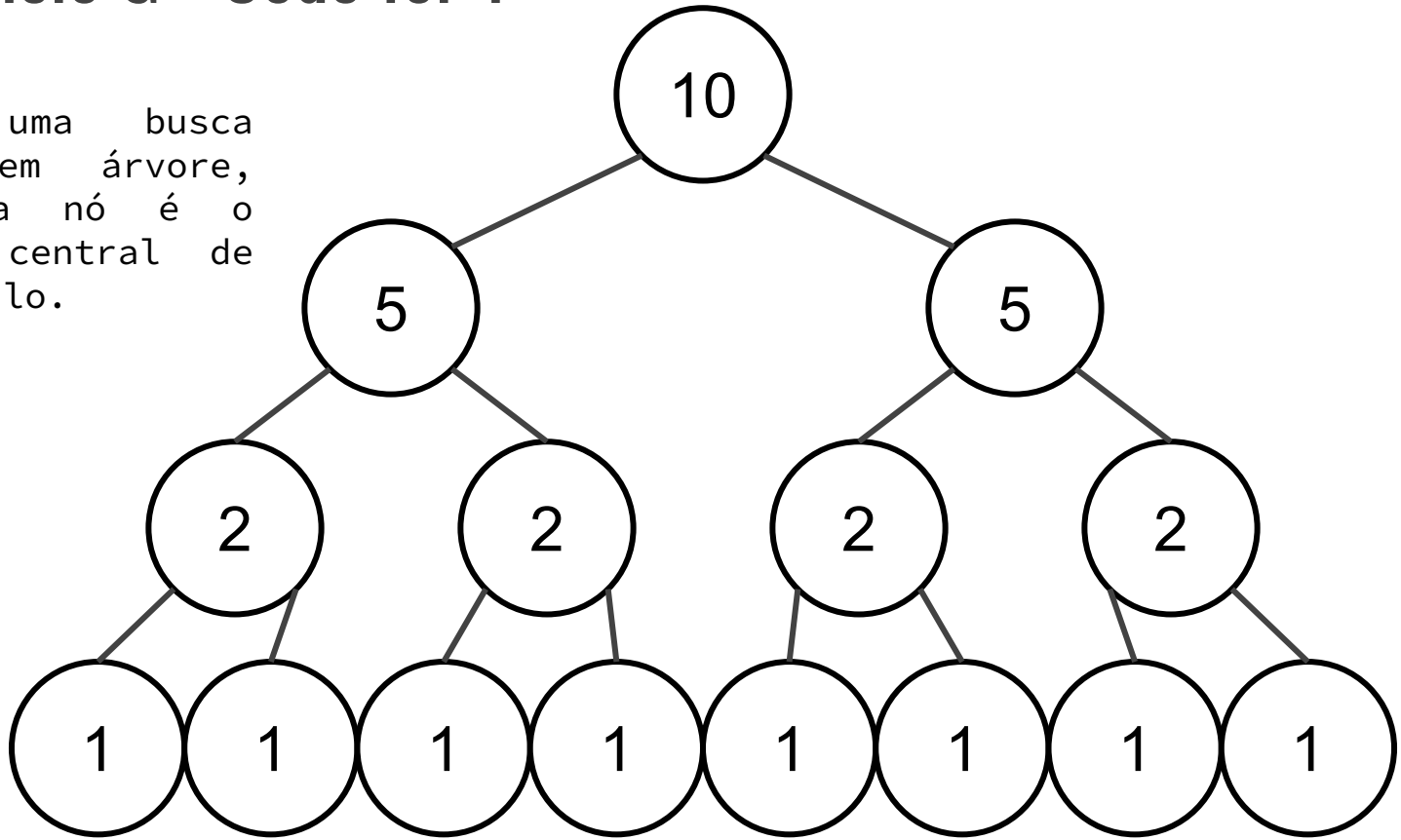
Exercício G - Code for 1

- O algoritmo para definir o elemento na posição X se assemelha a uma busca binária, segmentando nosso vetor em vetores menores e acessando o elemento central;

- No entanto, como podemos montar nosso vetor para aplicar essa busca? Também, como podemos recuperar o número de 1’s em determinado intervalo sem precisar

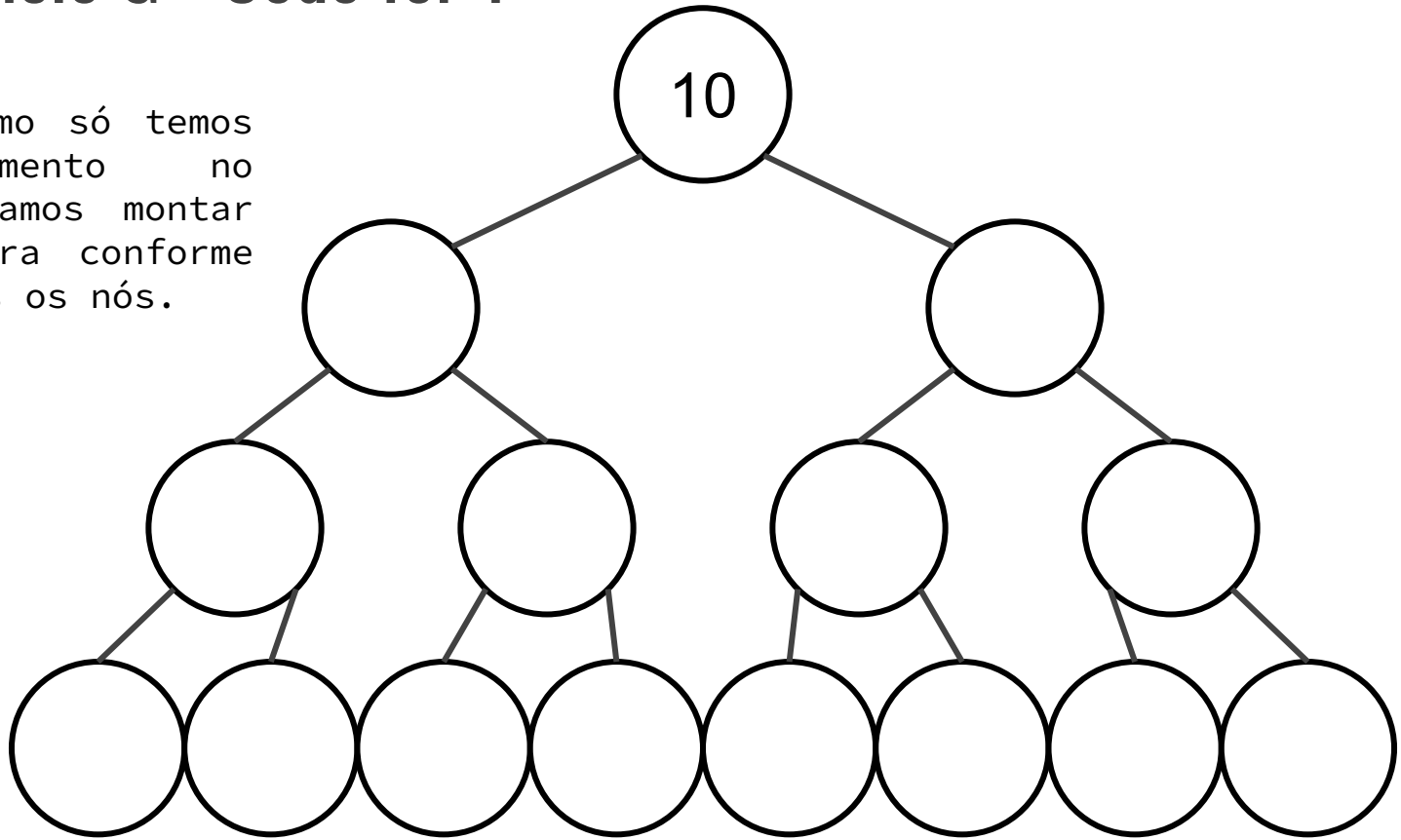
Exercício G - Code for 1

Simular uma busca binária em árvore, onde cada nó é o elemento central de um intervalo.

