Universidade Autónoma de

Lisboa

Engenharia Informática

Paradigmas de Programação

Gecco : Dynamic Stacking Optimization in Uncertain Environments

Docente: Adrian Dediu

12 de junho de 2023

Introdução

Nesta simulação é fornecido um ambiente dinâmico que representa um cenário de empilhamento mais complexo e realista. É caracterizado por três tipos de pilhas chamadas Arrival, Buffer e Handover, bem como dois guindastes que realocam os blocos entre as pilhas.

O nosso propósito fundamental no processo de desenvolvimento consiste em cumprir três objetivos essenciais, e caso não sejamos capazes de os realizar, temos de minimizar o mais que pudermos os erros que possam ocorrer.

Objetivo #1 (min): O primeiro objetivo é evitar atrasos no processo upstream.

Para evitar tal estado, existe uma gama de pilhas de buffer que podem ser usadas para armazenar blocos por algum tempo. Os blocos só podem sair do sistema quando atingirem uma *Data de Pronto*, mas devem sair antes de sua *Data de Vencimento*. Os blocos devem permanecer na pilha de chegada (com o risco de bloquear o processo upstream) ou em uma pilha de buffer. Para remover um bloco, ele deve ser colocado na pilha de transferência, após o que a transferência fica indisponível por algum tempo.

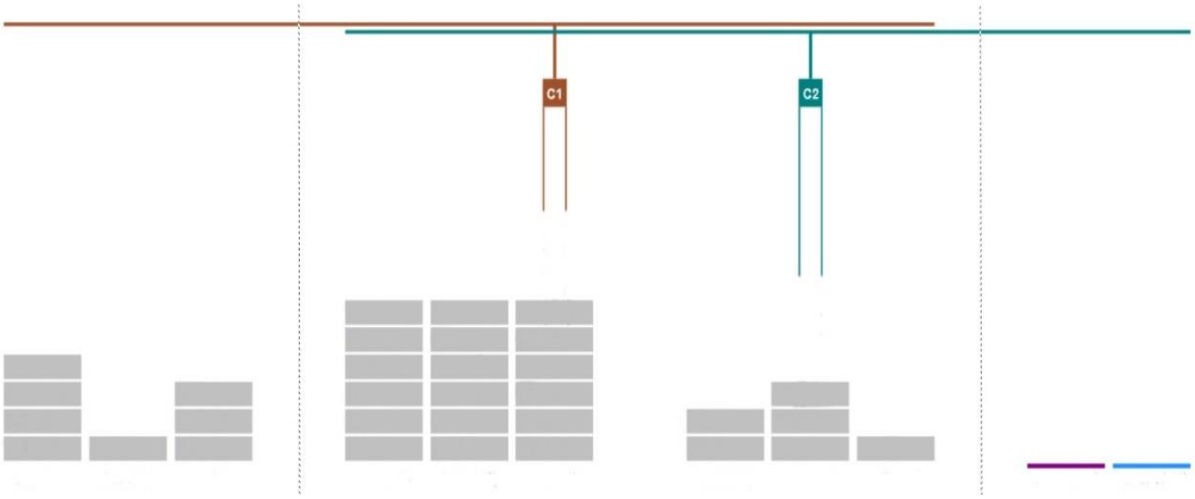
Objetivo nº 2 (máximo): O segundo objetivo é entregar o máximo de blocos possível no prazo.

Um guindaste está disponível para realizar realocações de blocos dentro das pilhas. Este guindaste pode pegar um bloco de cada vez das pilhas de chegada ou de buffer e deixá-lo em um buffer ou na pilha de transferência. Essa realocação é normalmente composta de quatro etapas:
(1) mover para a pilha onde um bloco deve ser recolhido
(2) pegar o bloco mais alto da pilha
(3) mover para a pilha onde o bloco deve ser descartado
(4) solte o bloco no topo da pilha.
Neste guindaste, o guindaste está sempre disponível e executa todos os pedidos com precisão perfeita. No entanto, seu uso deve ser minimizado para economizar energia e reduzir o desgaste.

