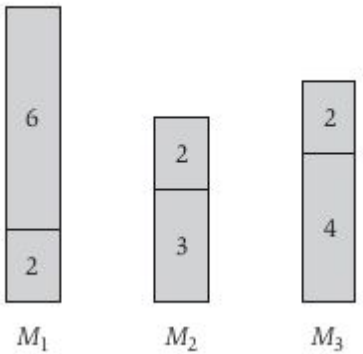
Escalonamento de Tarefas

Trabalho 3 - Análise de Algoritmos

Jean Ferreira
Nágela Machado
Vitor Hugo Honorato Tiago

Definição do Problema

- Vários servidores precisam executar uma lista de tarefas
- Todos servidores são iguais
- Existem ***m*** servidores ordenados de $M_1, \dots, M_m$
- Existem ***n*** tarefas a ser executadas, $n_1, \dots, n_n$
- Cada tarefa ***j*** leva um tempo de processamento t_j



- Todas as tarefas devem ser atribuídas as máquinas
- Cada máquina ao fim tem uma carga que é o somatório do tempo das tarefas atribuídas aquela máquina
- O objetivo é minimizar a maior carga, chamada de makespan.

Algoritmo Ótimo

- Analisa todas as combinações possíveis de máquinas e tarefas
- Utiliza programação dinâmica para guardar todas as organizações de tarefas possíveis.

```python
def loadBalancingDynamicProgramming(machines, tasks):
    dp = {}
    dp[(0,) * machines] = 0

    for task in tasks:
        newDp = {}
        for state in dp:
            for i in range(machines):
                newState = list(state)
                newState[i] += task
                newState = tuple(newState)
                newDp[newState] = max(newState)
        dp = newDp

    return min(max(state) for state in dp.keys())
```

Algoritmo Ótimo

- Complexidade Assintótica de tempo do algoritmo ótimo:
  - O algoritmo itera sobre cada tarefa O(n).
  - Para cada tarefa, ele itera sobre todos os estados armazenados em dp. No pior caso, pode haver $O(S^m)$ estados.
  - Para cada estado, ele itera sobre todas as máquinas O(m).
  - Dentro dessas iterações, ele realiza operações de complexidade O(m) para encontrar a máquina com maior load em *max(newState)*
  - $O(n \cdot m^2 \cdot S^m)$

- Complexidade de espaço do algoritmo ótimo:
  - O algoritmo armazena dp e newDp. Cada lista pode ter tamanho máximo $O(S^m)$
  - $O(2S^m) = O(S^m)$

Algoritmo Aproximado

```python
def loadBalancingGreedy(machines, tasks):
    tasks.sort(reverse = True)
    machinesTimes = [0] * machines

    for task in tasks:
        minCost = machinesTimes[0]
        machine = 0
        for i in range(1, machines):
            if machinesTimes[i] < minCost:
                minCost = machinesTimes[i]
                machine = i

        machinesTimes[machine] += task

    return max(machinesTimes)
```

- O algoritmo tem uma abordagem gulosa
- Ordena as tarefas em ordem decrescente
- Para cada tarefa ele acha a máquina que tem a menor carga e atribui a ela a tarefa

Sendo m o número de máquinas e n o número de tarefas:

- A complexidade de ordenar as tarefas é O(n log n)
- O algoritmo itera sobre cada tarefa: O(n)
- Dentro dessa iteração ele itera sobre as máquinas para alcançar a carga mínima O(m - 1)
- Logo o tempo total do algoritmo é O(n logn + n · (m - 1))

Algoritmo Aproximado

- O algoritmo não é ótimo pois falha em encontrar a melhor solução em alguns casos
- Contra exemplo em que o algoritmo falha:
  - m = 3
  - tarefas = [8, 7, 6, 5, 4, 3, 2, 1]

Solução encontrada

M_1 = [8, 3, 2] = 13
M_2 = [7, 4, 1] = 12
M_3 = [6, 5] = 11

Makespan = 13

Solução ótima

M_1 = [8, 4] = 12
M_2 = [7, 5] = 12
M_3 = [6, 3, 2, 1] = 12

Makespan = 12

Razão de Aproximação

Suponha:

T - tempo encontrado pelo Algoritmo Aproximado
T* - tempo encontrado pelo Algoritmo Ótimo

Vamos encontrar limites inferiores para T*:

Limite 1: $T^* \geq \frac{1}{m} \sum_j t_j$

Limite 2:
Se n ≤ m T = T*, pois cada tarefa tem sua máquina
Se n > m:

A máquina M_i tem pelo menos duas tarefas
A tarefa t_{m+1} é a segunda tarefa nessa máquina
Como o vetor está ordenado

$$t_{m+1} \leq t_m \qquad T^* \geq t_m + t_{m+1} \qquad T^* \geq 2 \cdot t_{m+1}$$

Vamos chamar a máquina que tem a menor carga T de M_i, essa máquina receberá a última tarefa. Suponha t_j a última tarefa a ser adicionada em M_i

$$j \geq m+1 \qquad t_j \leq t_{m+1}$$

Se a última tarefa t_j foi atribuída a M_i essa máquina tinha a menor carga antes de j, ou seja: $T_i - t_j$. Então, cada máquina tinha no mínimo essa carga, logo:

$$m \cdot (T_i - t_j) \leq \sum_k t_k$$

$$T_i - t_j \leq \frac{1}{m} \sum_k t_k$$

Razão de Aproximação

Limite 1 $T^* \geq \frac{1}{m} \sum_j t_j$

Limite 2 $T^* \geq 2 \cdot t_{m+1}$ $\frac{1}{2} T^* \geq t_{m+1}$

$j \geq m+1$ $t_j \leq t_{m+1}$

$m \cdot (T_i - t_j) \leq \sum_k t_k$

$T_i - t_j \leq \frac{1}{m} \sum_k t_k$

Usando o Limite 1, temos:

$$T_i - t_j \leq T^*$$

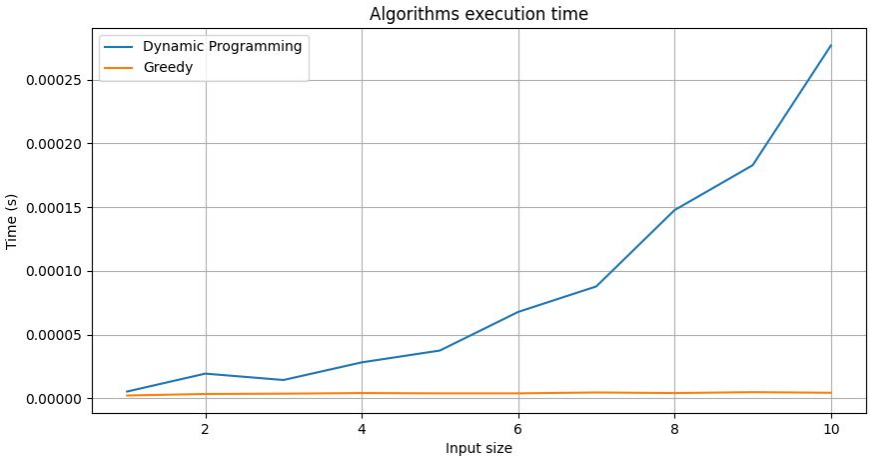
Somando t_j do lado direito da equação podemos somar algo maior ou igual a t_j do outro lado da equação, então usando o desenvolvimento do limite 2 temos:

$$(T_i - t_j) + t_j \leq T^* + t_{m+1}$$

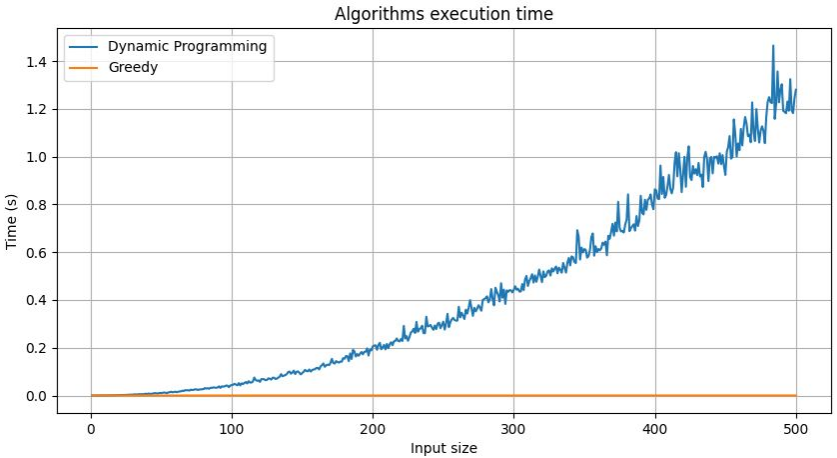
$$(T_i - t_j) + t_j \leq T^* + \frac{1}{2} T^*$$

$$T_i \leq \frac{3}{2} T^*$$

Execuções



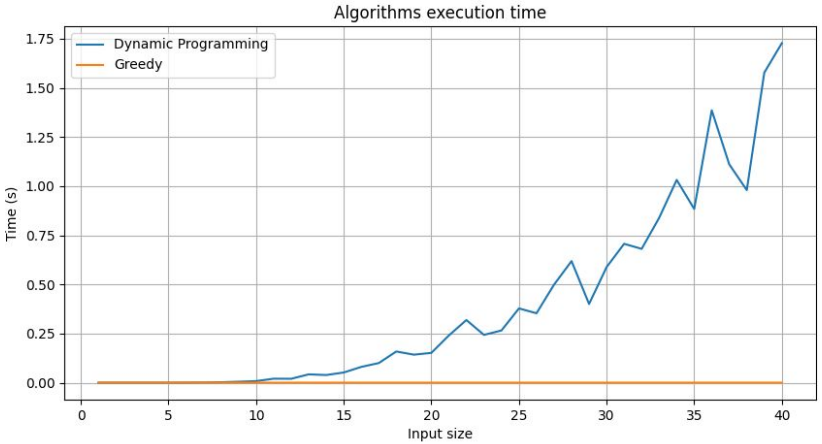
2 máquinas com 10 tarefas



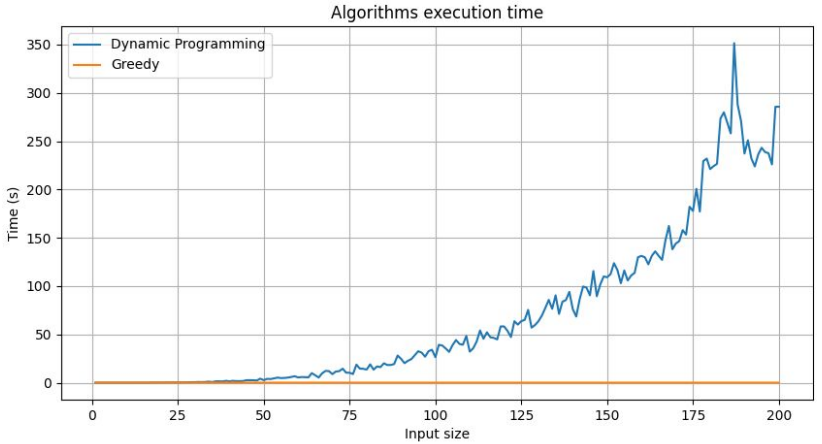
2 máquinas com 500 tarefas

Quantidade de máquinas fixas e número de tarefas variando
Tarefas com custos aleatórios entre 1 e 20
Complexidade ótimo: $O(n \cdot 4 \cdot S^2) = O(nS^2)$
Complexidade aproximado: $O(n \log n + 2n) = O(n \log n)$