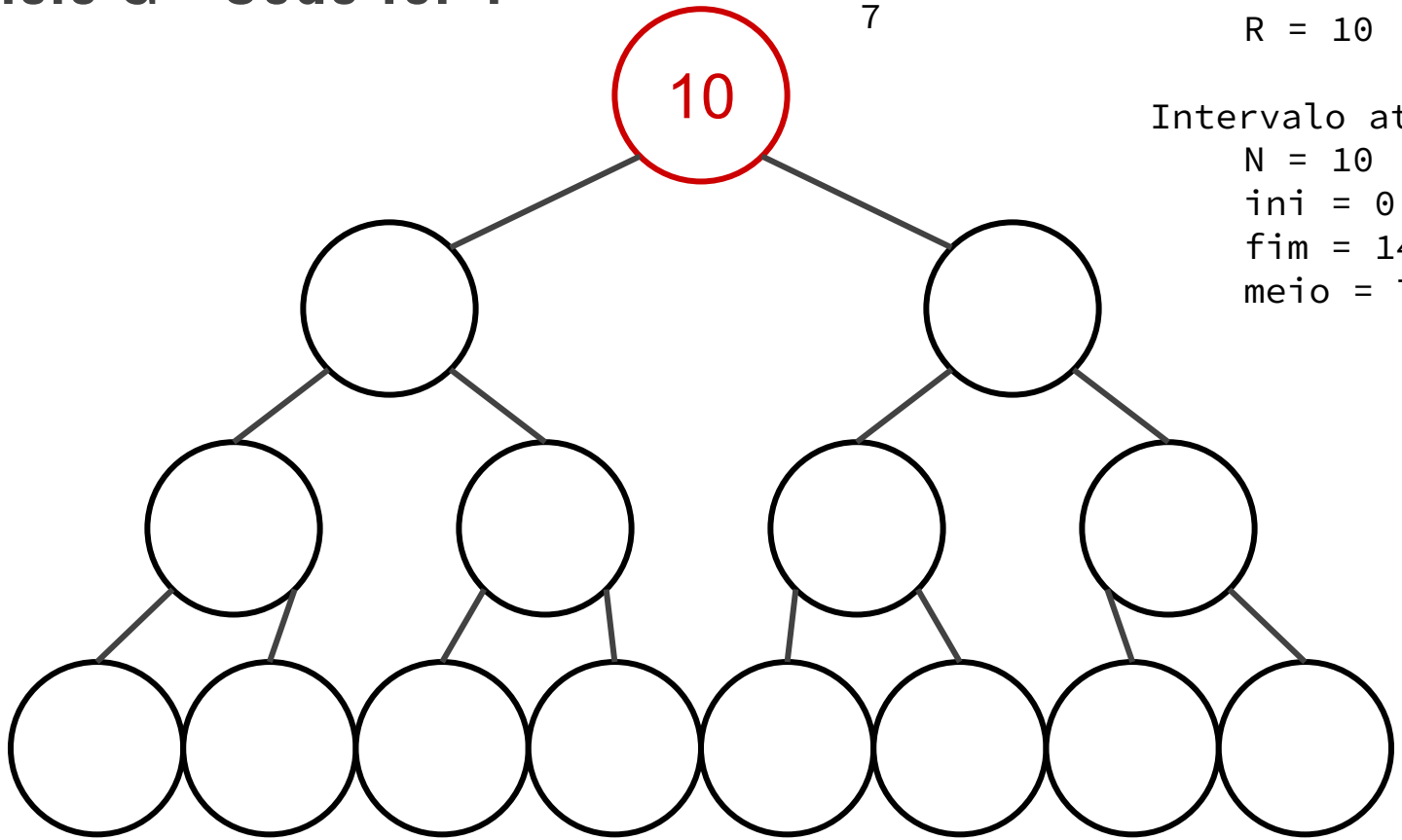


Exercício G - Code for 1

Porém, como só temos um elemento no começo, vamos montar a estrutura conforme acessarmos os nós.



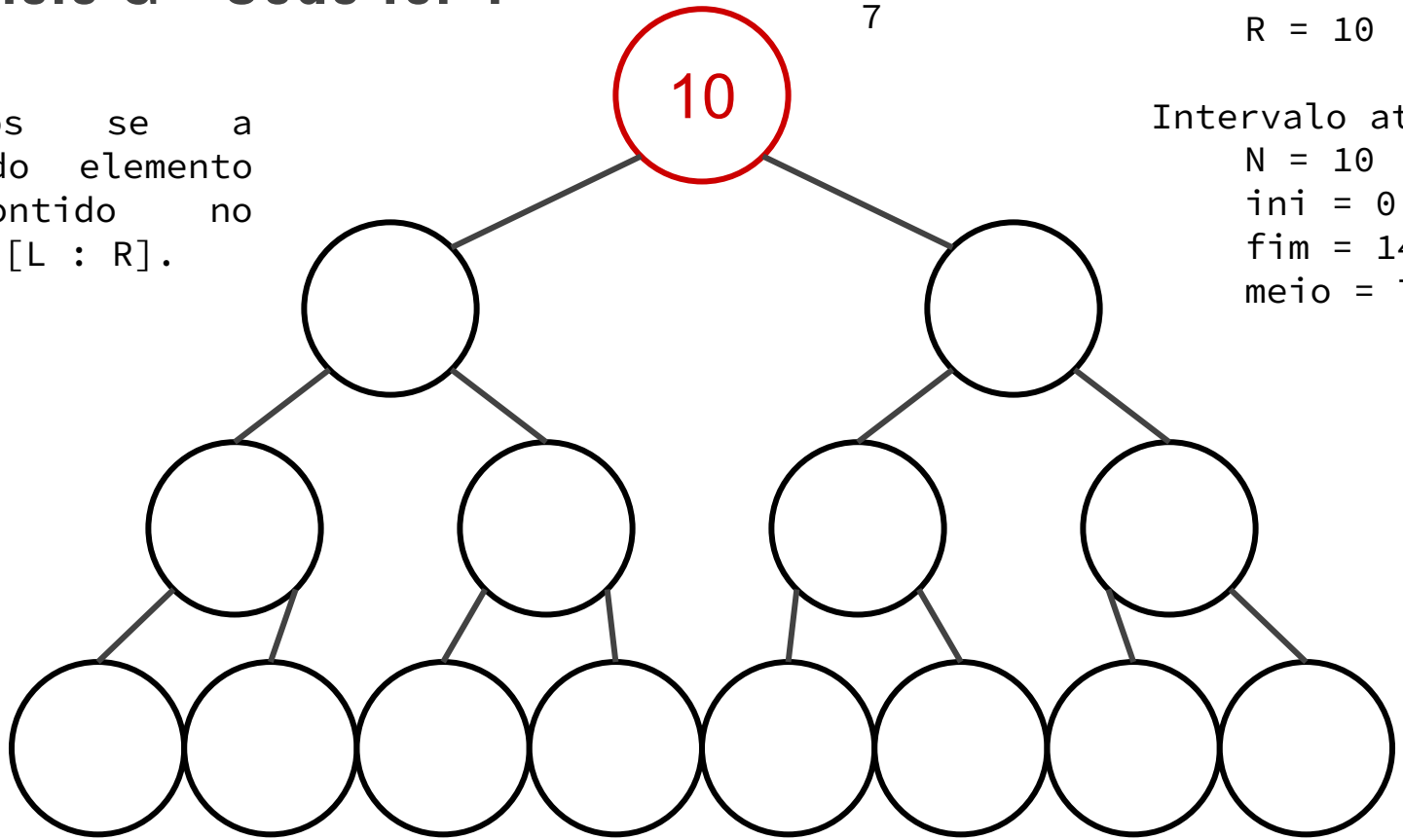
Exercício G - Code for 1




```
Resp = 0

Intervalo dado:
     L = 3
     R = 10

Intervalo atual:
     N = 10
     ini = 0
     fim = 14
     meio = 7
```

Exercício G - Code for 1

Verificamos se a posição do elemento está contido no intervalo [L : R].




```
Resp = 0

Intervalo dado:
    L = 3
    R = 10

Intervalo atual:
    N = 10
    ini = 0
    fim = 14
    meio = 7
```

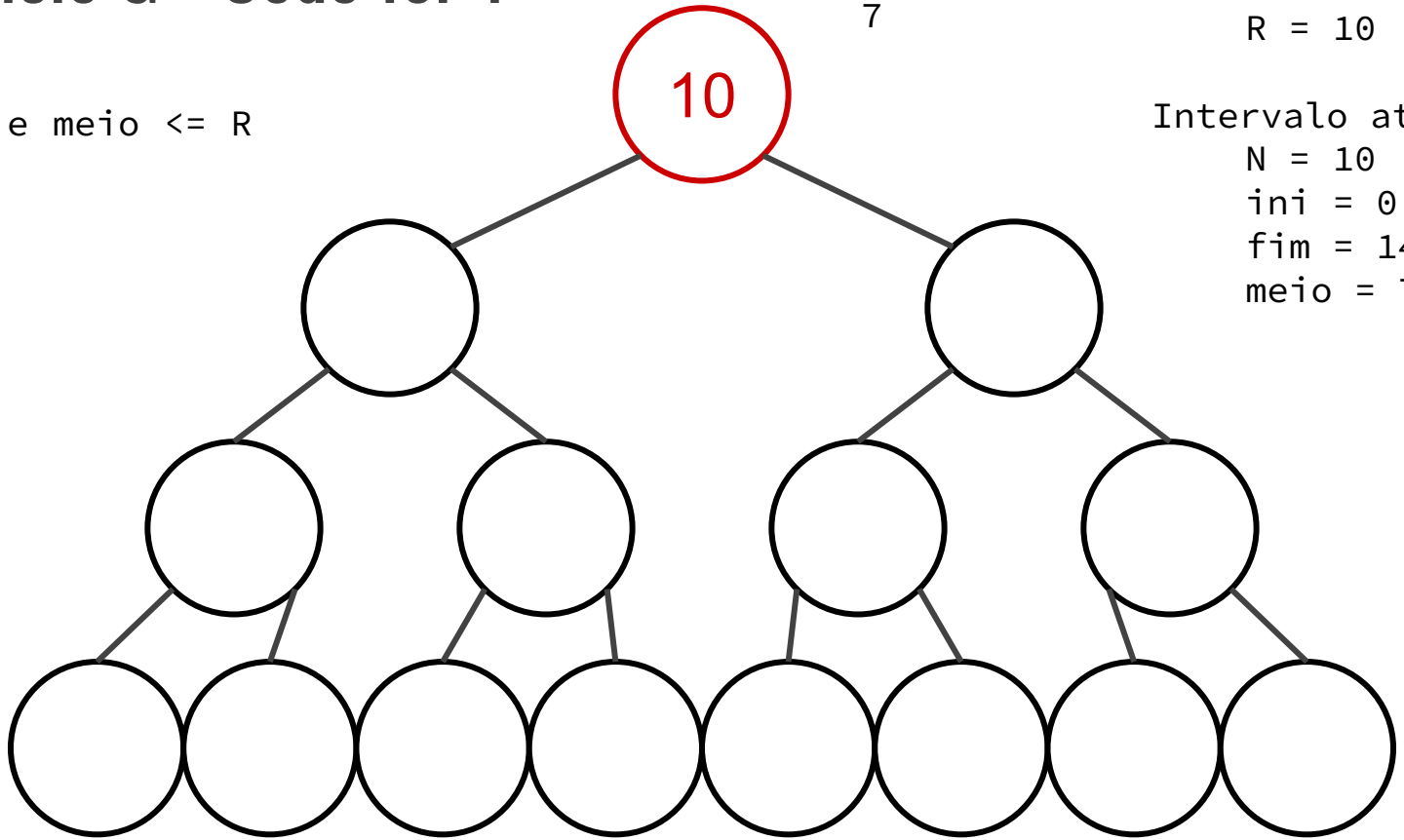
Exercício G - Code for 1

```
Resp = 0

Intervalo dado:
     L = 3
     R = 10

Intervalo atual:
     N = 10
     ini = 0
     fim = 14
     meio = 7
```

[0 : 14]
7

10

meio >= L e meio <= R
?



