Pontifícia Universidade Católica de Minas Gerais

Fundamentos de Projeto e Análise de Algoritmos

Relatório do Trabalho Prático em grupo

Alunos:

- Caio Elias Rodrigues Araujo
- Victor de Souza Friche Passos
- Vítor de Figueiredo Pereira

2024
Belo Horizonte

Informações

Critério de avaliação escolhido: **Individual**

Linguagem de programação: **Java**

Algoritmos implementados: **Backtracking, Programação Dinâmica e Divisão e Conquista**

Link do repositório: https://github.com/DisciplinasProgramacao/projetodealgoritmos-leilaoenergia-projetodealgoritmos_leilaoenergia

Divisão de tarefas

Integrante	Tarefas executadas
Victor	• Implementação do método de Backtracking • Elaboração da seção Backtracking, Informações e Conclusão Final • Participação na seção de Comparação
Caio	• Implementação do método de Programação Dinâmica • Participação na seção de Comparação
Vítor	• Implementação do método de Divisão e Conquista • Participação na seção de Comparação

Backtracking

Victor de Souza Friche Passos

Primeira implementação (Corresponde a Backtracking no repositório)

Estratégia Geral:

Definem-se variáveis para armazenar a energia total disponível (`energiaTotal`), os lances disponíveis (`lances`), a melhor combinação de lances (`melhorLances`), o melhor valor total obtido (`melhorValor`) e a energia total utilizada na melhor combinação (`energiaUtilizada`).

A função `encontrarMelhorVenda` é a função principal que inicializa o processo de busca e, ao final, imprime os melhores resultados encontrados. Ela chama recursivamente uma funcao `encontrarMelhorVenda` de backtracking com parâmetros que tenta todas as combinações possíveis de lances, respeitando o limite de energia total.

A cada combinação de lances, se a soma dos valores dos lances for maior do que o melhor valor obtido até o momento (`melhorValor`), a combinação atual é armazenada como a melhor solução.

Critério de Poda:

O critério de poda é garantir que a soma da energia dos lances atuais não exceda a energia total disponível. Isto é feito com a verificação “if (energiaUsada + lances[i].megawatts <= energiaTotal)” antes de adicionar um novo lance à combinação atual.

Passo a passo:

1. Inicializa as variáveis.

2. Início da busca: a função *encontrarMelhorVenda* chama a função recursiva *encontrarMelhorVenda* com uma lista vazia para lances atuais, índice inicial 0, energia usada 0 e valor atual 0.

3. Verificação de condição: a função recursiva verifica se a energia utilizada até o momento não excede a energia total disponível. Se sim, continua a busca; se não, volta (backtrack).

4. Se a combinação atual de lances proporciona um valor total maior do que o melhor valor obtido até o momento, atualiza-se a melhor solução.

5. Para cada lance a partir do índice atual, tenta-se adicionar o lance à combinação atual, respeitando o limite de energia, e a busca recursiva continua com o próximo lance.

6. Após tentar um lance, o lance é removido da combinação atual para explorar outras combinações possíveis.

7. Quando todas as combinações foram exploradas, a melhor solução é impressa.

Testes:

Teste 1

*** Conjunto de empresas interessadas 1 (nome, quantidade, valor):**

E1;430;1043
E2;428;1188
E3;410;1565
E4;385;1333
E5;399;1214
E6;382;1498
E7;416;1540
E8;436;1172
E9;416;1386
E10;423;1097
E11;400;1463
E12;406;1353
E13;403;1568
E14;390;1228
E15;387;1542
E16;390;1206
E17;430;1175
E18;397;1492
E19;392;1293
E20;393;1533
E21;439;1149
E22;403;1277
E23;415;1624
E24;387;1280
E25;417;1330

Considerando 8000 megawatts.