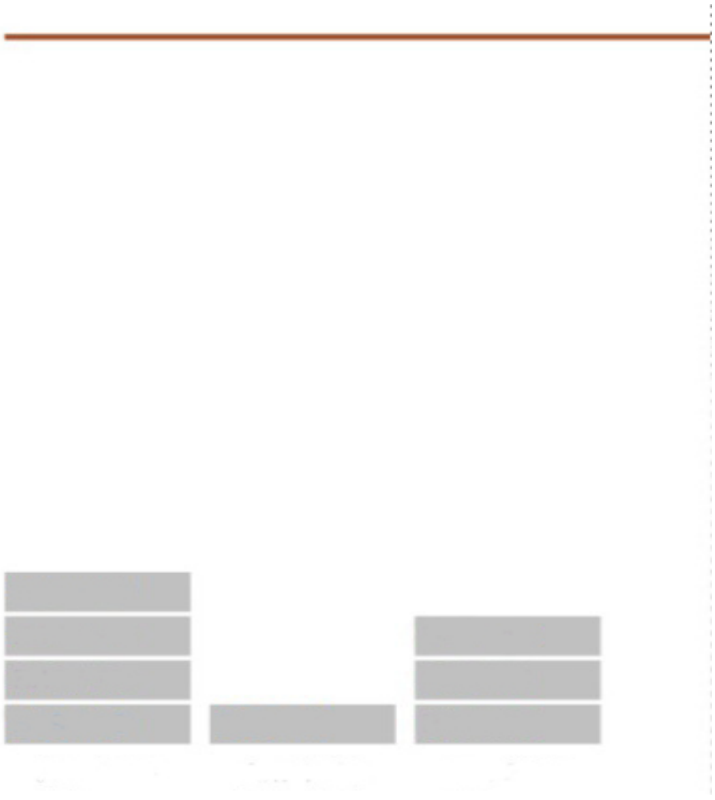
Objetivo nº 3 (min): O terceiro objetivo é realizar o menor número possível de operações com o guindaste.

Desenvolvimento do Gecco

O nosso código tem como função principal receber os containers no local de chegada (arrival), transferi-los para o buffer, onde se realiza o manejo dos containers de acordo com as suas características, para que possamos movê-los ao handover de forma eficiente e otimizada, entregando o máximo de blocos possível dentro do prazo estabelecido.

