# Gerenciador de Memória

Rafaelo Pinheiro
Eduardo Veiga
Harlan Maas Martins
Eliezer Flores

# Sumário:

- Introdução;
- Arquivo .pyc;
- Processo;
- Estrutura do Programa;
- Operação de Alocação;
- Operações de Referência;
- Varredura da Memória;
- Garbage Collector;
- Coclusão;
- Referências;

# Introdução

- Implementação de um simulador de gerência dinâmica de memória;
- Utilização da Linguagem de Programação Python;
- Projeto desenvolvido com base no algoritmo Mark-and-Sweep;

# Arquivo .mypyc

- Contém bytecodes de alocações de memória de um programa;
- Contém informações essenciais sobre as classes de um programa;
- O Mark Sweep irá ler este arquivo para realizar as operações de alocação

# Processo

classe : casa
campos : quarto,patio,sala
tipos : ptr,ptr,ptr
tamanho : 4

classe : patio
campos : n
tipos : int
tamanho : 5
classe : sala
campos : n
tipos : int
tamanho : 5

classe : quarto
campos : n
tipos : int
tamanho : 5

Bytecodes:
a = casa();
a.quarto = quarto();
a.sala = sala();
a.patio = patio();

# Processo

classe : Tree
campos : left,right,info
tipos : ptr,ptr,int
tamanho : 7

bytecodes:
root = tree();
root.left = tree();
root.right = tree();

classe : List
campos : info,prox
Tipos : int,ptr
tamanho : 6

bytecodes:
Var1 = list();
Var2 = list();
var1.prox = var2;

# Processo

- Busca seqüencial de uma instrução no arquivo .mypyc
- Instrução é decodificada para definir qual operação será feita(alocação, referenciação)
- Decodificação das instruções (referencia dos objetos);
- Execução da operação de alocação(caso seja essa a operação a ser feita).
- Ajuste das referências.

# Processos

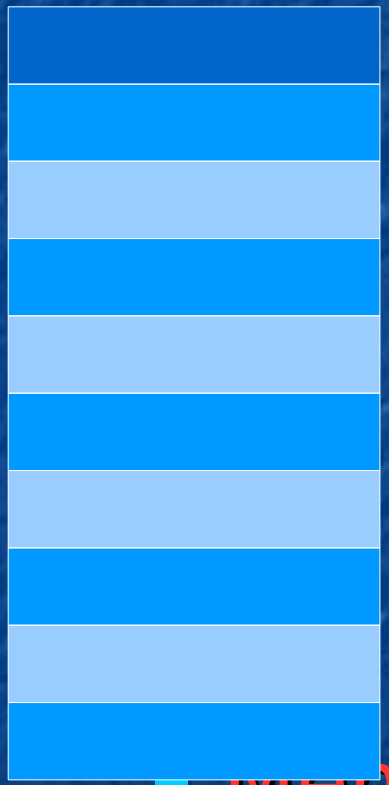
- Operação de alocação
- Obj = class();
- Operação de referenciação
- Var1 = var2;

# Processos

- Operação de alocação
- Obj = class();
- Operação de referenciação
- Var1 = var2;

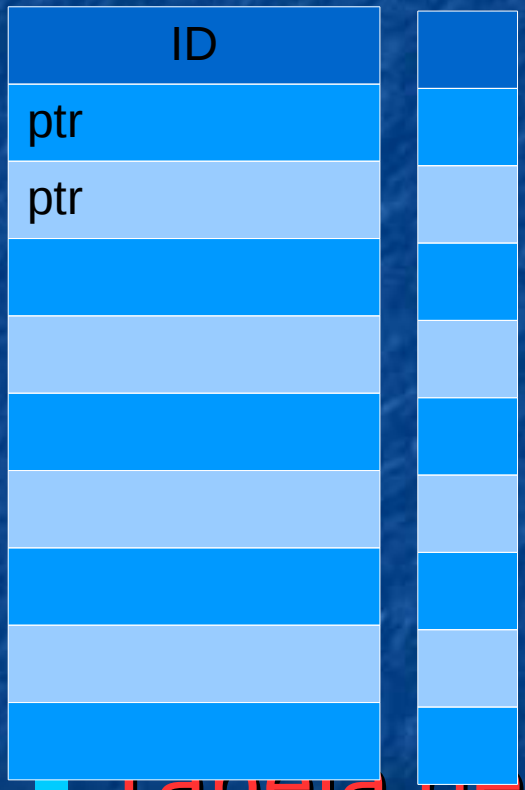
- Obs: operações de alocação também possuem manipulação de referências