Resultado:

```
Melhor solução encontrada:
Lances:
        (megawatts=410, valor=1565, empresa='E3')
        (megawatts=385, valor=1333, empresa='E4')
        (megawatts=399, valor=1214, empresa='E5')
        (megawatts=382, valor=1498, empresa='E6')
        (megawatts=416, valor=1540, empresa='E7')
        (megawatts=416, valor=1386, empresa='E9')
        (megawatts=400, valor=1463, empresa='E11')
        (megawatts=406, valor=1353, empresa='E12')
        (megawatts=403, valor=1568, empresa='E13')
        (megawatts=390, valor=1228, empresa='E14')
        (megawatts=387, valor=1542, empresa='E15')
        (megawatts=390, valor=1206, empresa='E16')
        (megawatts=397, valor=1492, empresa='E18')
        (megawatts=392, valor=1293, empresa='E19')
        (megawatts=393, valor=1533, empresa='E20')
        (megawatts=403, valor=1277, empresa='E22')
        (megawatts=415, valor=1624, empresa='E23')
        (megawatts=387, valor=1280, empresa='E24')
        (megawatts=417, valor=1330, empresa='E25')

Com valor total de 26725 dinheiros e energia total utilizada de 7588 megawatts
Conjunto: Conjunto 1, Tempo médio de execução: 437 ms
```

A distribuição de energia entre as empresas participantes no leilão resultou em uma

utilização total de 7588 megawatts dos 8000 megawatts disponíveis. Esse resultado foi obtido após a execução do algoritmo de backtracking que buscou maximizar o valor total dos lances aceitos, que durou 437ms, ou seja, nem 1 segundo! O valor total obtido com a melhor combinação de lances (19 lances) foi de 26725 dinheiros. Esse é o valor máximo encontrado pelo algoritmo considerando todas as combinações possíveis dentro da restrição de energia.

Teste 2

*** Conjunto de empresas interessadas 2 (nome, quantidade, valor):**

E1;313;1496
E2;398;1768
E3;240;1210
E4;433;2327
E5;301;1263
E6;297;1499
E7;232;1209
E8;614;2342
E9;558;2983
E10;495;2259
E11;310;1381
E12;213;961
E13;213;1115
E14;346;1552
E15;385;2023
E16;240;1234
E17;483;2828
E18;487;2617
E19;709;2328
E20;358;1847
E21;467;2038
E22;363;2007
E23;279;1311
E24;589;3164
E25;476;2480

Considerando 8000 megawatts.

Resultado:

A distribuição de energia entre as empresas participantes no leilão resultou em uma utilização total de 7962 megawatts dos 8000 megawatts disponíveis. Esse resultado foi obtido após a execução do algoritmo de backtracking que buscou maximizar o valor total dos lances aceitos, que durou 420ms, ou seja, nem 1 segundo! O valor total obtido com a melhor combinação de lances (20 lances) foi de 40348 dinheiros. Esse é o valor máximo encontrado pelo algoritmo considerando todas as combinações possíveis dentro da restrição de energia.

Teste 3

Gerador de Problemas: o algoritmo de geração de problemas é utilizado para gerar conjuntos de lances (ofertas de empresas) de forma randômica dentro de um intervalo.

Informações importantes:

`TAM_BASE` é um valor de referência escolhido como ponto de partida para a geração de valores. Ele é uma constante que facilita a criação de valores de teste escaláveis.

int megawatts = 200 + aleatorio.nextInt(401); define os valores dos megawatts entre 200 e 600.

