PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS

Bacharelado em Engenharia de Software

Fundamentos de Projeto e Análise de Algoritmos

Trabalho Prático

Laura Lourdes Coutinho Rodrigues
Pedro de Sousa Motta

Belo Horizonte

2024

a) Solução com Backtracking

Na abordagem de backtracking, foram testadas todas as opções possíveis de lances garantindo que a soma total dos megawatts não exceda a quantidade e buscando a combinação que maximize o valor obtido. Para evitar que a execução se prolongue indefinidamente, há um limite de tempo de 30 segundos para a busca.

Parâmetros:
- lances: Uma lista de tuplas, onde cada tupla representa um lance e contém dois elementos: a quantidade de recursos (mw) e o valor do lance (dinheiros).
- energia_disponivel: A quantidade total de recursos disponíveis para alocação.

Retorna:
- O valor total máximo que pode ser obtido com a alocação ótima dos recursos disponíveis.

b) Solução com Algoritmo Guloso

Na abordagem gulosa, foram utilizadas duas estratégias diferentes: escolher os primeiros lances com maior valor e escolher os primeiros lances com o maior valor por megawatt. O objetivo é maximizar o valor total sem ultrapassar a energia disponível.

Guloso pelo maior valor:
Ordena os lances em ordem decrescente de valor e, em seguida, seleciona os lances até que o limite de energia disponível seja alcançado ou excedido.

Parâmetros:
- lances: Uma lista de tuplas, onde cada tupla representa um lance e contém dois elementos: a quantidade de energia (lote) e o valor do lance.
- energia_disponivel: A quantidade de energia disponível para ser alocada.

Retorna:
- O valor total alcançado pela seleção dos lances dentro do limite de energia disponível.

Guloso pelo maior valor por megawatt:
Ordena os lances em ordem decrescente de valor por unidade de energia (valor/megawatt) e, em seguida, seleciona os lances até que o limite de energia disponível seja alcançado ou excedido.

Parâmetros:
- lances (list of tuples): Uma lista de tuplas, onde cada tupla representa um lance e contém dois elementos: a quantidade de energia (lote) e o valor do lance.
- energia_disponivel (int/float): A quantidade de energia disponível para ser alocada.

Retorna:
- int/float: O valor total alcançado pela seleção dos lances dentro do limite de energia disponível.

c) Solução com Divisão e Conquista

A estratégia de divisão e conquista foi dividir uma lista de lances em duas menores, resolver independentemente e encontrar a melhor combinação entre as soluções das metades.

Parâmetros:

- lances: Uma lista de tuplas, onde cada tupla representa um lance e contém dois elementos: a quantidade de energia (em megawatts) e o valor do lance (em unidades monetárias).

- energia_disponivel: A quantidade de energia disponível para ser alocada (em megawatts).

- memo: Um dicionário usado para armazenar os resultados de subproblemas já calculados. Padrão é None, o que significa que nenhum resultado de subproblema é inicialmente armazenado.

d) Solução com Programação Dinâmica

Na programação dinâmica, foi utilizado uma abordagem semelhante ao problema da mochila, onde é utilizado uma tabela para armazenar os valores máximos possíveis para diferentes capacidades de energia, as linhas representam os lances e as colunas a energia disponível.

Resultados

Primeira tentativa:

1) **Considerando 25 empresas para o primeiro e segundo conjunto**

Resultados Laura e Pedro respectivamente

```
Resumo dos Resultados:

Conjunto 1:
  Guloso (Maior Valor) => Resultado: 26725, Tempo: 0.02s
  Guloso (Maior Valor por MW) => Resultado: 26725, Tempo: 0.01s
  Divisão e Conquista => Resultado: 7480, Tempo: 0.00s
  Programação Dinâmica => Resultado: 26725, Tempo: 0.03s
  Backtracking => Resultado: 26725, Tempo: 13.76s


Conjunto 2:
  Guloso (Maior Valor) => Resultado: 38673, Tempo: 0.00s
  Guloso (Maior Valor por MW) => Resultado: 39271, Tempo: 0.01s
  Divisão e Conquista => Resultado: 13899, Tempo: 0.00s
  Programação Dinâmica => Resultado: 40348, Tempo: 0.03s
  Backtracking => Resultado: 40348, Tempo: 13.70s
```

```
Resumo dos Resultados:

Conjunto 1:
  Guloso (Maior Valor) => Resultado: 26725, Tempo: 0.00s
  Guloso (Maior Valor por MW) => Resultado: 26725, Tempo: 0.01s
  Divisão e Conquista => Resultado: 7480, Tempo: 0.00s
  Programação Dinâmica => Resultado: 26725, Tempo: 0.03s
  Backtracking => Resultado: 26725, Tempo: 14.32s


Conjunto 2:
  Guloso (Maior Valor) => Resultado: 38673, Tempo: 0.00s
  Guloso (Maior Valor por MW) => Resultado: 39271, Tempo: 0.01s
  Divisão e Conquista => Resultado: 13899, Tempo: 0.00s
  Programação Dinâmica => Resultado: 40348, Tempo: 0.03s
  Backtracking => Resultado: 40348, Tempo: 14.06s
```