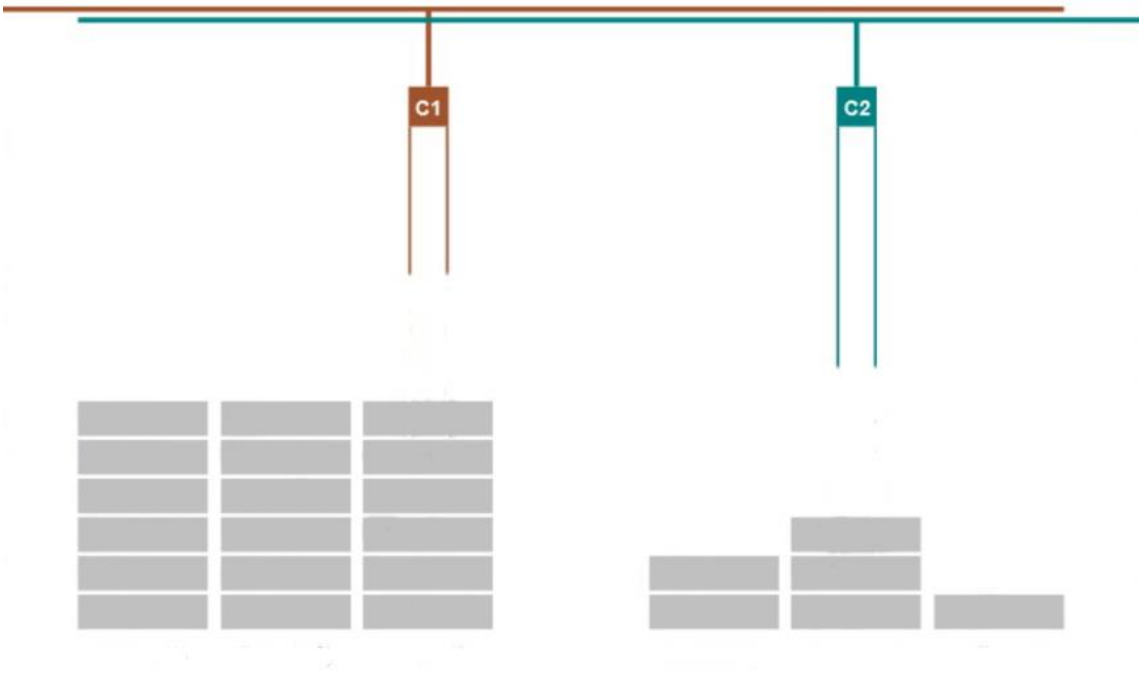


- **<u>Arrival</u>**



O Arrival designa o local de chegada dos containers, onde existem três colunas com uma altura máxima de quatro containers por coluna, não havendo a possibilidade de as três colunas ficarem cheias devido ao atraso na entrega ao buffer.