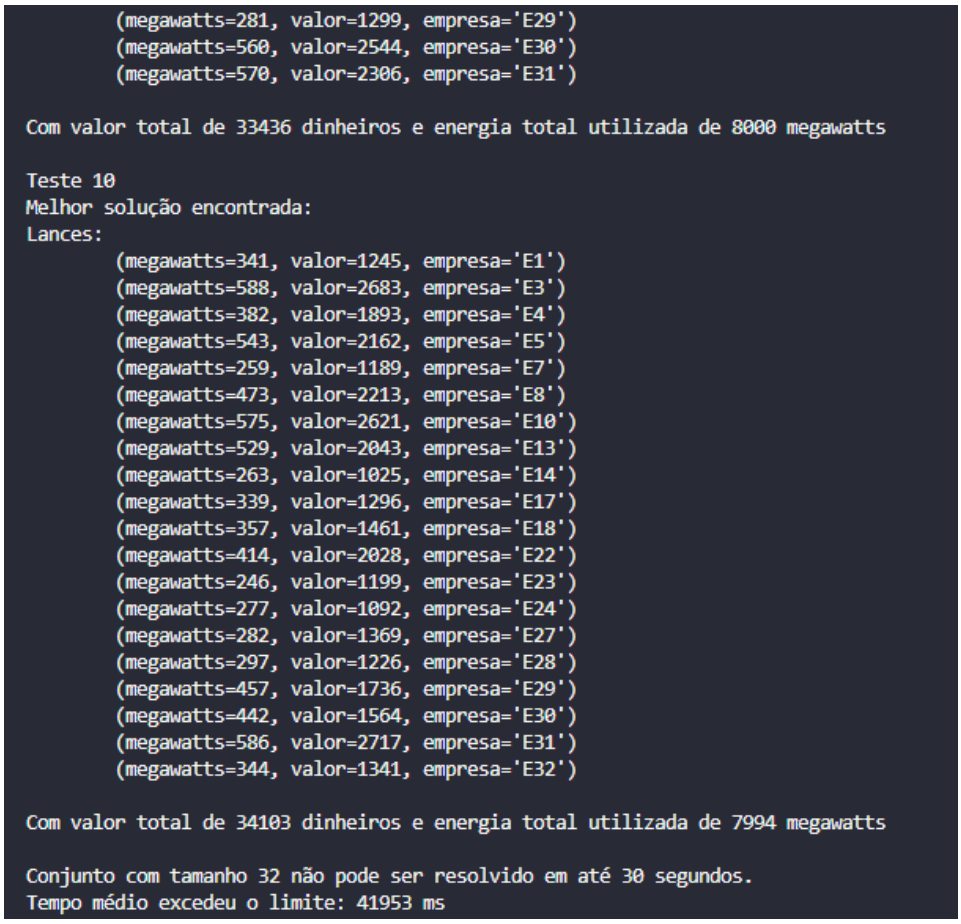
valorMinimo é 3 vezes o valor do megawatts e ***valorMaximo*** é 5 vezes maior.

int valor = valorMinimo + aleatorio.nextInt(valorMaximo - valorMinimo + 1);
Gera o valor do lance dentro do intervalo [valorMinimo, valorMaximo], garantindo que o valor seja pelo menos `valorMinimo` e no máximo `valorMaximo`.

Utilizando TAM_BASE = 13 e essas configurações de randomização das variáveis são gerados e utilizados valores parecidos com os conjuntos de dados anteriores já testados. Assim, as chances de ocorrer dados irreais como 3 megawatts por 2100 dinheiros são pequenas.

IMPORTANTE: a cada teste de determinado conjunto ele gerará dados diferentes em relação ao teste anterior. Assim ele sempre irá encontrar soluções diferentes, mesmo tendo a mesma quantidade de lances.

Resultado: no terminal o resultado ficou gigante, então vou deixar uma print do final da execução e um arquivo mostrando pelo o menos as 10 execuções de teste para o conjuntos 32 e 31.



O algoritmo rodou/testou 10 conjuntos de teste achando soluções para cada tamanho T parando em T = 32 lances. 31 foi o último número de lances que não excedeu o tempo limite de 30 segundos, chegando a executar na média 21143 ms ou aproximadamente 21 segundos.

10 testes de conjuntos de 32 lances ultrapassaram 30 segundos e o algoritmo parou sua execução. Confira no arquivo abaixo:

https://drive.google.com/file/d/1GpqgF94-MGX3IMJhM5nnQEWg0C78ZTZF/view?usp=sharing

Análise Crítica de resultados

O critério de poda para esses conjuntos de dados foi efetivo, já que o tempo de execução foi relativamente rápido para os conjuntos de dados testados e ele sempre conseguia resolver o problema do leilão de energia, que era vender sua energia produzida, obtendo o maior valor possível no conjunto de suas vendas.

Pontos Positivos:

1. Nos testes realizados, o algoritmo de Backtracking conseguiu utilizar uma quantidade significativa de megawatts disponíveis, próximo ao máximo permitido, resultando em uma utilização eficiente dos recursos.

2. O algoritmo foi bem-sucedido em encontrar a combinação de lances que maximiza o valor total obtido. No primeiro teste, obteve 26725 dinheiros e, no segundo, 40348 dinheiros, ambos valores máximos possíveis dentro das restrições de energia.