Execuções



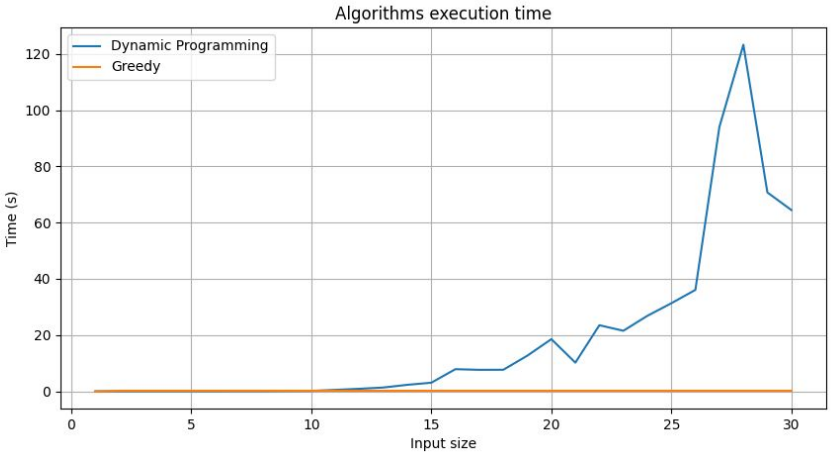
3 máquinas com 40 tarefas



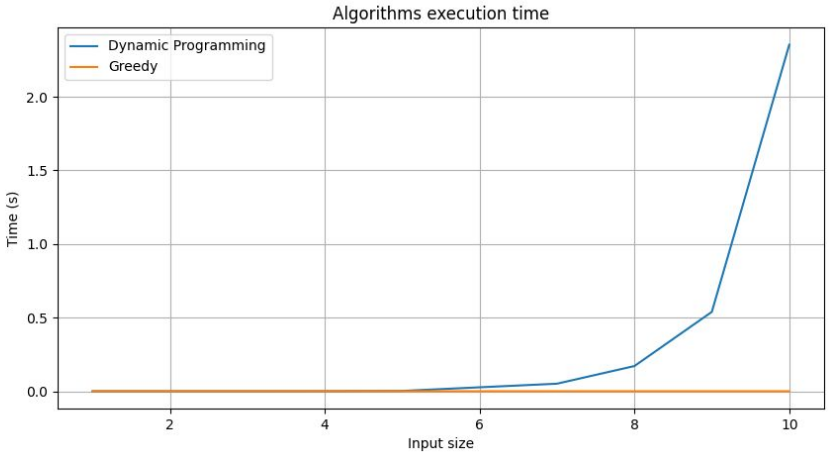
3 máquinas com 200 tarefas

Quantidade de máquinas fixas e número de tarefas variando
Tarefas com custos aleatórios entre 1 e 20
Complexidade ótimo: $O(n \cdot 9 \cdot S^3) = O(nS^3)$
Complexidade aproximado: $O(n \log n + 3n) = O(n \log n)$