# Estrutura do programa



- Memória: lista de tamanho fixo

# Estrutura do programa

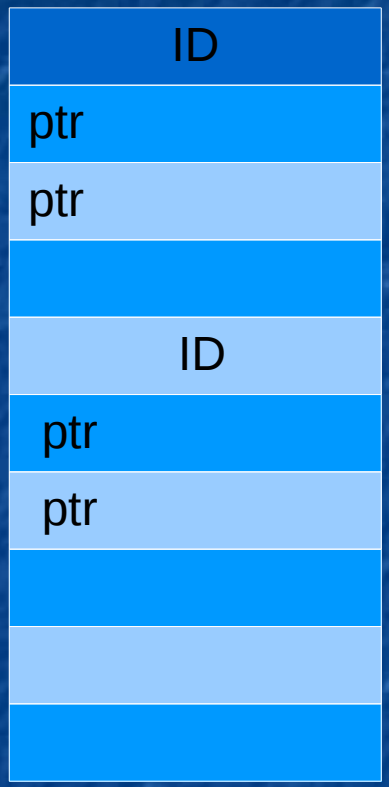



- Tabela de alocações: Lista indicando se a posição de memória é livre ou não

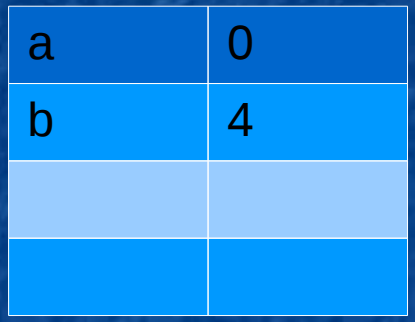
# Estrutura do programa

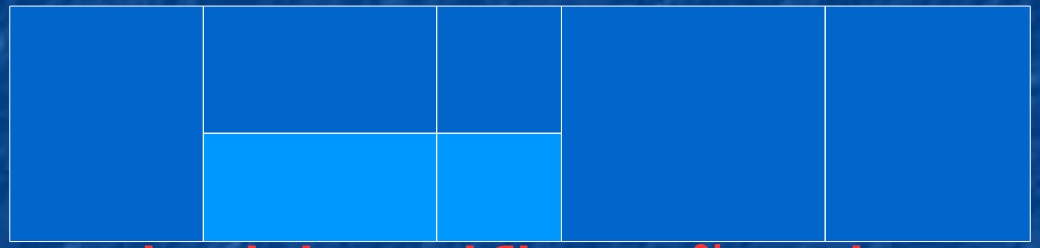

- Tabela de raizes: Contém os endereços das variaveis raizes do programa

# Estrutura do programa



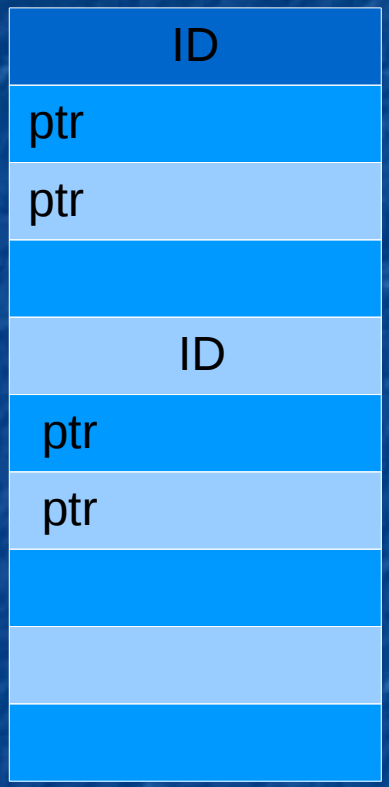




- ID: Cabeçalho de identificação do objeto

# Estrutura do programa




| T |
|---|
| T |
| T |
| F |
| F |
| F |
| F |
| F |
| F |
| F |

| a | 0 |
|---|---|
| b | 4 |
| | |
| | |

| Classe | atributo1 | Pos | Tamanho | Flag P |
|---|---|---|---|---|
| | atributo2 | Pos | | |

# Operação de alocação

classe : Tree

campos :  left,right,info

tipos : ptr,ptr,int

tamanho : 7

bytecodes:

root = tree();

root.left = tree();

root.right = tree()

# Operação de alocação

classe : Tree

campos :  left,right,info

tipos : ptr,ptr,int

tamanho : 7

bytecodes:

root = tree();

root.left = tree();

root.right = tree()

# Operação de alocação

classe : Tree

campos : left,right,info

tipos : ptr,ptr,int

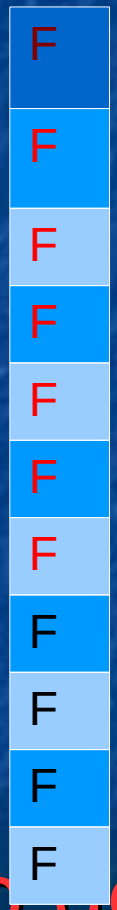
tamanho : 7

bytecodes:

root = tree();

root.left = tree();

root.right = tree()




- Procura um número específico de posições livres, definida pelo tamanho do classe

# Operação de alocação

classe : Tree

campos : left,right,info

tipos : ptr,ptr,int

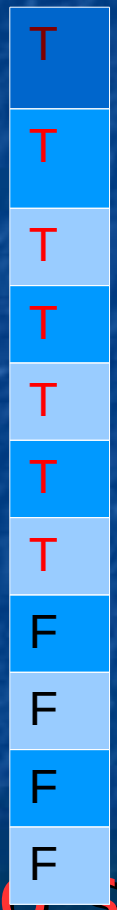
tamanho : 7

bytecodes:

root = tree();

root.left = tree();

root.right = tree()




- Seta as posições onde o objeto será alocado como True

# Operação de alocação


classe : Tree

campos : left,right,info

tipos : ptr,ptr,int

tamanho : 7

bytecodes:

root = tree();

root.left = tree();

root.right = tree()

- Criação do cabeçalho do Arquivo

# Operação de alocação


classe : Tree

campos : left,right,info

tipos : ptr,ptr,int

tamanho : 7


bytecodes:

root = tree();

root.left = tree();

root.right = tree()

- Nome da classe

# Operação de alocação


classe : Tree

campos : left,right,info

tipos : ptr,ptr,int

tamanho : 7


bytecodes:

root = tree();

root.left = tree();

root.right = tree()

- Atributos:

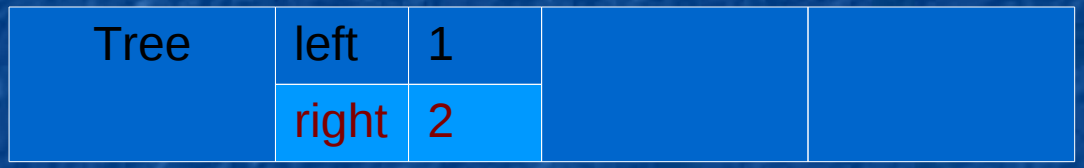
# Operação de alocação


classe : Tree

campos : left,right,info

tipos : ptr,ptr,int

tamanho : 7




bytecodes:

root = tree();

root.left = tree();

root.right = tree()

- Atributos:

# Operação de alocação


classe : Tree

campos : left,right,info

tipos : ptr,ptr,int

tamanho : 7


bytecodes:

root = tree();

root.left = tree();

root.right = tree()

- Atributos:

# Operação de alocação

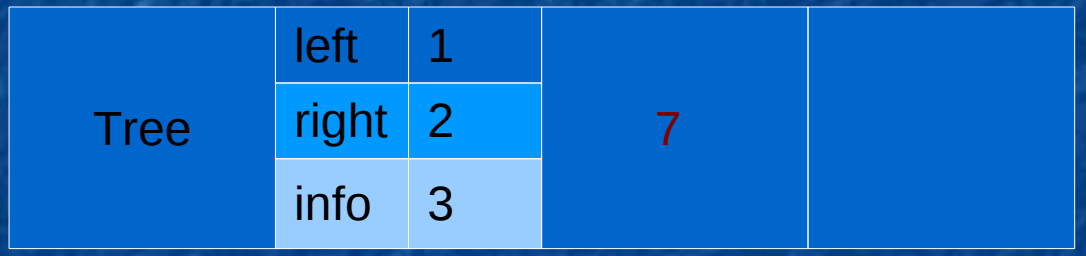
| T | T | T | T | T | T | T | F | F | F | F |
|---|---|---|---|---|---|---|---|---|---|---|

classe : Tree

campos : left,right,info

tipos : ptr,ptr,int

tamanho : 7




bytecodes:

root = tree();

root.left = tree();

root.right = tree()

- Tamanho:

# Operação de alocação

| T | T | T | T | T | T | T | F | F | F | F |
|---|---|---|---|---|---|---|---|---|---|---|

classe : Tree

campos : left,right,info

tipos : ptr,ptr,int

tamanho : 7

| Tree | left | 1 | 7 | False |
|---|---|---|---|---|
| | right | 2 | | |
| | info | 3 | | |

bytecodes:

root = tree();

root.left = tree();

root.right = tree()

- Setar False na flag de percorrimento: garante que a

# Operação de alocação

| T | T | T | T | T | T | T | F | F | F | F |
|---|---|---|---|---|---|---|---|---|---|---|

classe : Tree

campos : left,right,info

tipos : ptr,ptr,int

tamanho : 7

| Tree | left | 1 | 7 | False |
|---|---|---|---|---|
| | right | 2 | | |
| | info | 3 | | |

bytecodes:

root = tree();

root.left = tree();

root.right = tree()

- Este cabeçalho deve ser inserido primeira posição do objeto

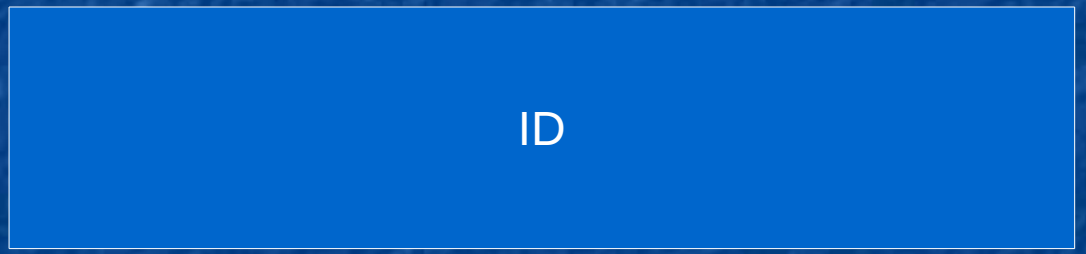
# Operação de alocação


classe : Tree

campos : left,right,info

tipos : ptr,ptr,int

tamanho : 7




bytecodes:

root = tree();

root.left = tree();

root.right = tree()

- Este cabeçalho deve ser inserido primeira posição do objeto

# Operação de alocação


classe : Tree

campos : left,right,info

tipos : ptr,ptr,int

tamanho : 7


bytecodes:

root = tree();

root.left = tree();

root.right = tree()

- Este cabeçalho deve ser inserido primeira posição do objeto

# Operação de alocação


classe : Tree

campos : left,right,info

tipos : ptr,ptr,int

tamanho : 7


bytecodes:

root = tree();

root.left = tree();

root.right = tree()

- Este cabeçalho deve ser inserido primeira posição do objeto

# Operação de alocação


classe : Tree

campos : left,right,info

tipos : ptr,ptr,int

tamanho : 7

bytecodes:

root = tree();

root.left = tree();

root.right = tree()



- Este cabeçalho deve ser inserido primeira posição do objeto

# Operação de alocação

classe : Tree

campos :  left,right,info

tipos : ptr,ptr,int

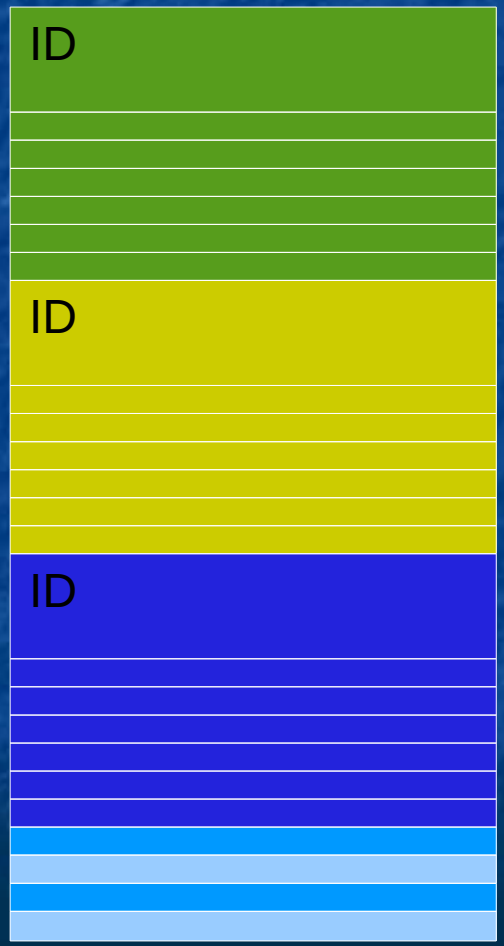
tamanho : 7

bytecodes:

root = tree();

root.left = tree();

root.right = tree()