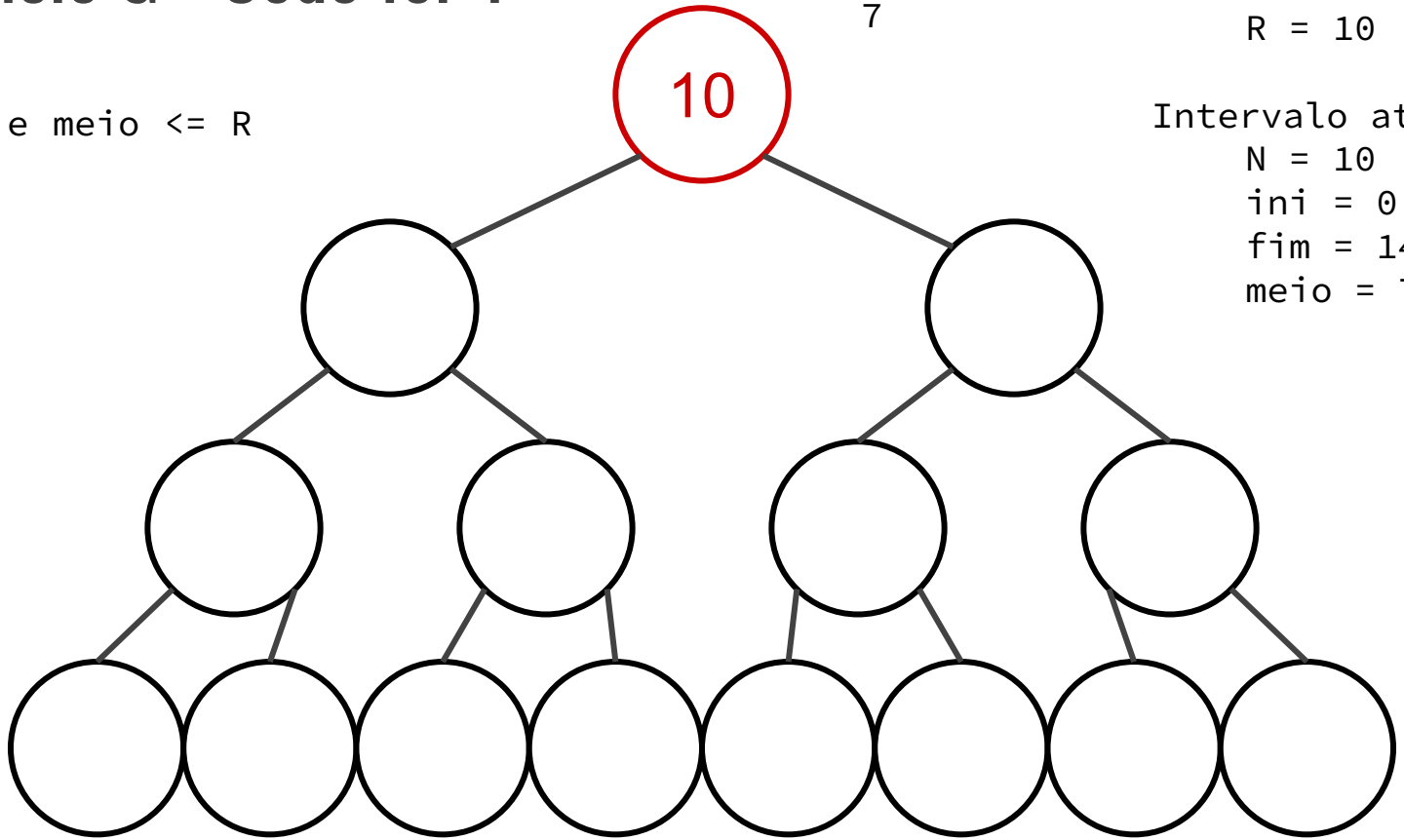
Exercício G - Code for 1

```
Resp = 0

Intervalo dado:
     L = 3
     R = 10

Intervalo atual:
     N = 10
     ini = 0
     fim = 14
     meio = 7
```

[0 : 14]
7

10

meio >= L e meio <= R
? SIM!



