Sistema Operativos



LEI

Turma 3

Nome: Gonçalo Vieira Número: 202100296 Nome: Francisco Silva Número: 202100984 Nome: Rui Barroso Número: 202100299

Índice

- 1. Sumario
- 2. Funcionamento
- 3. Resultados e observações

Sumário

No âmbito da disciplina de Sistemas operativos, foi nos pedido para o desenvolvimento de um programa capaz de resolver o problema do Knapsack de modo aleatório, de modo paralelo e concorrencial utilizando várias threads.

Funcionamento

Nós desenvolvemos 2 versões de resolver este problema: knapsack_B(Base); knapsack_A(Avançada);

knapsack_B(Base) Versão base pedida no enunciado

knapsack_A(Avançada)

Versão avançada pedida no enunciado

Observação: Esta versão é a mais lenta devido aos sinais, sincronização e a necessidade de restringir o acesso aos resultados para a sua atualização.

Execução

Menu

O menu pode se escolher entre 2 versões, a base e a avançada, sendo a base a predefinida.

Quando a versão avançada é escolhida é pedido a percentagem das interrupções. Depois de escolher a versão é preciso colocar o comando e clicar no enter.

```
************Menu***********

Versão atual - Avançada de 50% em 50%

1 - Mudar Para Versão Base

2 - Mudar Para Versão Avançada

S - Sair

Comando - (ex: "kp ex23.txt 10 60")Opção>
```

Resultados e observações

Observações:

Quando o valor ótimo foi calculado com 0 iterações significa que esse valor é o lower bound inicial.

Apesar da escolha de soluções no Beam Search ser aleatório o nº de iterações é sempre igual entre todos os testes para o mesmo exercício.

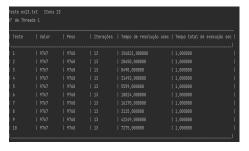
```
public List<KnapBag> selectSolutions(List<KnapBag> solutions){
    int numberOfSoulutions = NUMBER_OF_ITEMS/2;
    if (solutions.size()
    return solutions;
}
List<KnapBag> bestSolutions = new ArrayList<>();

int solutionsIndexes = -1;
for(int i=0;isnumberOfSoulutions;i++){
    solutionsIndexes = ((int)(Math.randam() *188)) % solutions.size();
    KnapBag bag = solutions.get(solutionsIndexes);
    if(!bestSolutions.contains(bag)){
        bestSolutions.add(bag);
    }
}
return bestSolutions;
```

cap: Metodo de responsavel por escolher as soluçõe

A escolha das soluções é feita aleatoriamente atravez:

```
((int)(Math.random() *100)) % solutions.size()
```



cap:Resultado



cap: nº de iterações de todas as atualizações do resultado local.

Possível explicação:

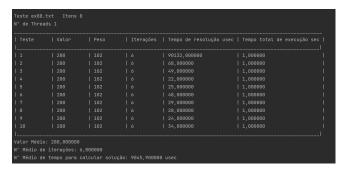
Tendo uma mochila com 10 itens ordenada decrescentemente por valor/peso Mochila ótima

1111101000

Só poderá ser calculada na iteração 7 que é quando são colocados os 7 primeiros itens na mochila.

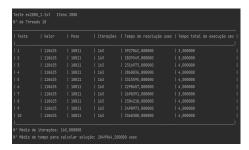
ex:

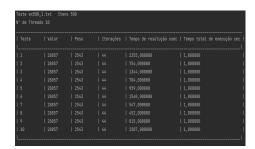
Na mochila do enunciado: v = {15, 100, 90, 60, 40, 15, 10, 1} w = { 2, 20, 20, 30, 40, 30, 60, 10} O valor ótimo é 1 1 1 1 0 1 0 0 e o nº de iterações é 6 que corresponde ao 6 elemento, o último a ser colocado.

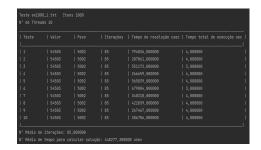


Resultados:

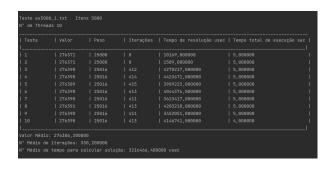
O programa até ao exercício 2000_1.txt foi capaz de encontrar o valor da mochila ótimo.

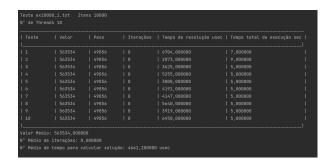






No exercicio 5000_1.txt e 10000_1.txt o programa não conseguiu calcular o valor ótimo apenas uma aproximação





Base vs Avançada 50 em 50 vs Avançada 25 em 25 Sem diferenças de performance.

Base Avançada 50 em 50

Avançada 25 em 25