# Operações de Referência

- Ao final de cada alocação na memória possuimos um objeto alocado e uma série de atributos que estão inicialmente nulos
- Esse objeto precisa ser referenciado por outro objeto ou estrutura
- Atributos do tipo ponteiro podem referenciar outros objetos

# Operações de Referência

- A = class();
  - Uma tabela de raizes precisa armazenar o endereço deste objeto alocado
- A.B = class();
  - O atributo B do objeto referenciado por A guarda um ponteiro para outro objeto que foi alocado em memória
- A = B;
  - O objeto referenciado por uma variável B será referenciado também por uma variável A . uma tabela de raizes também fará o apontamento.
- A.B = C;
  - O atributo B dentro do objeto referenciado por A guarda um ponteiro para um objeto referenciado por C

# Operações de referência

- Node = tree();
- Node.right = Tree();
- Root = Node;
- Root.left = Node2;

# Operações de referência

- Node = tree();
- Node.right = Tree();
- Root = Node;
- Root.left = Node2;

| Node | 0 |
|---|---|
| | |
| | |
| | |

| 0 | ID |
|---|---|
| 1 | |
| 2 | |
| 3 | |
| 4 | |
| 5 | |
| 6 | |

# Operações de referência

- Node = tree();
- Node.right = Tree();
- Root = Node;
- Root.left = Node2;

| Node | 0 |
|---|---|
| | |
| | |
| | |

| 0 | ID |
|---|---|
| 1 | “left” |
| 2 | “right” |
| 3 | “info” |
| 4 | “info” |
| 5 | “info” |
| 6 | “info” |

| 7 | ID |
|---|---|
| 8 | |
| 9 | |
| 10 | |
| 11 | |
| 12 | |
| 13 | |

# Operações de referência

- Node = tree();
- Node.right = Tree();
- Root = Node;
- Root.left = Node2;

| Node | 0 |
|---|---|
| | |
| | |
| | |

| 0 | ID |
|---|---|
| 1 | |
| 2 | |
| 3 | |
| 4 | |
| 5 | |
| 6 | |

| 7 | ID |
|---|---|
| 8 | |
| 9 | |
| 10 | |
| 11 | |
| 12 | |
| 13 | |

# Operações de referência

- Node = tree();
- Node.right = Tree();
- Root = Node;
- Root.left = Node2;

| Node | 0 |
|---|---|
| | |
| | |
| | |

| 0 | ID |
|---|---|
| 1 | |
| 2 | 7 |
| 3 | |
| 4 | |
| 5 | |
| 6 | |

| 7 | ID |
|---|---|
| 8 | |
| 9 | |
| 10 | |
| 11 | |
| 12 | |
| 13 | |

# Operações de referência

- Node = tree();
- Node.right = Tree();
- Root = Node;
- Root.left = Node2;

| Node | 0 |
|---|---|
| Root | 0 |
| | |
| | |

| 0 | ID |
|---|---|
| 1 | |
| 2 | 7 |
| 3 | |
| 4 | |
| 5 | |
| 6 | |

| 7 | ID |
|---|---|
| 8 | |
| 9 | |
| 10 | |
| 11 | |
| 12 | |
| 13 | |

# Operações de referência

- Node = tree();
- Node.right = Tree();
- Root = Node;
- Root.left = Node2;

| Node | 0 |
|---|---|
| Root | 0 |
| | |
| | |

| 0 | ID |
|---|---|
| 1 | X |
| 2 | 7 |
| 3 | |
| 4 | |
| 5 | |
| 6 | |

| 7 | ID |
|---|---|
| 8 | |
| 9 | |
| 10 | |
| 11 | |
| 12 | |
| 13 | |

# Operações de referência

- Node = Tree();
- Node.right = Tree();
- Root = Node;
- Root.left = Node2;

| Node | 0 |
|---|---|
| Root | 0 |
| Node2 | X |
| | |

| 0 | ID |
|---|---|
| 1 | X |
| 2 | 7 |
| 3 | |
| 4 | |
| 5 | |
| 6 | |

Nesse caso Node2 já foi previamente alocado por isso já deveria estar na tabela de referâncias

| 7 | ID |
|---|---|
| 8 | |
| 9 | |
| 10 | |
| 11 | |
| 12 | |
| 13 | |

# Operações de referência

- Node = Tree();
- Node.right = Tree();
- Root = Node;
- Root.left = Node2;

| Node | 0 |
|---|---|
| Root | 0 |
| Node2 | X |
| | |

| 0 | ID |
|---|---|
| 1 | X |
| 2 | 7 |
| 3 | |
| 4 | |
| 5 | |
| 6 | |

| 7 | ID |
|---|---|
| 8 | |
| 9 | |
| 10 | |
| 11 | |
| 12 | |
| 13 | |

# Varredura da memória

- Root.left.right = Tree();
- value = Root.left.info;
- Root.info = Node.info;
- A tabela de Referencias de raízes armazena ponteiros APENAS para as variáveis raízes.
- Não conseguimos acessar as posições que não são raizes diretamente
- Precisamos percorrer todos os objetos ate chegar onde precisamos