Exercício G - Code for 1

[0 : 14]
7

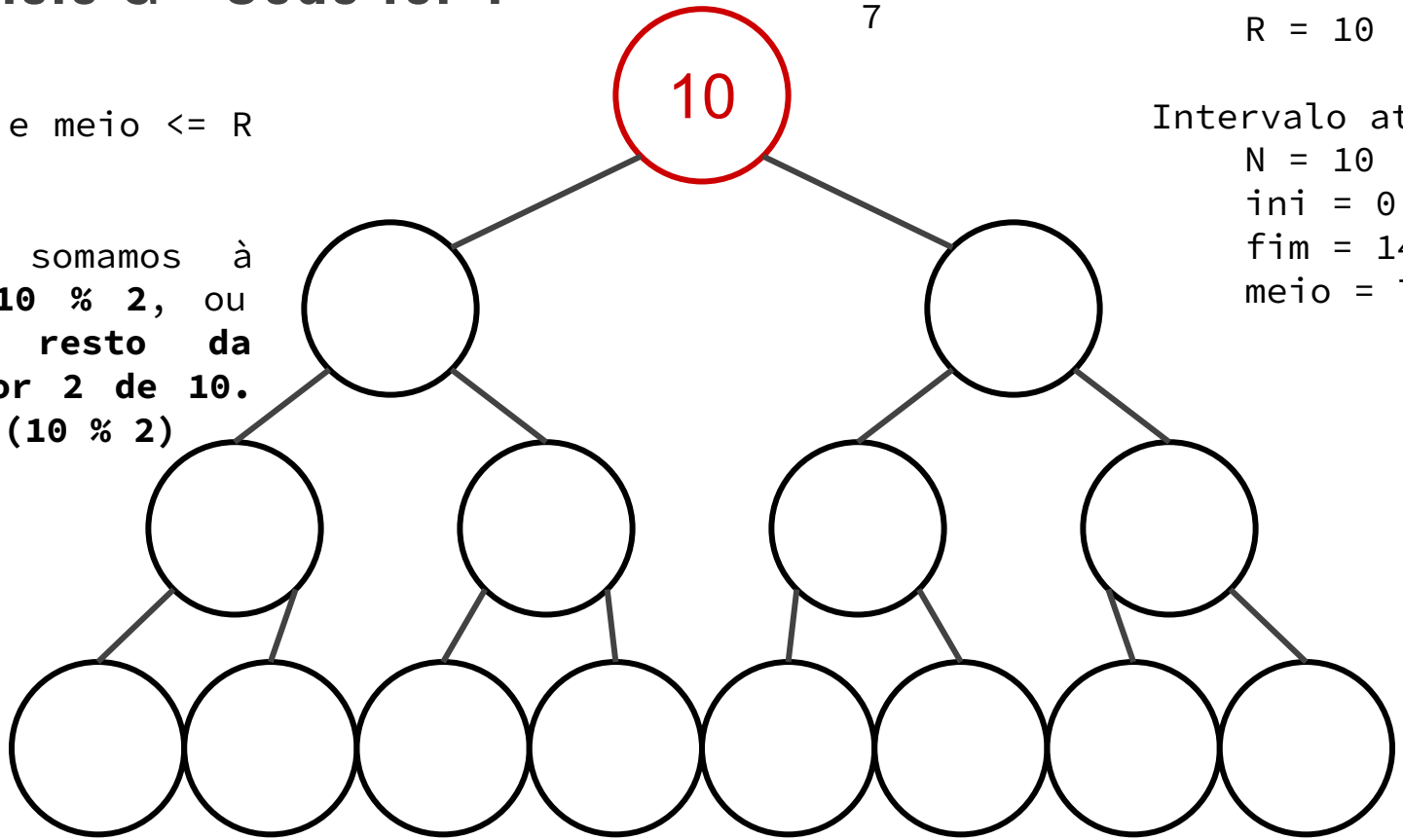
Resp = 0

Intervalo dado:
L = 3
R = 10

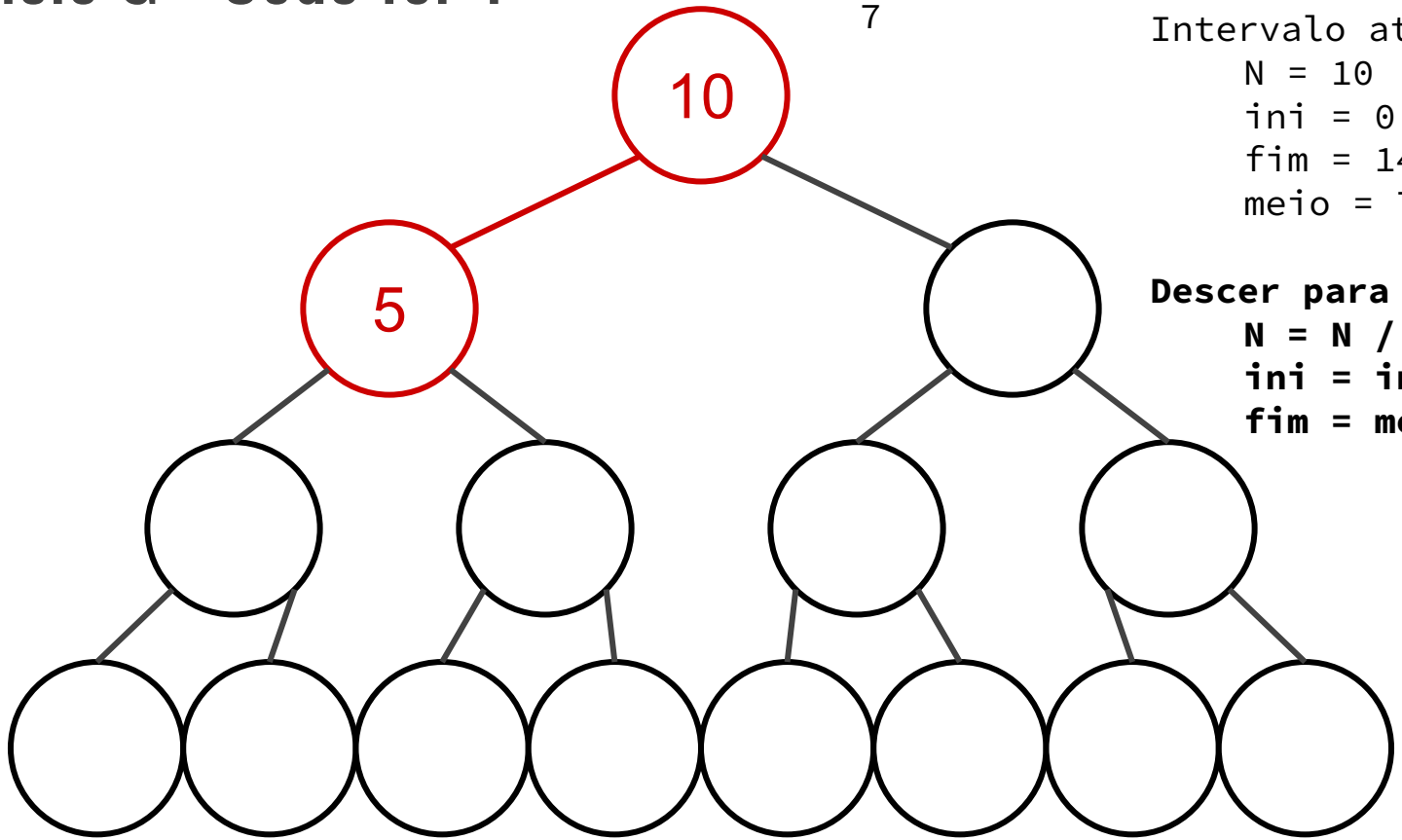
Intervalo atual:
N = 10
ini = 0
fim = 14
meio = 7

meio >= L e meio <= R
? SIM!

Portanto, somamos à resposta **10 % 2**, ou seja, o **resto da divisão por 2 de 10.**
Resp += 0 (10 % 2)



Exercício G - Code for 1



Intervalo dado:
    L = 3
    R = 10

Intervalo atual:
    N = 10
    ini = 0
    fim = 14
    meio = 7

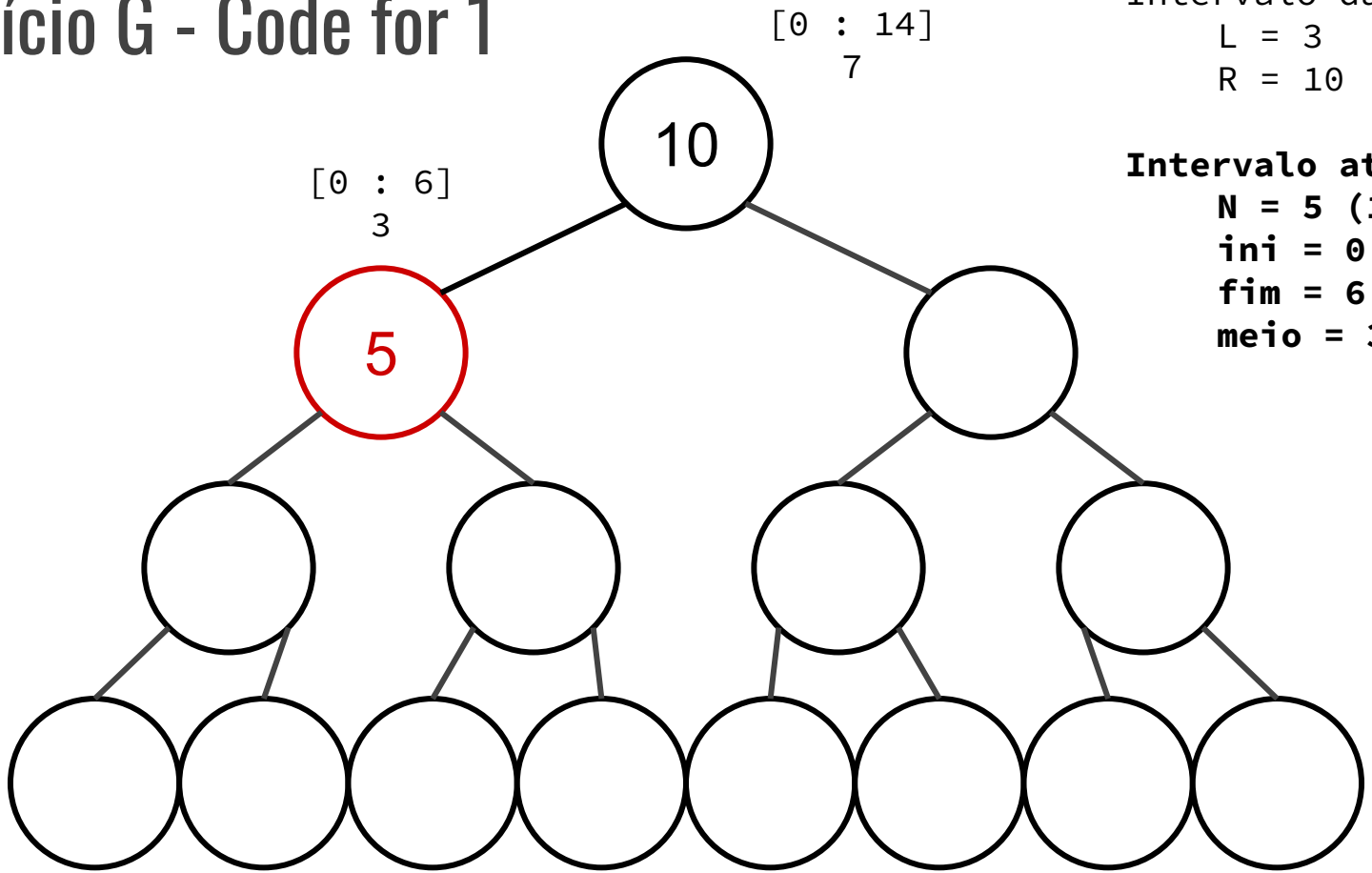
Descer para esquerda:
    N = N / 2
    ini = ini
    fim = meio - 1