2) **Considerando 25 empresas para o primeiro e 50 empresas para o segundo conjunto**

```
Resumo dos Resultados:

Conjunto 1:
  Guloso (Maior Valor) => Resultado: 26725, Tempo: 0.01s
  Guloso (Maior Valor por MW) => Resultado: 26725, Tempo: 0.01s
  Divisão e Conquista => Resultado: 7480, Tempo: 0.00s
  Programação Dinâmica => Resultado: 26725, Tempo: 0.03s
  Backtracking => Resultado: 26725, Tempo: 14.06s


Conjunto 2:
  Guloso (Maior Valor) => Resultado: 37755, Tempo: 0.00s
  Guloso (Maior Valor por MW) => Resultado: 39271, Tempo: 0.02s
  Divisão e Conquista => Resultado: 12469, Tempo: 0.00s
  Programação Dinâmica => Resultado: 40348, Tempo: 0.06s
  Backtracking => Resultado: 40348, Tempo: 30.00s
```

3) **Considerando 50 empresas para o primeiro e 25 empresas para o segundo conjunto**

```
Resumo dos Resultados:

Conjunto 1:
  Guloso (Maior Valor) => Resultado: 37755, Tempo: 0.01s
  Guloso (Maior Valor por MW) => Resultado: 39271, Tempo: 0.01s
  Divisão e Conquista => Resultado: 12360, Tempo: 0.01s
  Programação Dinâmica => Resultado: 40348, Tempo: 0.06s
  Backtracking => Resultado: 40348, Tempo: 30.01s


Conjunto 2:
  Guloso (Maior Valor) => Resultado: 26725, Tempo: 0.00s
  Guloso (Maior Valor por MW) => Resultado: 26725, Tempo: 0.00s
  Divisão e Conquista => Resultado: 7586, Tempo: 0.00s
  Programação Dinâmica => Resultado: 26725, Tempo: 0.05s
  Backtracking => Resultado: 26725, Tempo: 7.21s
```

```
Conjunto 3:
  Guloso (Maior Valor) => Resultado: 39643, Tempo: 0.01s
  Guloso (Maior Valor por MW) => Resultado: 38790, Tempo: 0.02s
  Divisão e Conquista => Resultado: 11809, Tempo: 0.00s
  Programação Dinâmica => Resultado: 39643, Tempo: 0.08s
  Backtracking => Resultado: 39643, Tempo: 14.55s
```

Testes Aleatórios:

Realizando a execução com conjuntos aleatórios de 20, 30, 40 e 50 empresas, obtivemos os seguintes resultados.

```
Conjunto 3:
  Guloso (Maior Valor) => Resultado: 36280, Tempo: 0.00s
  Guloso (Maior Valor por MW) => Resultado: 36280, Tempo: 0.00s
  Divisão e Conquista => Resultado: 12875, Tempo: 0.00s
  Programação Dinâmica => Resultado: 36280, Tempo: 0.03s
  Backtracking => Resultado: 36280, Tempo: 0.45s


Conjunto 4:
  Guloso (Maior Valor) => Resultado: 37480, Tempo: 0.00s
  Guloso (Maior Valor por MW) => Resultado: 40374, Tempo: 0.00s
  Divisão e Conquista => Resultado: 13510, Tempo: 0.00s
  Programação Dinâmica => Resultado: 40709, Tempo: 0.04s
  Backtracking => Resultado: 40709, Tempo: 30.00s
```

```
 Conjunto 5:
   Guloso (Maior Valor) => Resultado: 38445, Tempo: 0.01s
   Guloso (Maior Valor por MW) => Resultado: 40865, Tempo: 0.01s
   Divisão e Conquista => Resultado: 15475, Tempo: 0.00s
   Programação Dinâmica => Resultado: 41390, Tempo: 0.05s
   Backtracking => Resultado: 41390, Tempo: 30.00s


 Conjunto 6:
   Guloso (Maior Valor) => Resultado: 40672, Tempo: 0.01s
   Guloso (Maior Valor por MW) => Resultado: 41514, Tempo: 0.01s
   Divisão e Conquista => Resultado: 15950, Tempo: 0.01s
   Programação Dinâmica => Resultado: 41822, Tempo: 0.06s
   Backtracking => Resultado: 41822, Tempo: 30.00s
```