# Varredura de memória

- Root.left.right.info
-

| 0 | ID |
|---|---|
| 1 | |
| 2 | |
| 3 | |
| 4 | |
| 5 | |
| 6 | |

| 7 | ID |
|---|---|
| 8 | |
| 9 | |
| 10 | |
| 11 | |
| 12 | |
| 13 | |

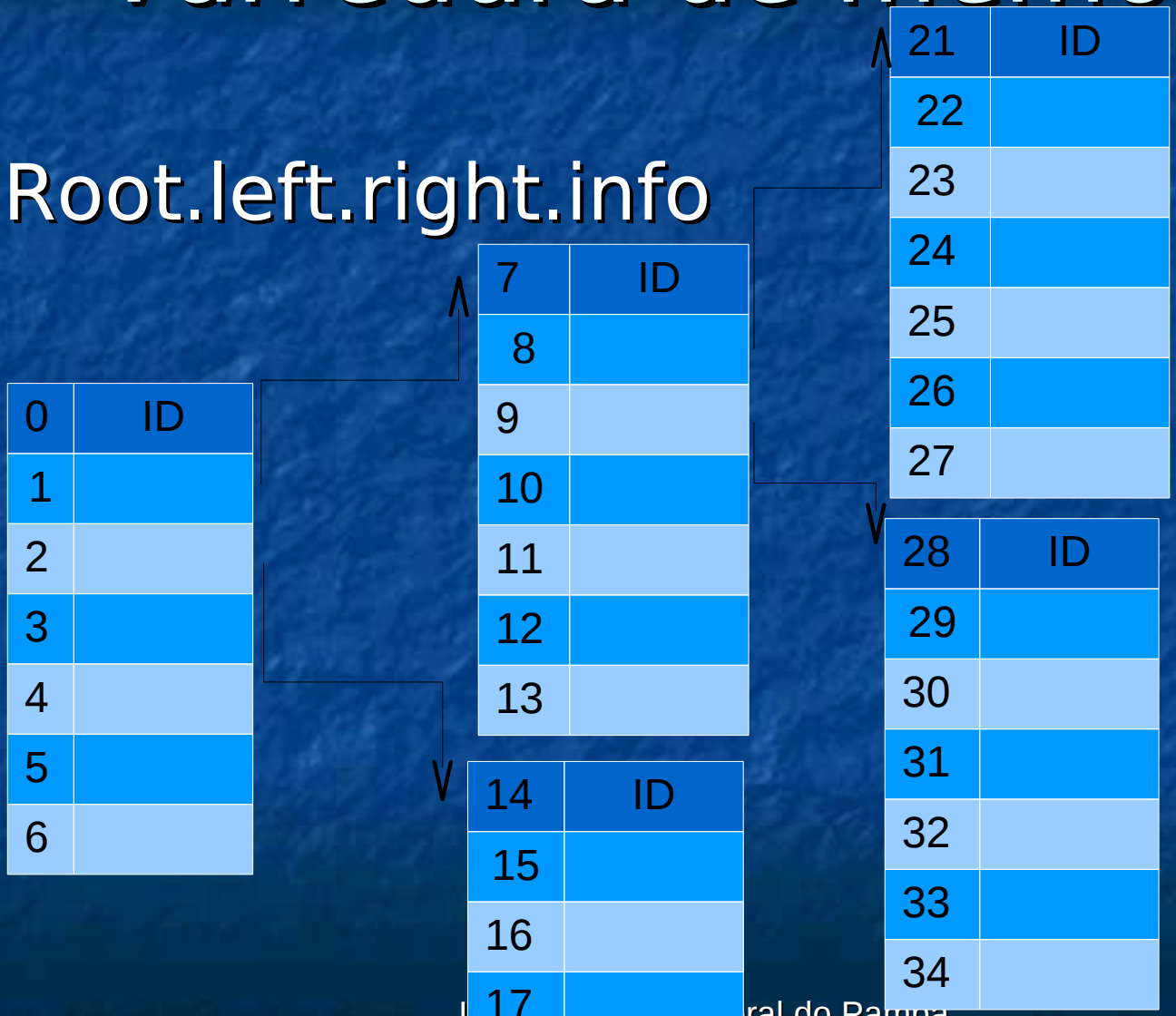
| 14 | ID |
|---|---|
| 15 | |
| 16 | |
| 17 | |