Exercício G - Code for 1

Resp = 0

Intervalo dado:
    L = 3
    R = 10

Intervalo atual:
    N = 5 (10 / 2)
    ini = 0
    fim = 6 (7 - 1)
    meio = 3



Exercício G - Code for 1

Resp = 1

Intervalo dado:
    L = 3
    R = 10

Intervalo atual:
    N = 5 (10 / 2)
    ini = 0
    fim = 6 (7 - 1)
    meio = 3

[0 : 14]
7

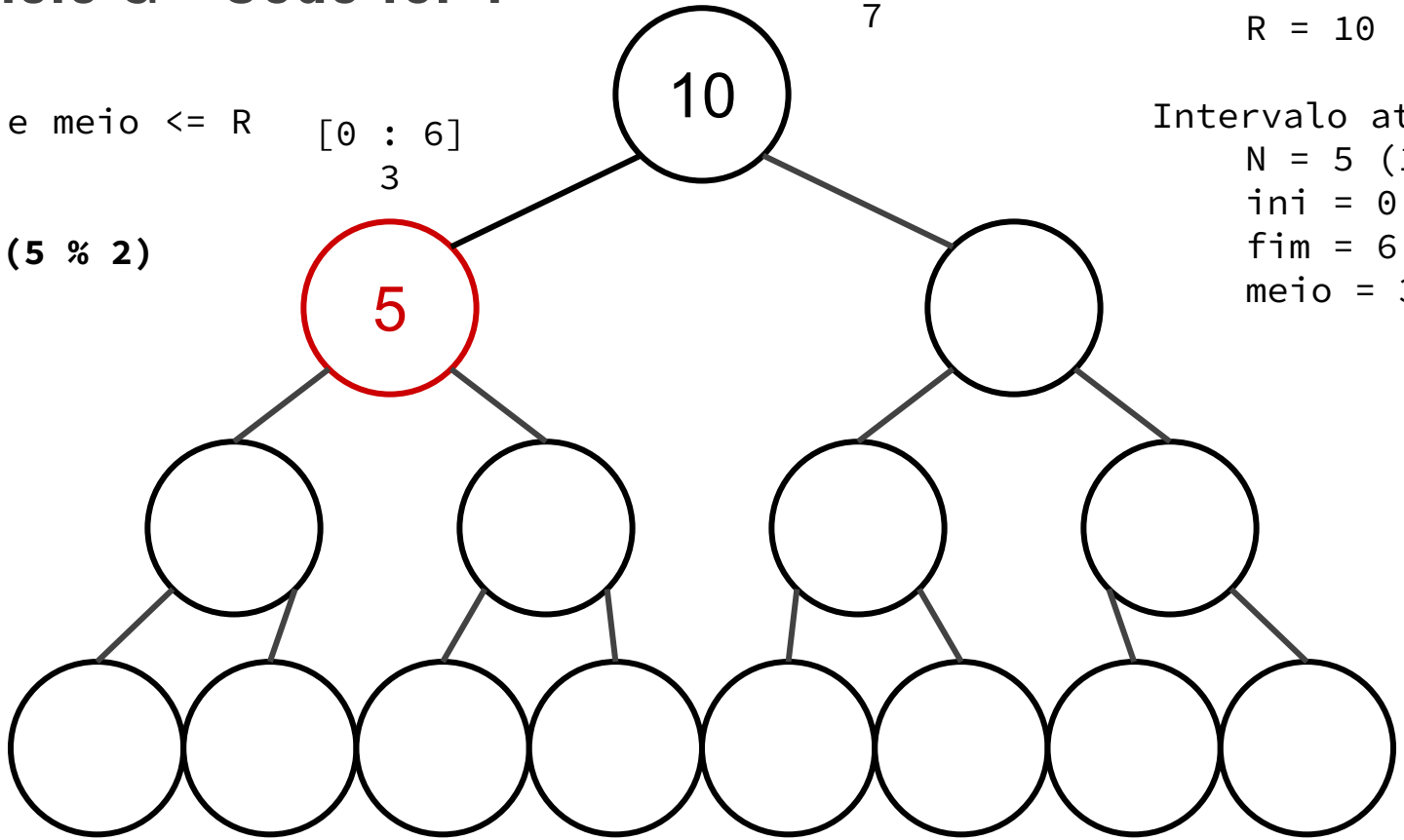
10

[0 : 6]
3

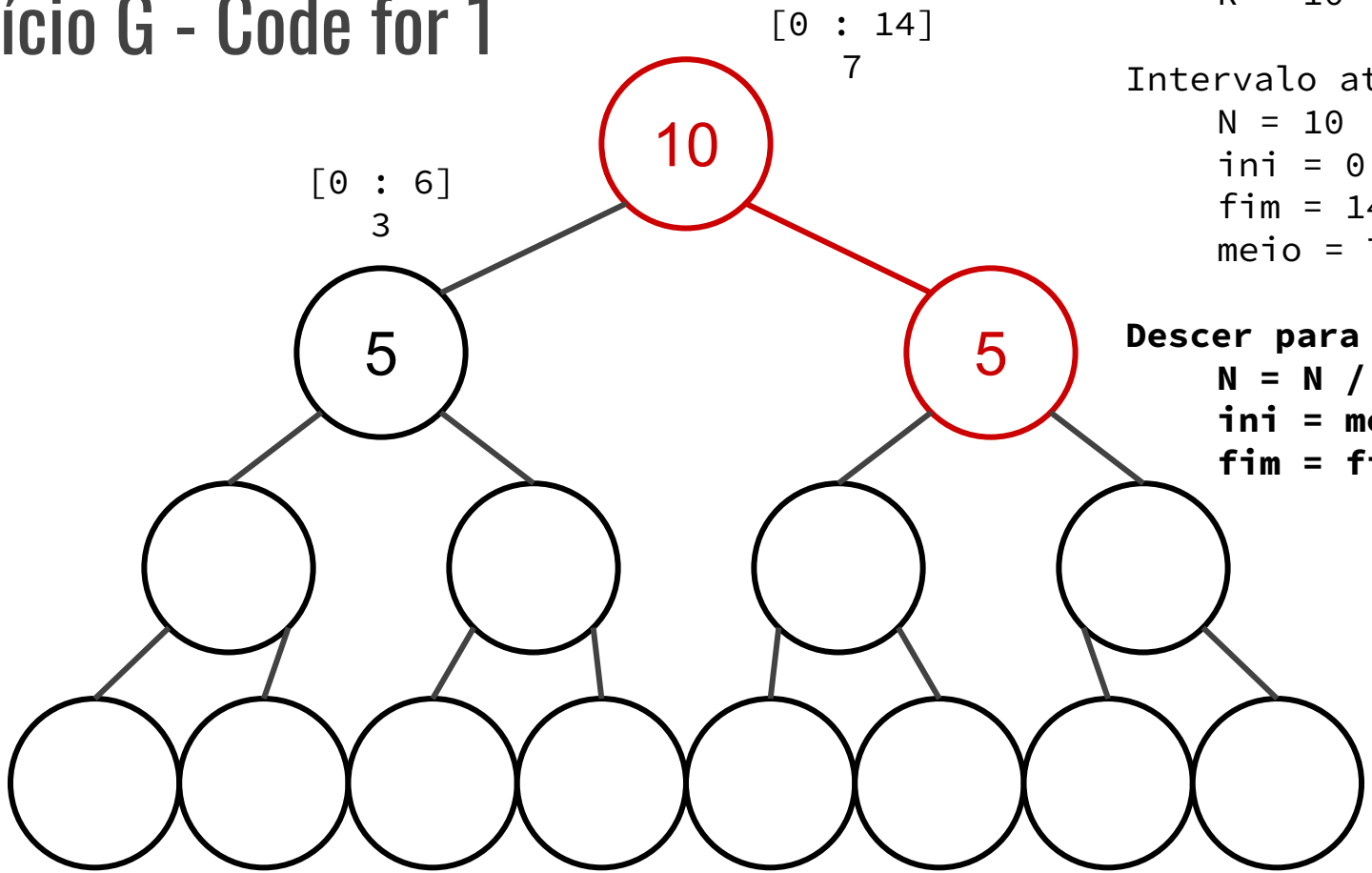
5

meio >= L e meio <= R
? SIM!

Resp += 1 (5 % 2)



Exercício G - Code for 1

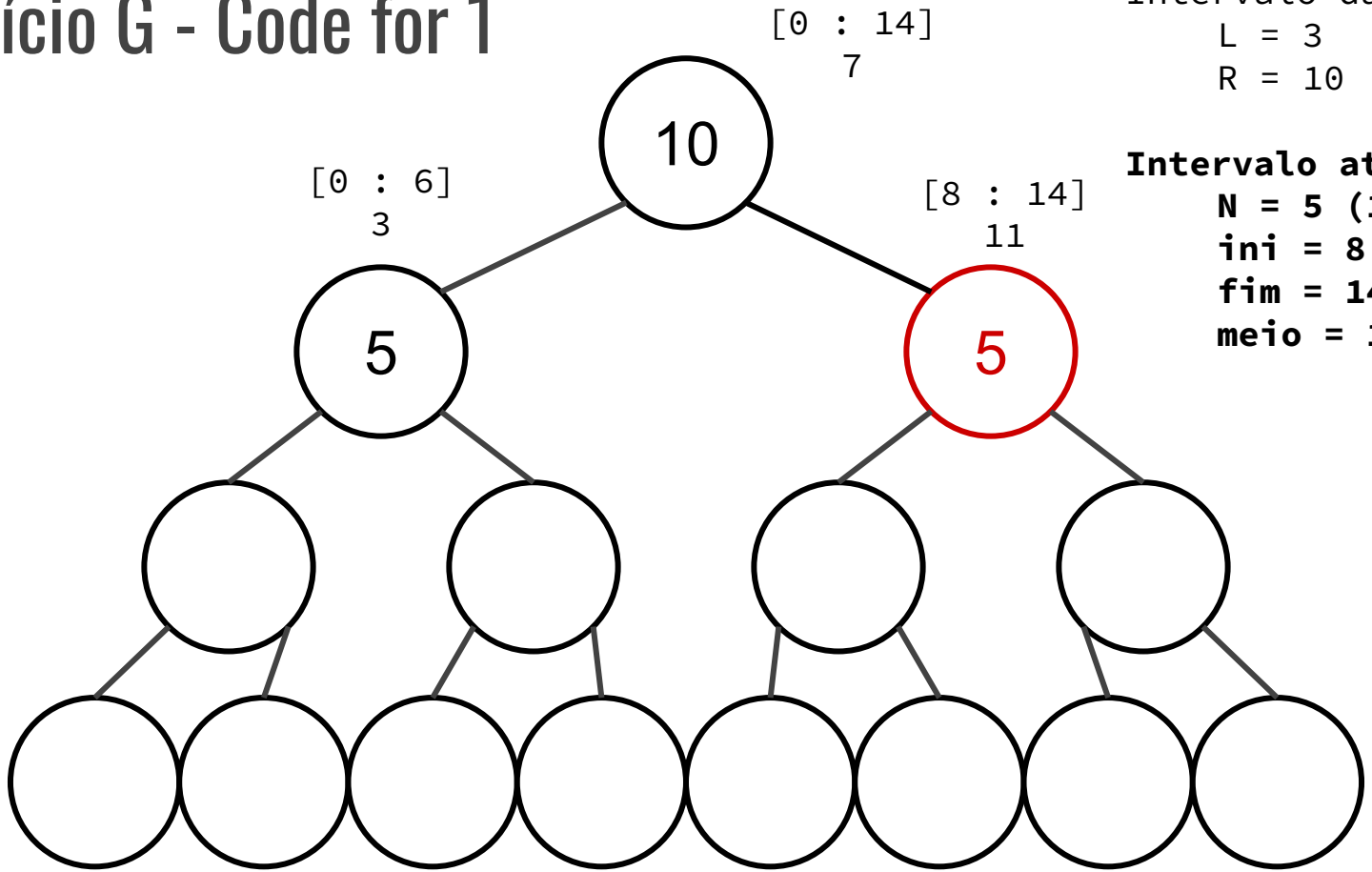


Intervalo dado:
    L = 3
    R = 10

Intervalo atual:
    N = 10
    ini = 0
    fim = 14
    meio = 7

Descer para direita:
    N = N / 2
    ini = meio + 1
    fim = fim

Exercício G - Code for 1



Resp = 2

Intervalo dado:
    L = 3
    R = 10

Intervalo atual:
    N = 5 (10 / 2)
    ini = 8 (7 + 1)
    fim = 14
    meio = 11