3. O tempo de execução do algoritmo foi relativamente rápido para os conjuntos de dados testados. Ambos os testes iniciais foram concluídos em menos de um segundo, mostrando que o algoritmo é eficiente para conjuntos de lances de tamanho pequeno e moderado.

Pontos Negativos:

1. Como evidenciado nos testes de conjuntos maiores, o tempo de execução aumenta exponencialmente (complexidade exponencial). No teste 3, a execução do algoritmo foi interrompida para conjuntos com 32 ou mais lances devido ao tempo de execução exceder 30 segundos. Isso indica que, para problemas de maior escala, o algoritmo pode não ser viável em termos de tempo de execução.

2. A eficiência do algoritmo pode ser fortemente influenciada pela estrutura dos dados de entrada. Eu percebi no teste 3 que quando os valores dos megawatts dos lances são pequenos, o número de combinações possíveis aumenta. Isso ocorre porque é possível selecionar mais lances dentro do limite total de energia, resultando em um aumento do espaço de busca que o algoritmo deve explorar.

3. Embora o critério de poda ajude a reduzir o espaço de busca, ele ainda pode ser insuficiente para grandes conjuntos de dados. Melhorar o critério de poda pode ajudar o algoritmo a executar conjuntos maiores que 32 mas de qualquer forma, em algum momento ele será ineficiente para algum conjunto de tamanho T de dados.

Conclusão

O algoritmo de Backtracking implementado se mostrou eficaz para resolver o problema do leilão de energia para conjuntos de dados pequenos e moderados. Ele foi capaz de maximizar o valor total obtido, utilizando uma quantidade significativa de megawatts disponíveis dentro do limite de energia estipulado. No entanto, a complexidade exponencial do algoritmo limita sua aplicação para conjuntos de dados maiores, onde o tempo de execução se torna impraticável. A eficiência do critério de poda é crucial para o desempenho do algoritmo, mas melhorias são necessárias para lidar com conjuntos maiores. A estrutura dos dados de entrada também desempenha um papel significativo na eficiência do algoritmo.

Divisão e Conquista

O algoritmo foi baseado no problema da mochila onde é definido se cada item deve ou não ser incluído no conjunto de itens.

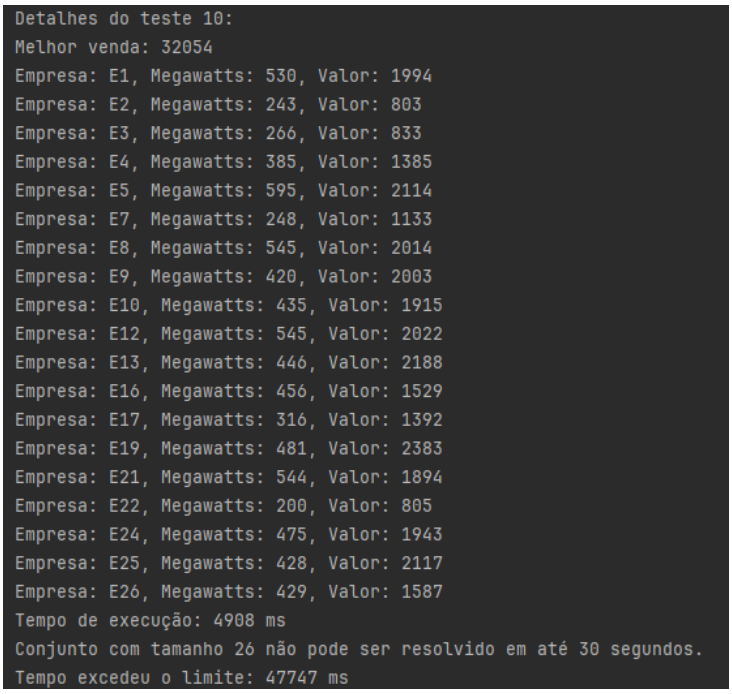
Conjunto 1:

```
Melhor venda para o conjunto 1: 26725
Empresa: E3, Megawatts: 410, Valor: 1565
Empresa: E4, Megawatts: 385, Valor: 1333
Empresa: E5, Megawatts: 399, Valor: 1214
Empresa: E6, Megawatts: 382, Valor: 1498
Empresa: E7, Megawatts: 416, Valor: 1540
Empresa: E9, Megawatts: 416, Valor: 1386
Empresa: E11, Megawatts: 400, Valor: 1463
Empresa: E12, Megawatts: 406, Valor: 1353
Empresa: E13, Megawatts: 403, Valor: 1568
Empresa: E14, Megawatts: 390, Valor: 1228
Empresa: E15, Megawatts: 387, Valor: 1542
Empresa: E16, Megawatts: 390, Valor: 1206
Empresa: E18, Megawatts: 397, Valor: 1492
Empresa: E19, Megawatts: 392, Valor: 1293
Empresa: E20, Megawatts: 393, Valor: 1533
Empresa: E22, Megawatts: 403, Valor: 1277
Empresa: E23, Megawatts: 415, Valor: 1624
Empresa: E24, Megawatts: 387, Valor: 1280
Empresa: E25, Megawatts: 417, Valor: 1330
Tempo de execução: 2448 ms
```