| 21 | ID |
|---|---|
| 22 | |
| 23 | |
| 24 | |
| 25 | |
| 26 | |
| 27 | |

| 28 | ID |
|---|---|
| 29 | |
| 30 | |
| 31 | |
| 32 | |
| 33 | |
| 34 | |

# Varredura de memória

- Root.left.right.info
-



# Varredura de memória

- Root.left.right.info
-

# Varredura de memória

- Root.left.right.info
-

# Garbage Collector

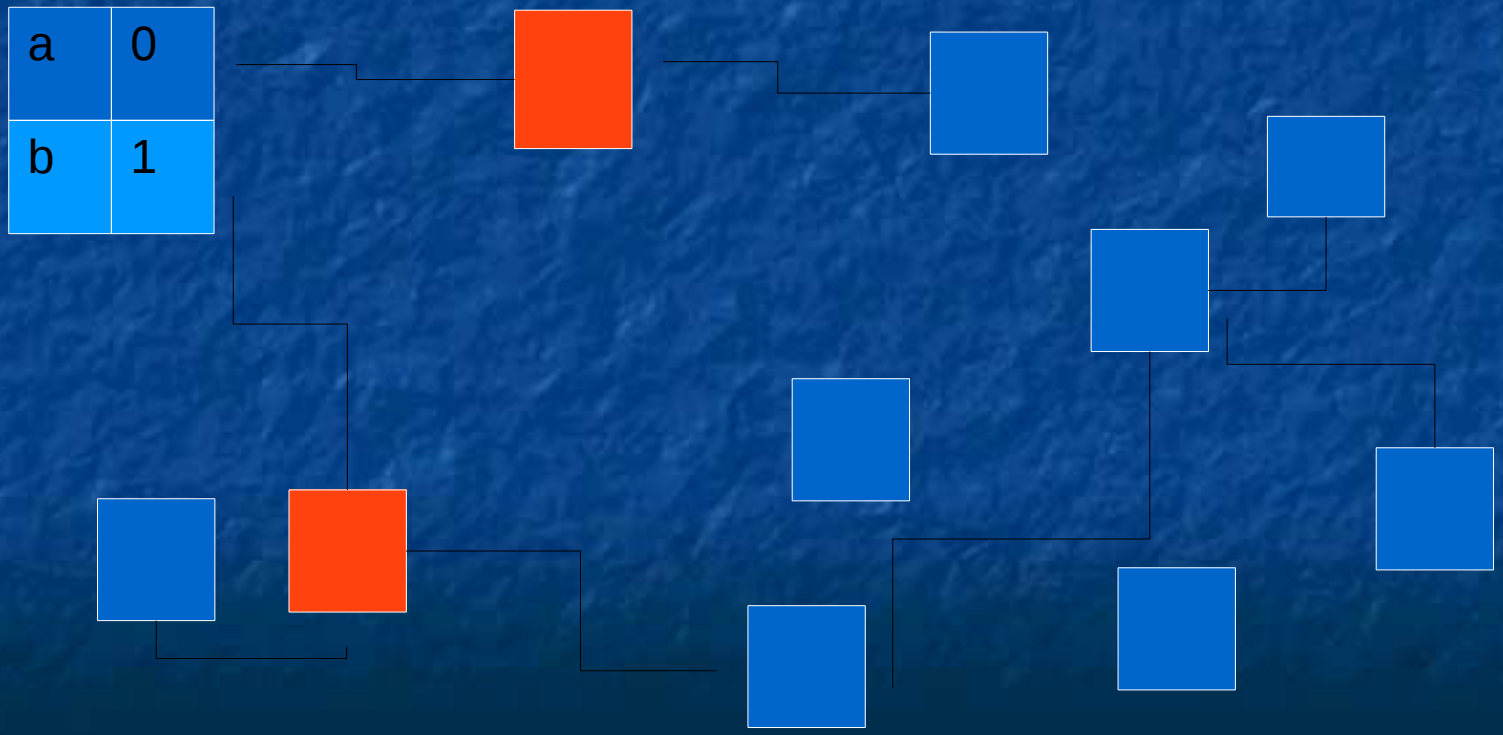
- Obj = list();
- Value = obj.info;
- Obj = Tree();
- O Objeto da classe list, alocado inicialmente, perde sua referência quando um novo objeto da classe Tree é alocado na mesma variável.

# Garbage Collector

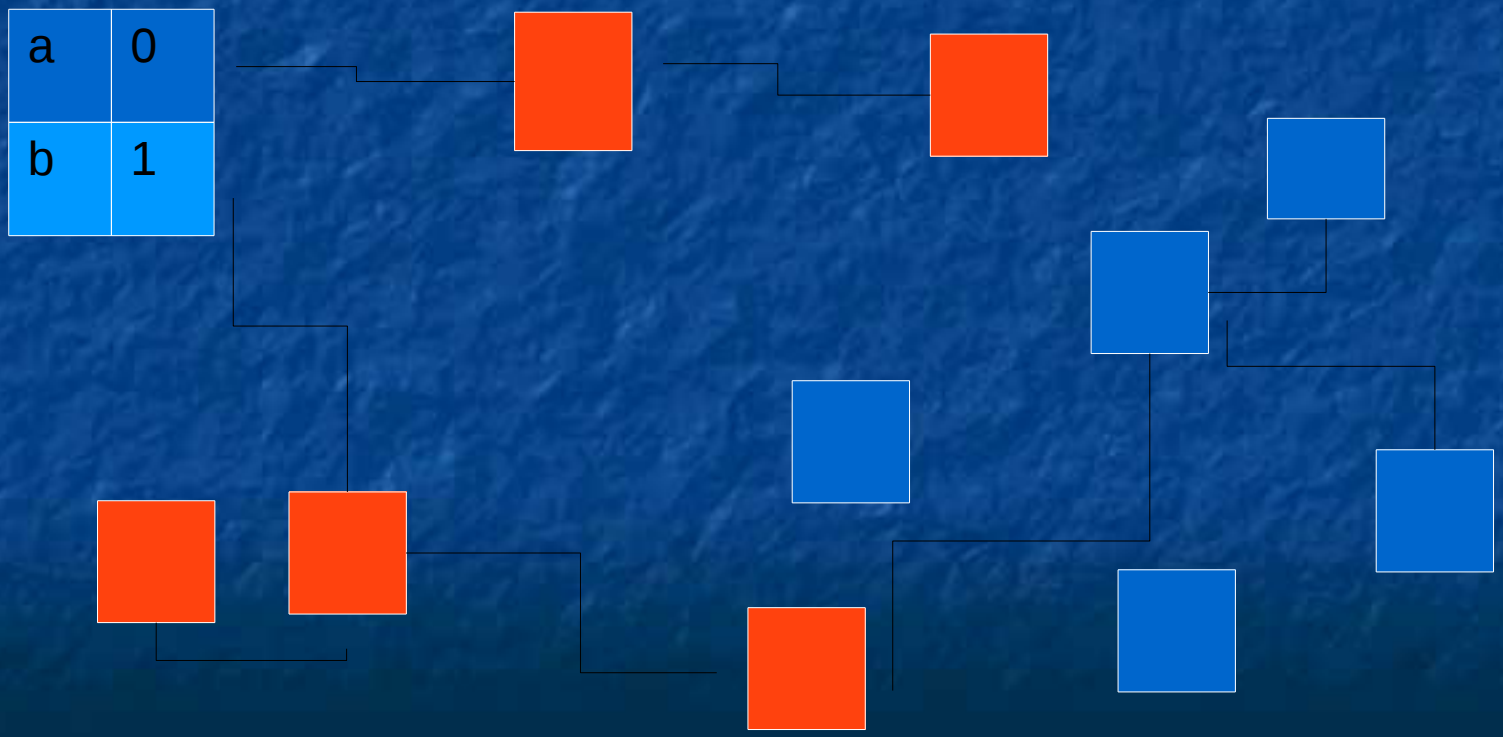
- Coletor de lixo que irá varrer a memória;
- Desalocará todos os objetos alocados e sem referência;
- Chamado após um número fixo de ciclos definido pelo usuário ou após a memória atingir uma porcentagem de alocação;