Exercício G - Code for 1

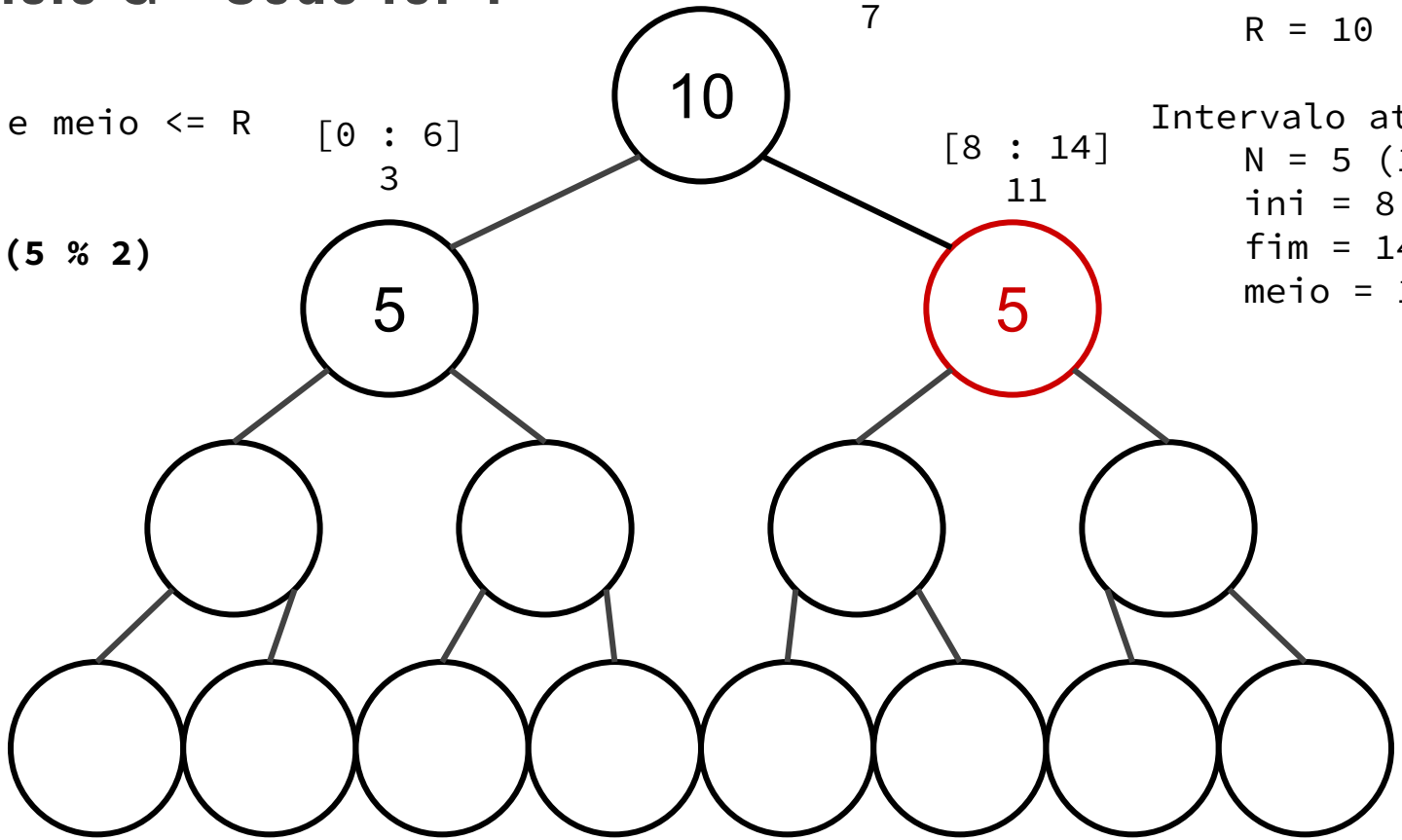
Resp = 2

Intervalo dado:
    L = 3
    R = 10

Intervalo atual:
    N = 5 (10 / 2)
    ini = 8 (7 + 1)
    fim = 14
    meio = 11

meio >= L e meio <= R
? NÃO!

Resp += 1 (5 % 2)