Conjunto 2:

```
Melhor venda para o conjunto 2: 40348
Empresa: E1, Megawatts: 313, Valor: 1496
Empresa: E2, Megawatts: 398, Valor: 1768
Empresa: E3, Megawatts: 240, Valor: 1210
Empresa: E4, Megawatts: 433, Valor: 2327
Empresa: E6, Megawatts: 297, Valor: 1499
Empresa: E7, Megawatts: 232, Valor: 1209
Empresa: E9, Megawatts: 558, Valor: 2983
Empresa: E10, Megawatts: 495, Valor: 2259
Empresa: E11, Megawatts: 310, Valor: 1381
Empresa: E13, Megawatts: 213, Valor: 1115
Empresa: E14, Megawatts: 346, Valor: 1552
Empresa: E15, Megawatts: 385, Valor: 2023
Empresa: E16, Megawatts: 240, Valor: 1234
Empresa: E17, Megawatts: 483, Valor: 2828
Empresa: E18, Megawatts: 487, Valor: 2617
Empresa: E20, Megawatts: 358, Valor: 1847
Empresa: E21, Megawatts: 467, Valor: 2038
Empresa: E22, Megawatts: 363, Valor: 2007
Empresa: E23, Megawatts: 279, Valor: 1311
Empresa: E24, Megawatts: 589, Valor: 3164
Empresa: E25, Megawatts: 476, Valor: 2480
Tempo de execução: 2364 ms
```

Conjunto gerador:



O processo do algoritmo de divisão e conquista se encerrou no conjunto 25, pois o conjunto de tamanho 26 já excedia o tempo limite. Em análise, foi identificado que o tempo de execução do próximo conjunto é próximo ao dobro do tempo do conjunto anterior a ele, como exemplo, se tem o conjunto de tamanho 25 e 26 na imagem acima.

Observação: Antes da conclusão de código, o processo que foi desenvolvido, não mantinha em vista quais empresas participaram da venda, o que fez com que o processo apenas finalizasse próximo do conjunto 35. É possível observar que a diferença dos dados que serão tratados pode causar uma mudança de execução abrupta, devido à complexidade das estruturas.

Pontos positivos:

1. Garantia de encontrar a solução ótima: Como a busca exaustiva tenta todas as combinações possíveis, ela sempre encontrará a solução ótima se ela existir.

2. Simplicidade: A busca exaustiva é um algoritmo simples de entender e implementar. Não requer nenhum conhecimento prévio sobre o problema além do critério de solução.

Pontos negativos:

1. Ineficiência: A busca exaustiva pode ser muito ineficiente para problemas grandes, pois o número de combinações possíveis pode crescer exponencialmente com o tamanho do problema. Isso pode levar a tempos de execução muito longos.

2. Uso de memória: A busca exaustiva pode exigir uma grande quantidade de memória para armazenar todas as combinações possíveis, especialmente para problemas grandes.

3. Não é prático para problemas em tempo real: Devido ao seu tempo de execução

potencialmente longo, a busca exaustiva pode não ser prática para problemas que requerem uma solução em tempo real.