# Garbage Collector

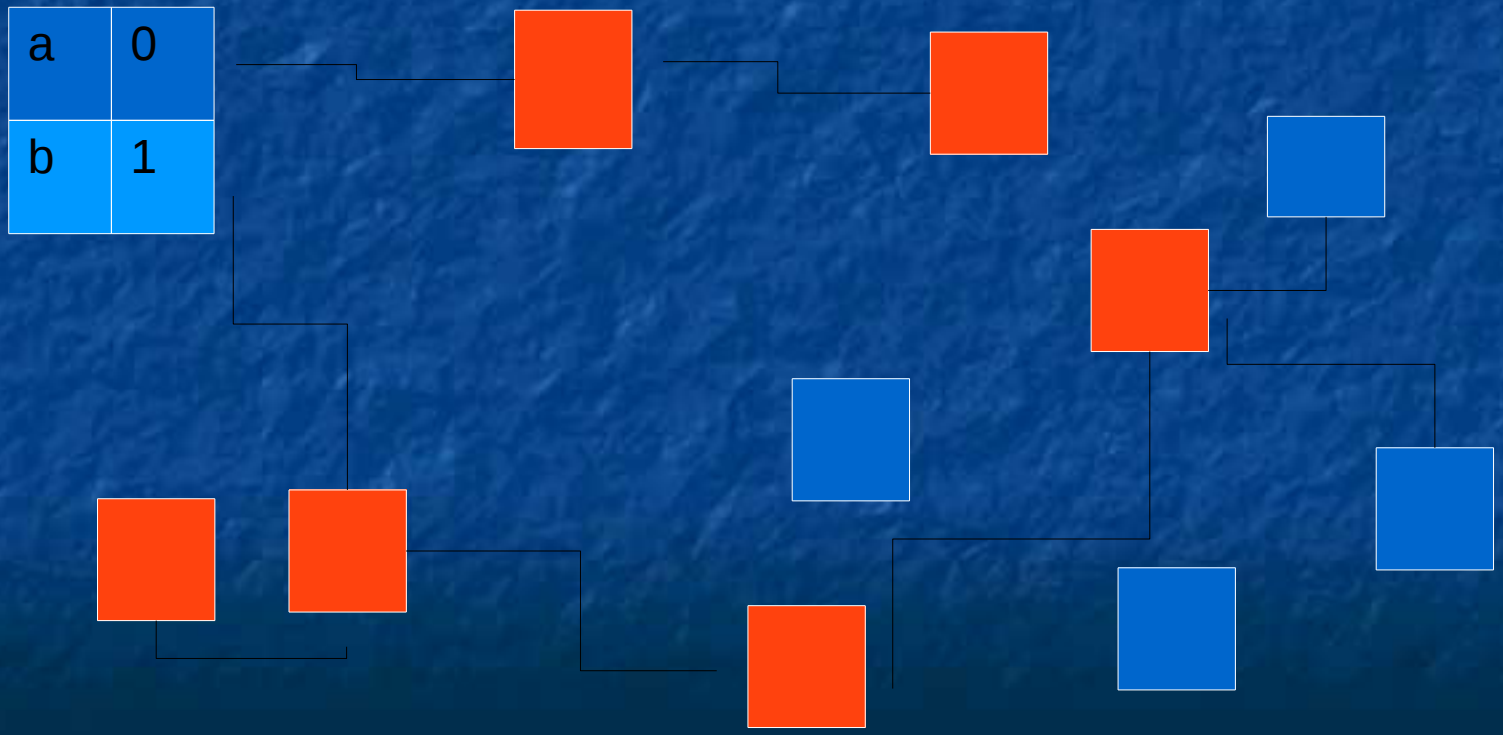
# Garbage Collector



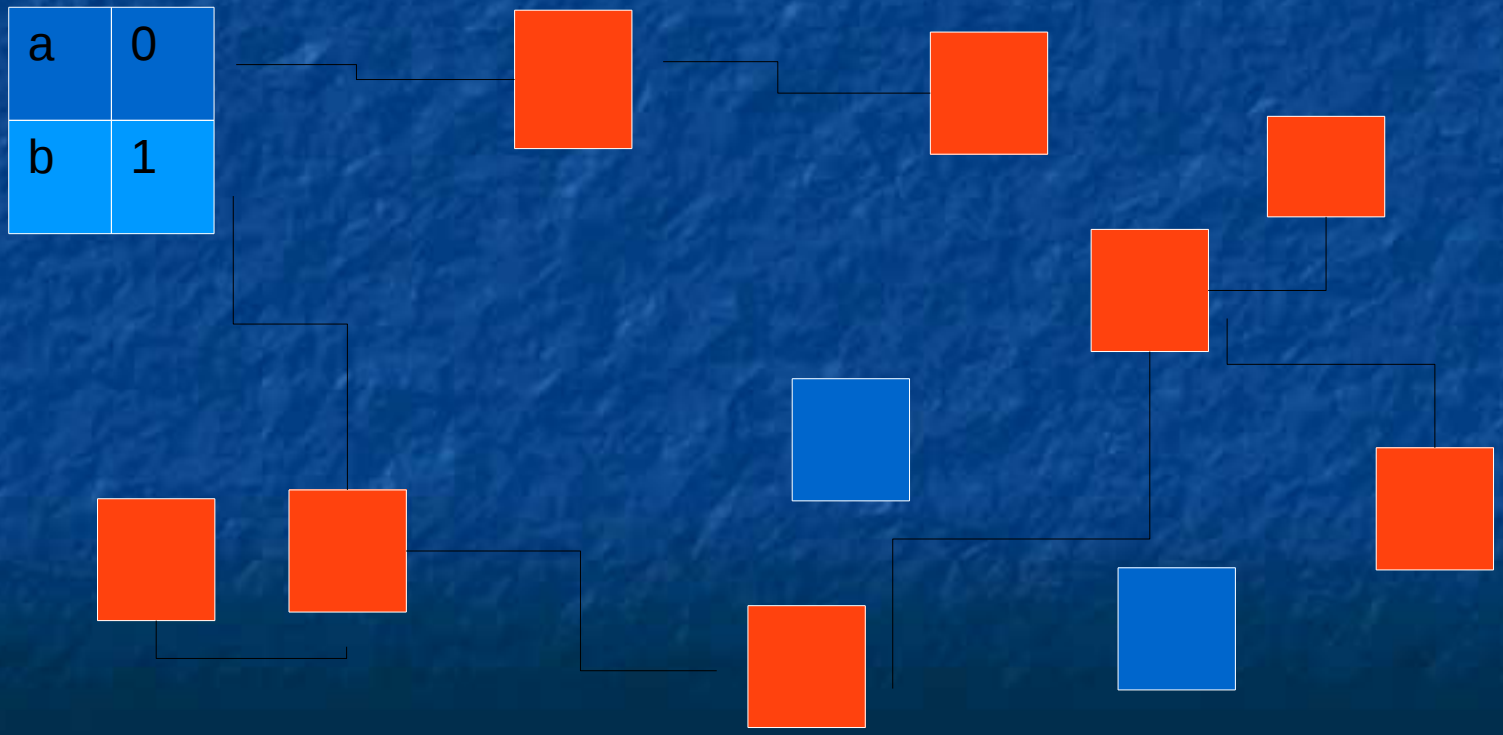
# Garbage Collector



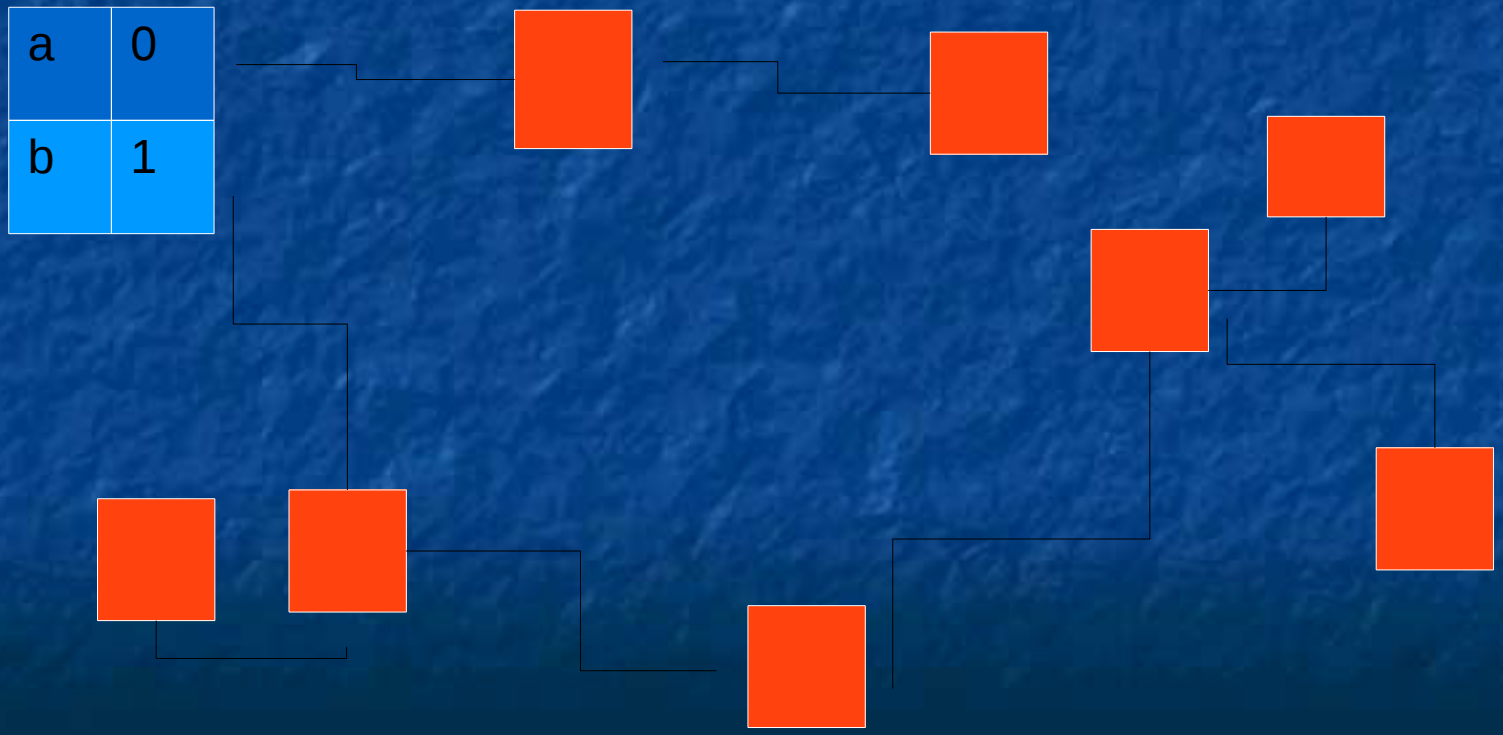
# Garbage Collector



# Garbage Collector



# Garbage Collector



# Conclusão

- Esse projeto trouxe grandes desafios, pois ao mesmo tempo em que tivemos que implementar o gerenciador, tivemos também que buscar informações sobre o funcionamento do algoritmo Mark-and-Sweep. Outro desafio encontrado foi a aprendizagem na linguagem de programação Python.

# Referências

1 - Barros, Alexandra, Uma análise da evolução da coleta de lixo distribuida;

2 - Caelum, Arquitetura e Design de Software;

3 - Tostes, Ramiro, Estudo do ambiente Java no contexto de desenvolvimento de jogos;

4 – Neves, Bruno, Gerência Dinâmica de Memória;