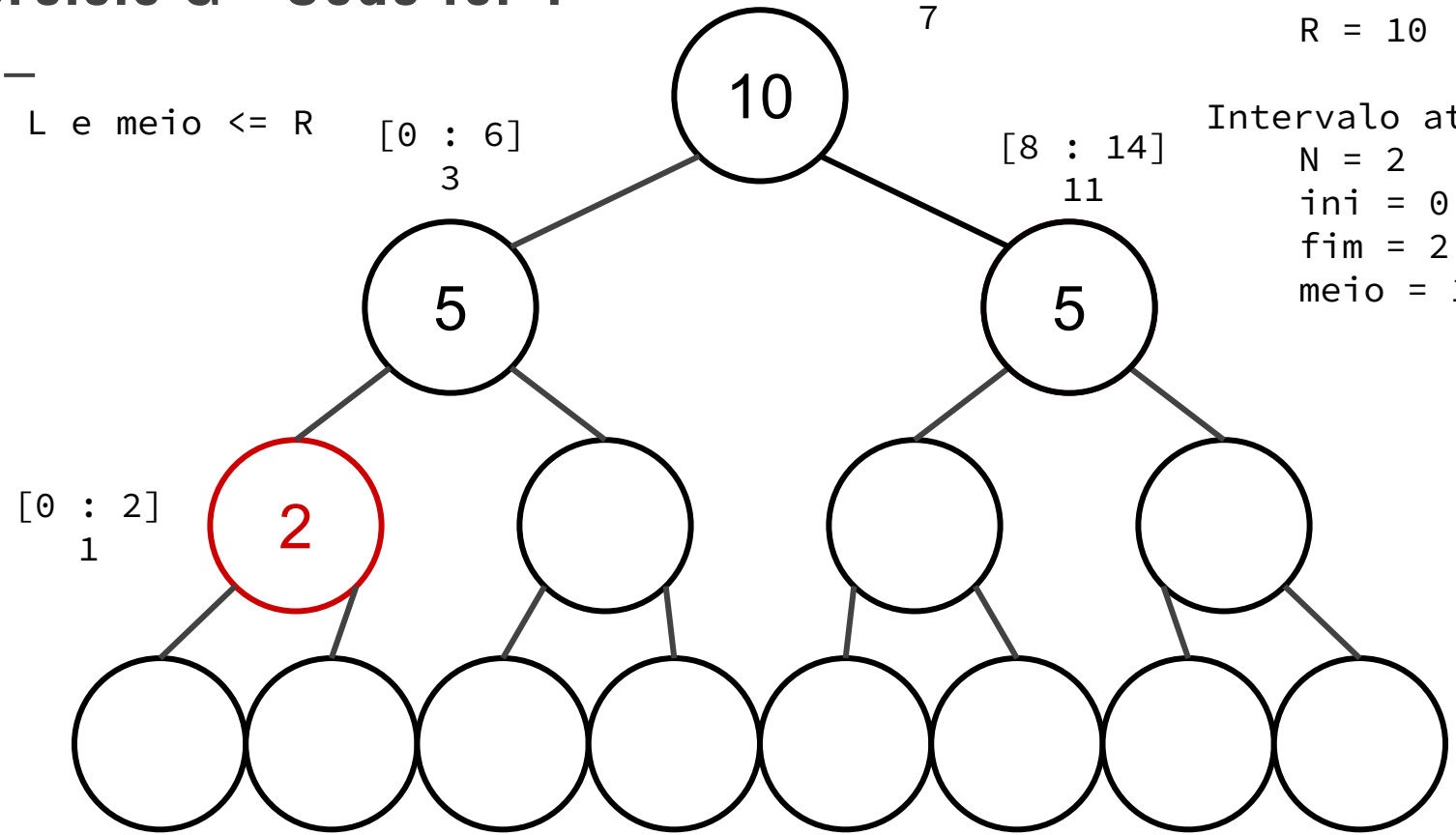


Exercício G - Code for 1

Resp = 2

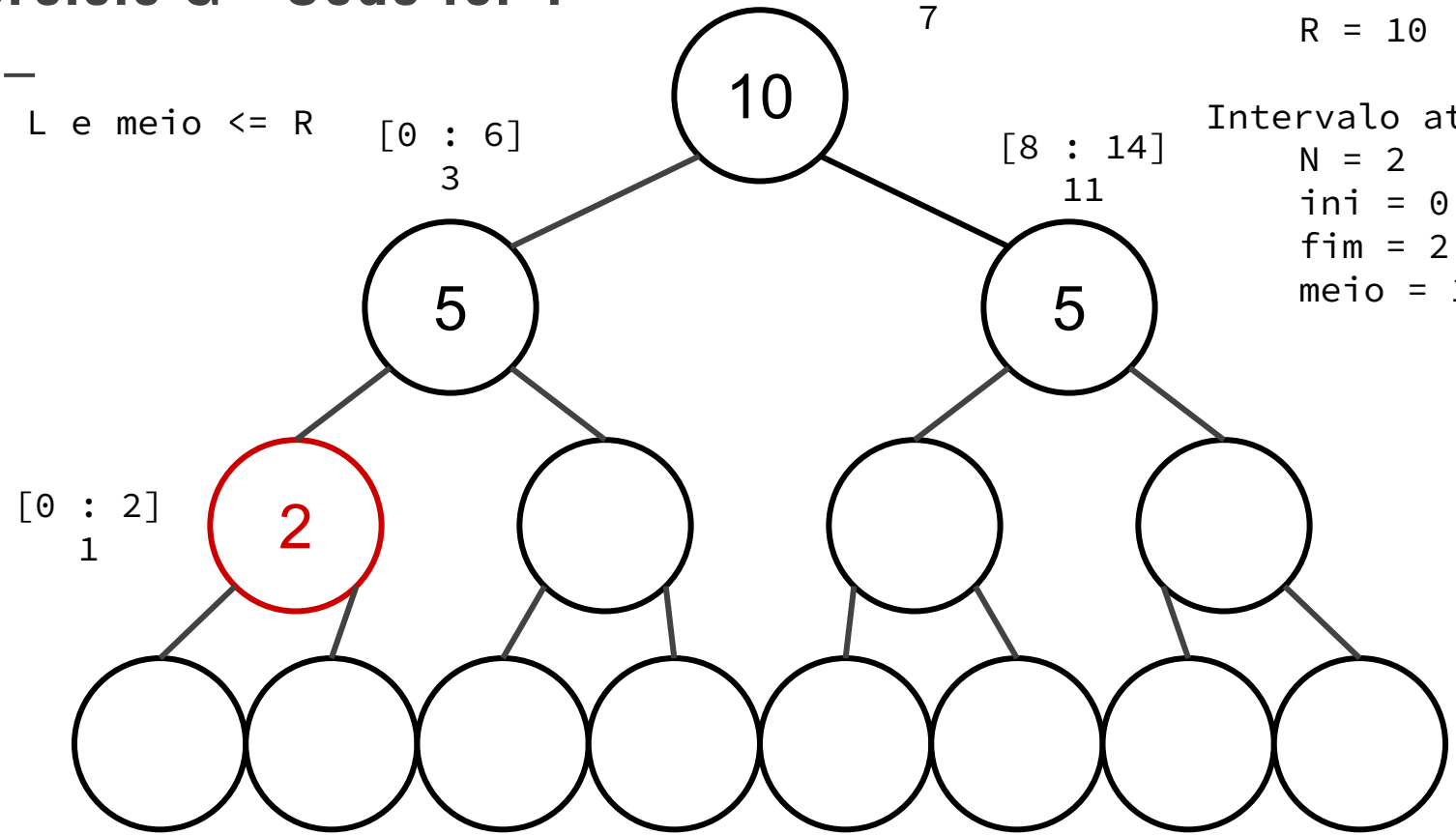
Intervalo dado:
L = 3
R = 10

Intervalo atual:
N = 2
ini = 0
fim = 2
meio = 1



Exercício G - Code for 1

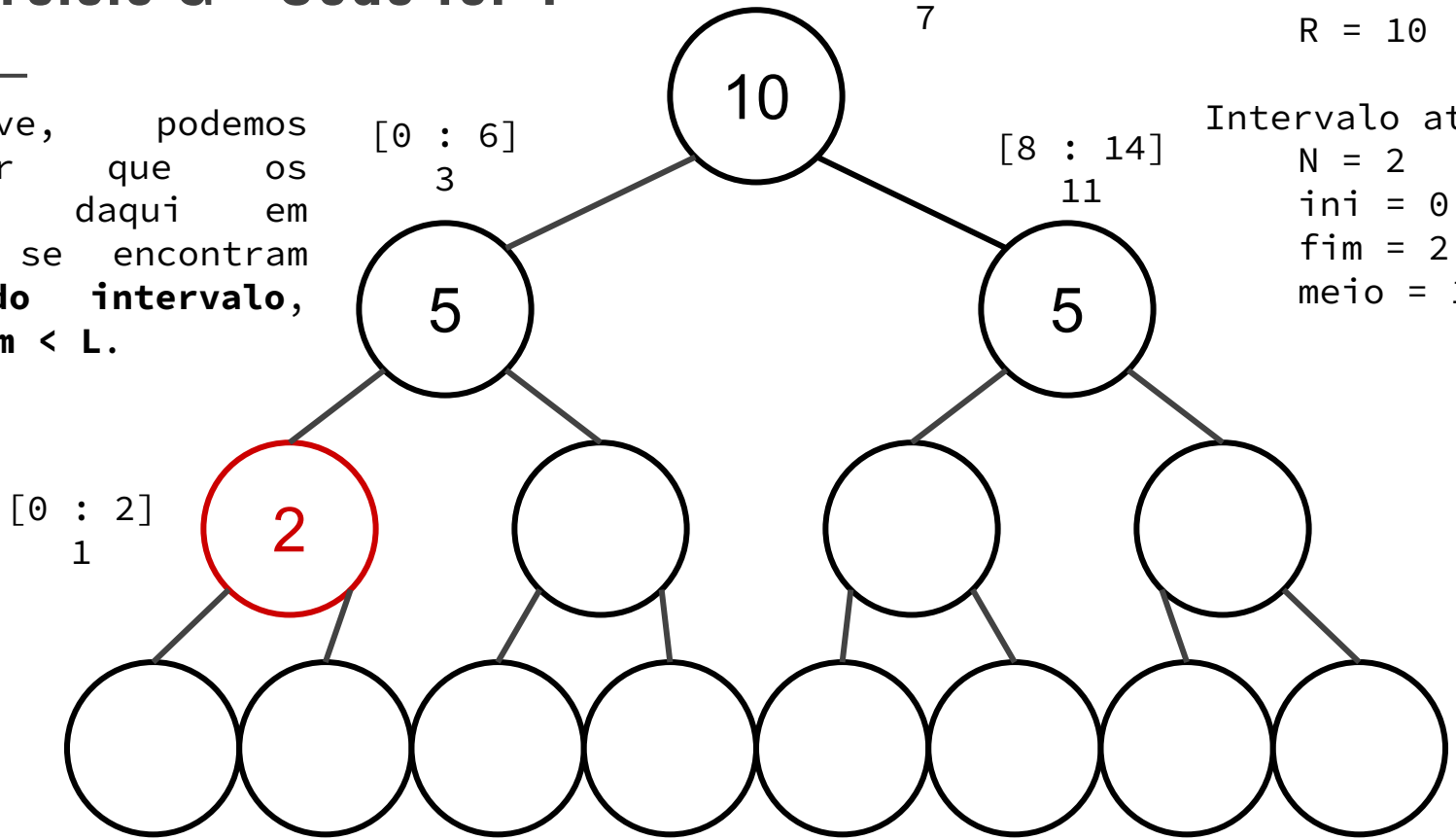
```
Resp = 2

Intervalo dado:
     L = 3
     R = 10

Intervalo atual:
     N = 2
     ini = 0
     fim = 2
     meio = 1
```



meio >= L e meio <= R
? NÃO…

Exercício G - Code for 1

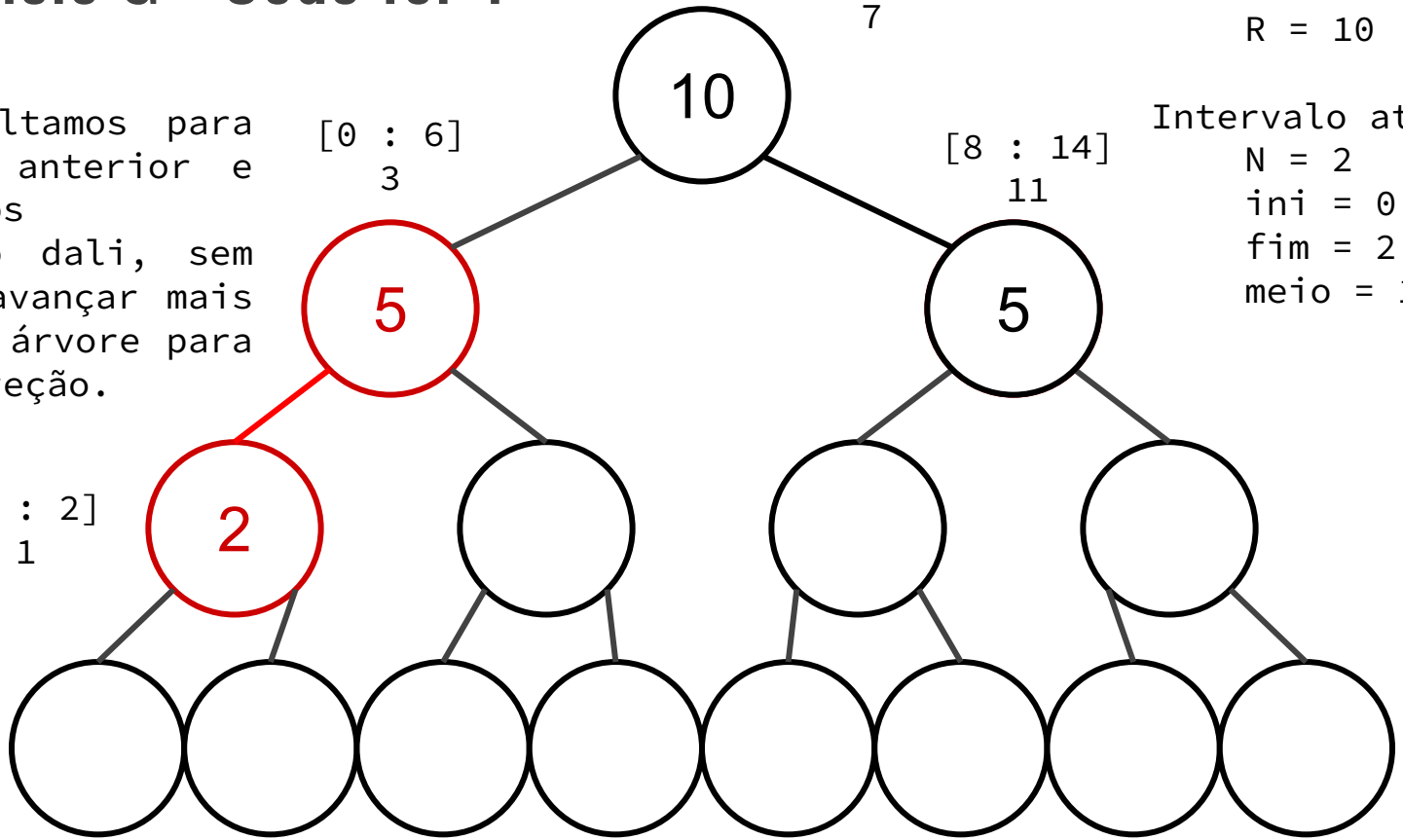
Inclusive, podemos perceber que os valores daqui em diante se encontram **fora do intervalo**, pois **fim < L**.




```
Resp = 2

Intervalo dado:
    L = 3
    R = 10

Intervalo atual:
    N = 2
    ini = 0
    fim = 2
    meio = 1
```

Exercício G - Code for 1

Assim, voltamos para o estado anterior e continuamos explorando dali, sem precisar avançar mais abaixo na árvore para aquela direção.