Programação Dinâmica

O algoritmo de programação dinâmica utilizado é o de "knapsack problem" (problema da mochila), que é aplicado da seguinte forma: quando recebemos as ofertas de energia (lances), somamos a quantidade total de megawatts (MW) ofertados e definimos a capacidade máxima que pode ser adquirida. Para cada capacidade parcial, encontramos o valor máximo arrecadado, usando nosso algoritmo (com sua tabela equivalente), que sempre tenta satisfazer a capacidade total de maneira ótima ou chega o mais próximo possível. No final, temos sempre o melhor resultado possível em termos de valor arrecadado, respeitando o limite de capacidade de MW, e pela tabela, podemos localizar quais ofertas devem ser aceitas.

Estrutura da Tabela:

Linhas: Ofertas disponíveis;

Colunas: Capacidade máxima de MW.

Resultados Obtidos:

Partes Interessadas	Média Tempo (ms)
32 (T)	34
128 (4T)	78
256 (8T)	106
320 (10T)	148

A análise dos dados revela uma tendência de aumento na média de tempo à medida que o número de partes interessadas cresce. Com 32 partes interessadas, a média de tempo é de 34 ms. Quando o número de partes interessadas é quadruplicado para 128, a média de tempo aumenta para 78 ms, representando um crescimento significativo, mas ainda mantendo a eficiência em termos de tempo de processamento.
À medida que o número de partes interessadas continua a aumentar, esse padrão se mantém. Com 256 partes interessadas, a média de tempo é de 106 ms, e com 320 partes interessadas, a média de tempo chega a 148 ms. Esses resultados indicam que o sistema mantém uma escalabilidade aceitável, com tempos de processamento aumentando de forma previsível e não exponencial.
Essa linearidade no aumento do tempo de processamento sugere que o algoritmo utilizado para lidar com as partes interessadas é eficiente e capaz de lidar com grandes volumes de dados sem comprometer significativamente o desempenho. Além disso, o fato de a média de tempo não aumentar de forma drástica entre as diferentes quantidades de partes interessadas demonstra a robustez do sistema frente ao crescimento de dados.
Em resumo, os dados indicam que o sistema é bem projetado para escalar com um número crescente de partes interessadas, mantendo tempos de resposta dentro de limites razoáveis e demonstrando uma complexidade computacional manejável.

Comparação de resultados dos algoritmos

Critério	Backtracking	Divisão e Conquista	Programação Dinâmica
Garantia de Solução Ótima	Sim	Sim	Sim
Escalabilidade	Limitada para grandes conjuntos de dados	Limitada para grandes conjuntos de dados	Alta, mantém tempos de execução razoáveis
Facilidade de Implementação	Moderada	Alta	Moderada
Exemplo de Resultado 1	26725 dinheiros (7588 MW, 437 ms)	26725 dinheiros (2448 ms)	Não disponível
Exemplo de Resultado 2	40348 dinheiros (7962 MW, 420 ms)	40348 dinheiros (2364 ms)	Não disponível
Exemplo de Resultado 3	Interrompido para >= 32 lances (excedeu 30s)	Interrompido para > 25 lances (excedeu tempo limite)	34 ms (32 lances), 78 ms (128 lances), 148 ms (320 lances)

Backtracking é eficiente para conjuntos de dados pequenos e moderados, mas sua complexidade exponencial limita a aplicação em problemas maiores.

Divisão e Conquista oferece uma abordagem simples e garantida para encontrar soluções ótimas, mas é impraticável para problemas maiores devido ao tempo de execução e uso de memória.

Programação Dinâmica se destaca pela eficiência e escalabilidade, mantendo tempos de resposta manejáveis mesmo com grandes volumes de dados. No entanto, pode ser mais complexa de implementar e menos eficiente em termos de uso de memória.

Para aplicações práticas, a escolha do algoritmo depende do tamanho do conjunto de dados e do tipo de problema a ser solucionado.

Conclusão Final

Para o problema do leilão de energia, a programação dinâmica foi a abordagem mais eficaz devido à sua capacidade de lidar com grandes conjuntos de dados de maneira eficiente e escalável. O backtracking também foi eficaz para conjuntos menores, mas tornou-se impraticável para grandes conjuntos devido à sua complexidade exponencial. A divisão e conquista foi a alternativa menos eficaz apesar de também conseguir solucionar o problema. Isso porque seu tempo de execução é maior e aumenta exponencialmente de acordo com o conjunto de dados e utiliza muita memória em sua execução.