- **Buffer**

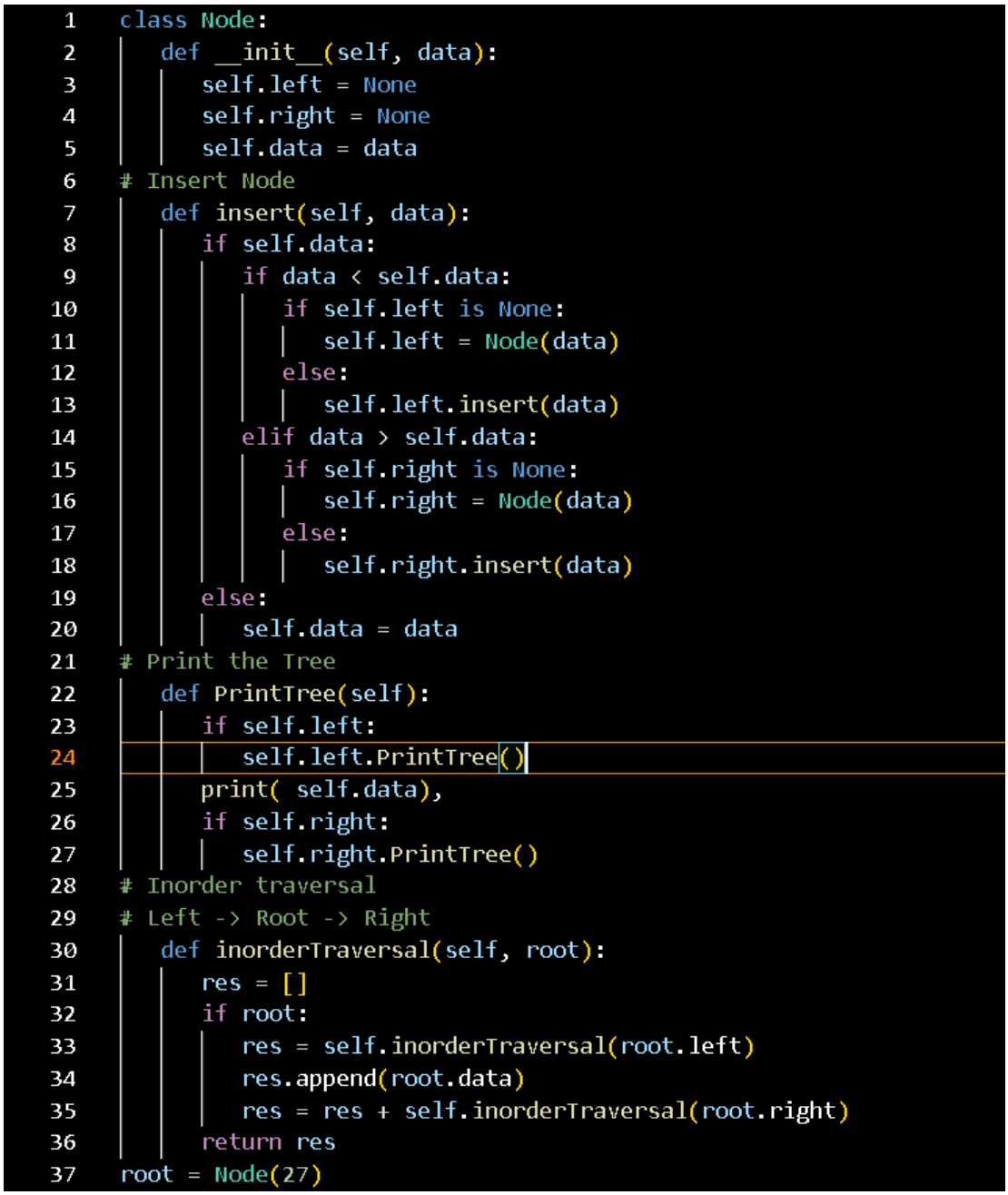


O buffer é o local onde vamos receber os containers e assim organizar para que eles sejam entregues no tempo certo, pois cada um tempo o seu tempo de entrega, uns sendo menor que os outros, a grande maioria diferente dos outros.

O buffer é constituído por 6 colunas e cada uma delas com 8 containers de altura máxima.

As duas primeiras colunas do buffer vamos utilizar para que podemos receber os containers e assim organizá-los, assim que uma estiver cheia começaremos a utilizar a outra, e assim não congestionamos o arrival.

O exemplo do código que vamos utilizar para o desenvolver da organização dos containers é esse mostrado abaixo, onde representa uma arvore binaria.



```
class Node:
    def __init__(self, data):
        self.left = None
        self.right = None
        self.data = data
# Insert Node
    def insert(self, data):
        if self.data:
            if data < self.data:
                if self.left is None:
                    self.left = Node(data)
                else:
                    self.left.insert(data)
            elif data > self.data:
                if self.right is None:
                    self.right = Node(data)
                else:
                    self.right.insert(data)
        else:
            self.data = data
# Print the Tree
    def PrintTree(self):
        if self.left:
            self.left.PrintTree()
        print( self.data),
        if self.right:
            self.right.PrintTree()
# Inorder traversal
# Left -> Root -> Right
    def inorderTraversal(self, root):
        res = []
        if root:
            res = self.inorderTraversal(root.left)
            res.append(root.data)
            res = res + self.inorderTraversal(root.right)
        return res
root = Node(27)
```

O Node da linha 37 representa o container central que ira ser dado na implementação de um código matemático que representa a media dos tempos dos containers.

```
37    root = Node(27)
38    root.insert(14)
39    root.insert(35)
40    root.insert(10)
41    root.insert(19)
42    root.insert(31)
43    root.insert(42)
44    print(root.inorderTraversal(root))
45
```

Feito a media do tempo dos containers vai ser realizado a introdução dos outros tempos onde vai ser inserido pelo código nos **root.insert()**.

```
[10, 14, 19, 27, 31, 35, 42]
PS C:\Users\edu07>
```

O output do código ira ser impresso em modo inorder, para que assim podemos entregar ao handover primeiro os containers que tiverem o menos tempo.

- **Handover**

O handover corresponde ao local de entrega, onde existem duas colunas, mas tal como a arrival, o handover possui uma altura máxima de quatro containers por coluna.