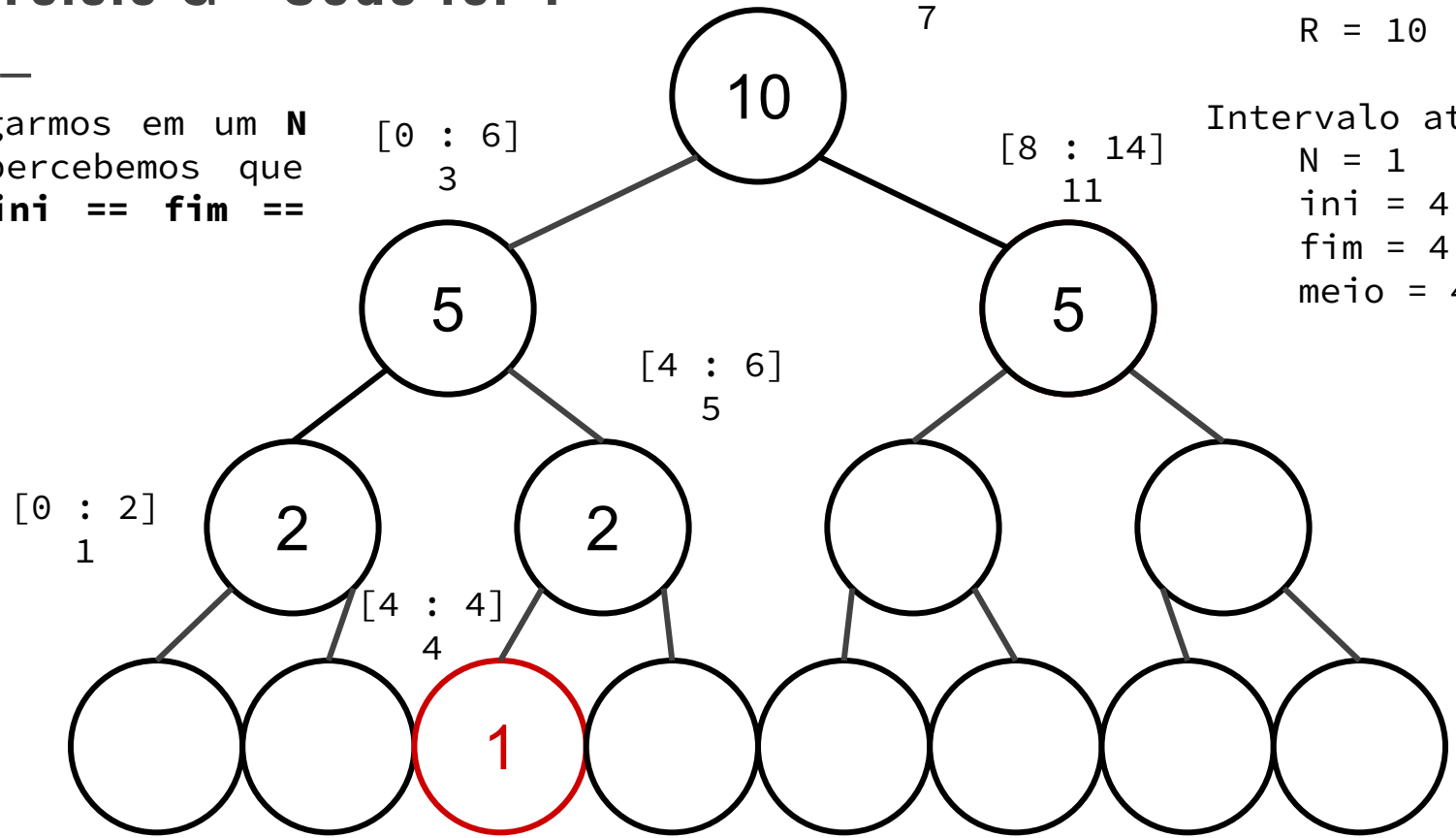
Resp = 2

Intervalo dado:
L = 3
R = 10

Intervalo atual:
N = 2
ini = 0
fim = 2
meio = 1

Exercício G - Code for 1

Se chegarmos em um **N = 1**, percebemos que nosso **ini == fim == meio**.



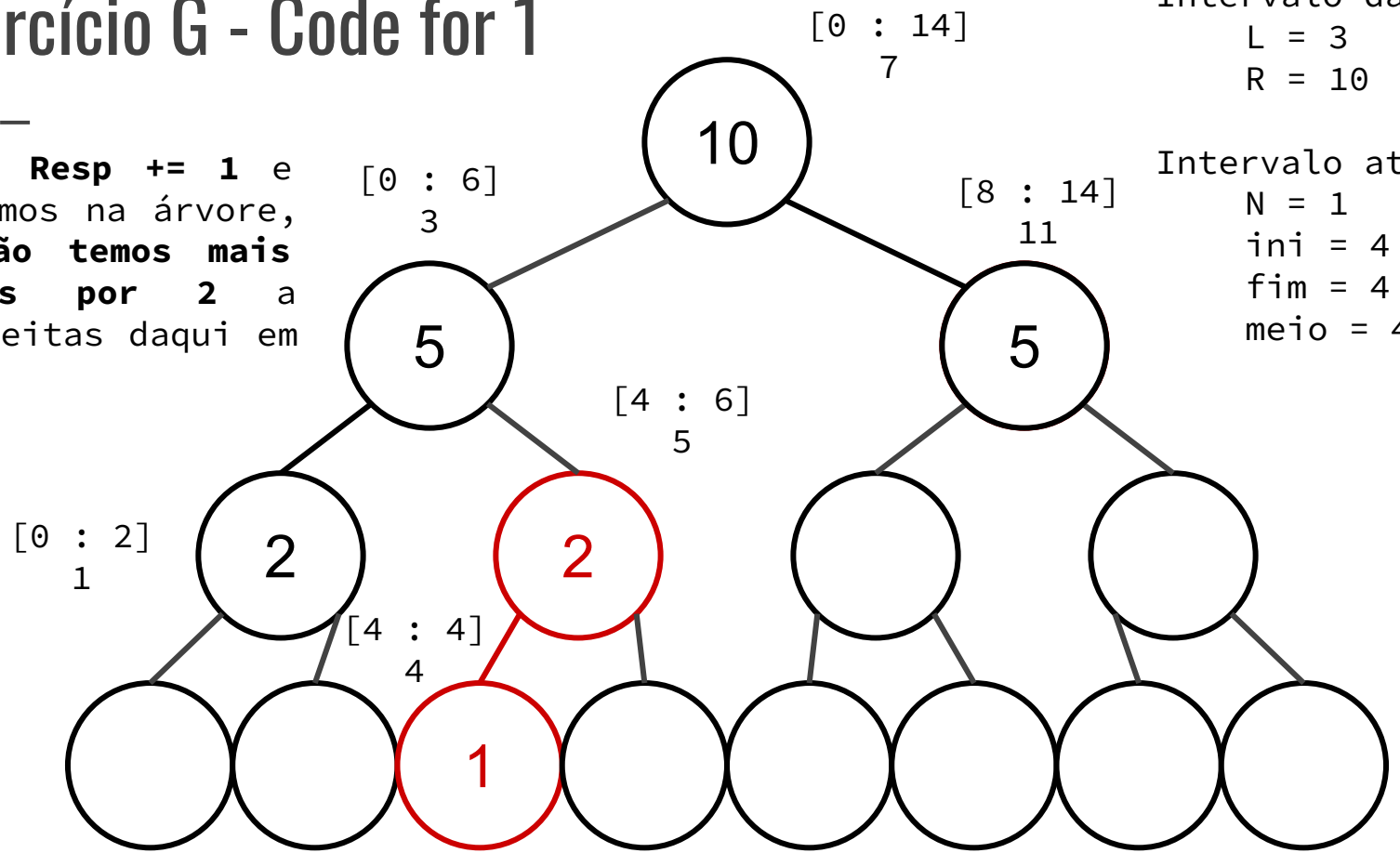
Resp = 2

Intervalo dado:
L = 3
R = 10

Intervalo atual:
N = 1
ini = 4
fim = 4
meio = 4

Exercício G - Code for 1

Somamos **Resp += 1** e retornamos na árvore, pois **não temos mais divisões por 2** a serem feitas daqui em diante.



Resp = 2

Intervalo dado:
L = 3
R = 10

Intervalo atual:
N = 1
ini = 4
fim = 4
meio = 4

Exercício G - Code for 1

```
int main(){
    ll n;
    cin >> n >> l >> r;
    ll i = 1;
    ll cont = 0;
    while(i <= n){
        i*=2;
        cont++;
    }
    ll fim = (1LL << cont) - 1LL;
    ll ini = 1LL;
    cout << func((fim+1)/2, ini, fim, n) <<
"\n";

    return 0;
}
```

Exercício G - Code for 1

```cpp
int main(){
    ll n;
    cin >> n >> l >> r;
    ll i = 1;
    ll cont = 0;
    while(i <= n){
        i*=2;
        cont++;
    }
    ll fim = (1LL << cont) - 1LL;
    ll ini = 1LL;
    cout << func((fim+1)/2, ini, fim, n) << "\n";

    return 0;
}
```

```cpp
ll func(ll pos, ll ini, ll fim, ll val){
    if(ini > r || fim < l || val == 0)return 0;

    if(val == 1){
        if(pos < l || pos > r) return 0;
        return 1;
    }
    ll pos1 = (pos+ini-1LL)/2LL;
    ll pos2 = (fim+pos+1LL)/2LL;

    ll soma = 0;
    if(val&1 && pos >= l && pos <= r)soma++;
    soma += func(pos1, ini, pos-1LL, val/2LL) + func(pos2, pos+1LL, fim, val/2LL);
    return soma;
}
```

Exercício G - Code for 1


```cpp
int main(){
    ll n;
    cin >> n >> l >> r;
    ll i = 1;
    ll cont = 0;
    while(i <= n){
        i*=2;
        cont++;
    }
    ll fim = (1LL << cont) - 1LL;
    ll ini = 1LL;
    cout << func((fim+1)/2, ini, fim, n) << "\n";

    return 0;
}
```

```cpp
ll func(ll pos, ll ini, ll fim, ll val){
    if(ini > r || fim < l || val == 0)return 0;

    if(val == 1){
        if(pos < l || pos > r) return 0;
        return 1;
    }
    ll pos1 = (pos+ini-1LL)/2LL;
    ll pos2 = (fim+pos+1LL)/2LL;

    ll soma = 0;
    if(val&1 && pos >= l && pos <= r)soma++;
    soma += func(pos1, ini, pos-1LL, val/2LL) + func(pos2, pos+1LL, fim, val/2LL);
    return soma;
}
```