Execuções



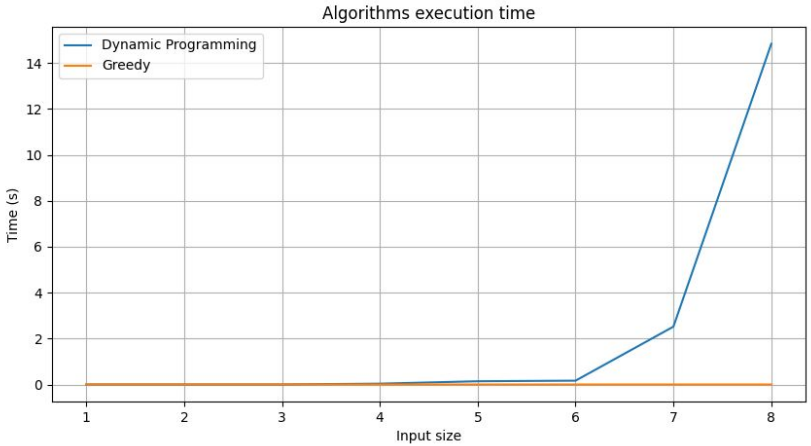
4 máquinas com 30 tarefas



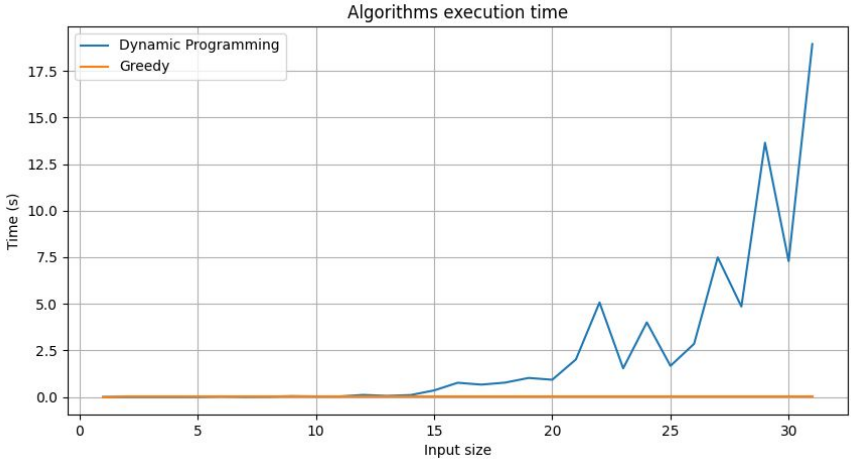
5 máquinas com 10 tarefas

Quantidade de máquinas fixas e número de tarefas variando
Tarefas com custos aleatórios entre 1 e 20
Complexidade ótimo: $O(n \cdot 16 \cdot S^4)$ e $O(n \cdot 25 \cdot S^5) = O(nS^4)$ e $O(nS^5)$
Complexidade aproximado: $O(n \log n + 4n)$ e $O(n \log n + 5n) = O(n \log n)$

Execuções



10 jobs variando as máquinas até 9



5 jobs variando até 31 máquinas

Quantidade de tarefas fixas e número de máquinas variando
Tarefas com custos aleatórios entre 1 e 5
Complexidade ótimo: $O(10 \cdot m^2 \cdot S^m)$ e $O(5 \cdot m^2 \cdot S^m)$= S é $10 \cdot 5$ e S é $5 \cdot 5$, logo $O(50^m)$ e $O(25^m)$
Complexidade aproximado: O(10 log 10 + 10m) e O(5 log 5 + 5m) = O(m)

Obs: Mesmo com apenas 5 jobs e 31 máquinas, o algoritmo ótimo de programação dinâmica calcula todas as possibilidades

Execuções

Comparação do resultado do algoritmo ótimo e do algoritmo aproximado para 200 execuções:

Máquinas (m)	Tarefas (n)	Taxa de acerto (%)	Maior erro
2	20	84	1,0117
2	50	99	1,0036
2	100	100	0
2	200	100	0
3	20	59.5	1,0434
3	50	93	1,0062
3	100	100	0

Os custos das tarefas foram geradas aleatoriamente com valores entre 1 e 20.

Conclusões

- O Algoritmo ótimo tem tempo de execução exponencial em relação a m
  - $O(n \cdot m \cdot S^m)$, considerando n constante temos $O(m \cdot c^m)$
- O Algoritmo ótimo tem tempo de execução polinomial em relação a n
  - $O(n \cdot m \cdot S^m)$, considerando m constante temos $O(n \cdot n^c)$
- O Algoritmo ótimo tem complexidade de espaço adicional $O(S^m)$ por usar programação dinâmica

- O tempo de execução do algoritmo aproximado é linear relativo a m
  - $O(n \log n + n \cdot m)$, considerando n constante temos $O(c \log c + cm) = O(m)$
- O tempo de execução do algoritmo aproximado é n log n relativo a n
  - $O(n \log n + n \cdot m)$, considerando m constante temos $O(n \log n + nc) = O(n \log n)$
- O algoritmo aproximado erra mais com o aumento da quantidade de máquinas
- O algoritmo aproximado tem uma taxa de acerto maior com uma altas quantidades de tarefas

Referências

- KLEINBERG, Jon; TARDOS, Éva. **Algorithm Design**. Boston: Addison-Wesley, 2006.

- CORMEN, T. H.; LEISERSON, C. E.; RIVEST, R. L.; STEIN, C. *Algoritmos: Teoria e Prática*. Rio de Janeiro: Elsevier, 2002.