Testes Aleatórios:

Os seguintes testes foram feitos gerando um conjunto de testes com números aleatórios com as seguintes especificações:

data > gerar_conjuntos.py > gerar_conjuntos

```python
import random

def gerar_conjuntos(tamanho_max, incremento):
    conjuntos = []
    for t in range(10, tamanho_max + 1, incremento):
        lances = []
        for i in range(t):
            energia = random.randint(1, 1000)  # Energias variando entre 1 e 1000 megawatts
            valor = random.randint(1, 2000)  # Valores variando entre 1 e 2000 dinheiros
            lances.append((energia, valor))
        conjuntos.append(lances)
    return conjuntos


```

```
Médias dos resultados e tempos de divisão e conquista: 10451.5, 3.234027624130249
Médias dos resultados e tempos de programação dinâmica: 29542.7, 0.06747698783874512

Todos os testes foram concluídos com sucesso!


Resumo dos Resultados:
Backtracking - Tamanho máximo resolvido: 76
Algoritmo Guloso 1 - Média de resultados: 9870.0, Média de tempos: 0.08863410949707032 segundos
Algoritmo Guloso 2 - Média de resultados: 29272.5, Média de tempos: 0.08224575519561768 segundos
Divisão e Conquista - Média de resultados: 10451.5, Média de tempos: 3.234027624130249 segundos
Programação Dinâmica - Média de resultados: 29542.7, Média de tempos: 0.06747698783874512 segundos
```

Aumentando os valores:

data > gerar_conjuntos.py > ...

```python
import random

def gerar_conjuntos(tamanho_max, incremento):
    conjuntos = []
    for t in range(10, tamanho_max + 1, incremento):
        lances = []
        for i in range(t):
            energia = random.randint(1, 10000)  # Energias variando entre 1 e 10000 megawatts
            valor = random.randint(1, 3000)  # Valores variando entre 1 e 3000 dinheiros
            lances.append((energia, valor))
        conjuntos.append(lances)
    return conjuntos
```

```
Resumo dos Resultados:
Backtracking - Tamanho máximo resolvido: 100
Algoritmo Guloso 1 - Média de resultados: 7666.9, Média de tempos: 0.1341808557510376 segundos
Algoritmo Guloso 2 - Média de resultados: 11267.0, Média de tempos: 0.12640082836151123 segundos
Divisão e Conquista - Média de resultados: 7015.2, Média de tempos: 1.0107312679290772 segundos
Programação Dinâmica - Média de resultados: 11395.1, Média de tempos: 0.05930426120758057 segundos
```

Aumentando os valores até atingir um ponto máximo

data > gerar_conjuntos.py > gerar_conjuntos

```python
import random

def gerar_conjuntos(tamanho_max, incremento):
    conjuntos = []
    for t in range(10, tamanho_max + 1, incremento):
        lances = []
        for i in range(t):
            energia = random.randint(1, 10000000000000)
            valor = random.randint(1, 300000000000000)
            lances.append((energia, valor))
        conjuntos.append(lances)
    return conjuntos


```

```
Médias dos resultados e tempos de divisão e conquista: 0.0, 0.005826926231384278
Médias dos resultados e tempos de programação dinâmica: 0.0, 0.0660715103149414

Todos os testes foram concluídos com sucesso!


Resumo dos Resultados:
Backtracking - Tamanho máximo resolvido: 100
Algoritmo Guloso 1 - Média de resultados: 0.0, Média de tempos: 0.1186812162399292 segundos
Algoritmo Guloso 2 - Média de resultados: 0.0, Média de tempos: 0.12560908794403075 segundos
Divisão e Conquista - Média de resultados: 0.0, Média de tempos: 0.005826926231384278 segundos
Programação Dinâmica - Média de resultados: 0.0, Média de tempos: 0.0660715103149414 segundos
```