daa_knapsack

February 22, 2023

1 Análise do algoritmo da mochila binária - Abordagens gulosa e dinâmica

1.1 O problema da mochila binária

O problema da mochila binária consiste em escolher um subconjunto de itens de peso e valor conhecidos que caibam em uma mochila de capacidade limitada.

A abordagem gulosa para resolver o problema da mochila binária consiste em selecionar, a cada iteração, o item com a maior relação valor/peso e incluí-lo na mochila, desde que a capacidade permita. Essa abordagem tem a vantagem de ser rápida, mas pode não produzir a solução ótima em todos os casos.

Já a abordagem de programação dinâmica para resolver o problema da mochila binária é baseada em uma tabela que armazena os valores máximos possíveis para diferentes capacidades de mochila e diferentes subconjuntos de itens. A tabela é preenchida de forma iterativa, considerando-se cada item e decidindo se ele deve ser incluído ou não na mochila. Essa abordagem garante a solução ótima, mas tem complexidade de tempo e espaço maior do que a abordagem gulosa.

Em resumo, a abordagem gulosa é mais rápida, mas pode não produzir a solução ótima em todos os casos, enquanto a abordagem de programação dinâmica garante a solução ótima, mas tem complexidade de tempo e espaço maior. A escolha da abordagem a ser utilizada depende das características do problema a ser resolvido.

1.2 Bibliotecas desenvolvidas

Para a presente analise, foi desenvolvido pelo grupo três bibliotecas escritas em **C+** com **bindings** para a linguagem python:

- **picods**: Biblioteca para visualização de dados em gráficos e tabelas.
- **pydaa**: Biblioteca para Projeto e analise de algoritmos, que contém as implementações do problema da mochila binária.
- **pyaon**: Biblioteca auxiliar para a leitura de arquivos de entrada do problema.

Sendo assim, para instalar essas libs:

[ ]:
```
!pip install picods pydaa pyaon
```

E importar as funções utilizadas:

[2]:
```
import os

from picods import picoplot
from picods import picotable
from pyaon.knapsack import load as read_knapsack
from pydaa.dynamic_programming import knapsack as dynamic_knapsack
from pydaa.greedy import knapsack as greedy_knapsack
```

1.3 Arquivos de entrada

O diretório `./data` contém todos os arquivos utilizados na analise. Cada arquivo tem o nome no formato *mochinaN.txt*, onde *N* denota a escala do problema, ou seja, a quantidade de itens. Por exemplo, o arquivo `mochila10.txt` possui como conteúdo:

```
105
3   42  5   48  42  13  3   20  12  37
2   35  13  29  9   25  2   14  4   17
```

Traduzindo esse arquivo como entrada de uma instância do problema, temos uma mochila com capacidade `105`, com benefícios `[3, 42, 5, 48, 42, 13, 3, 20, 12, 37]` e pesos respectivos `[2, 35, 13, 29, 9, 25, 2, 14, 4, 17]`. Os outros arquivos se comportam de maneira semelhante.

1.4 Tratamento dos dados

Dessa forma, aplicando uma série de funções compostas, podemos ler as entradas para cada instância do problema contidas nos arquivos, obtendo assim os dados: **tempo de execução**, **Resultado obtido** e **sequência de itens a serem inseridos na mochila**:

[3]:
```
data = list(
    map(
        lambda x: [
            x["size"],
            x["dynamic"][0].microseconds,
            x["greedy"][0].microseconds,
            x["dynamic"][1],
            x["greedy"][1],
        ],
        map(
            lambda x: {
                "size":
                x["size"],
                "dynamic":
                dynamic_knapsack(x["data"][0], x["data"][1], x["data"][2]),
                "greedy":
                greedy_knapsack(x["data"][0], x["data"][1], x["data"][2]),
            },
            sorted(
                map(
                    lambda x: {
```

```
                    "size": int("".join(filter(str.isdigit, x))),
                    "data": read_knapsack(f"./data/{x}"),
                },
                os.listdir("data"),
            ),
            key=lambda x: x["size"],
        ),
    ),
))
```

1.5 Analise dos Dados e Discussão

Dessa forma, tendo os dados da resolução de cada instância do problema, podemos prosseguir com a analise desses.

A tabela a seguir mostra, para cada instância do problema, seu tempo de execução e resultado obtido, dentro de cada uma das duas abordagens:

[4]:
```
picotable(
    "Knapsack Results",
    data,
    [
        "Size",
        "Dynamic Time( s)",
        "Greedy Time( s)",
        "Dynamic Result",
        "Greedy Result",
    ],
    ["" for i in range(len(data))],
)
```

Knapsack Results

Size	Dynamic Time(µs)	Greedy Time(µs)	Dynamic Result	Greedy Result
10	4	0	189	178
50	186	2	1196	1196
100	442	3	2028	2028
200	4012	12	5232	5229
300	9571	14	7203	7203
500	22466	25	11889	11889
750	66337	38	18050	18050
1000	21994	49	12072	12072
1250	135858	78	27479	27478
1500	198785	76	33639	33639
2000	308681	114	43194	43193
2500	56408	140	20359	20359
3000	34122	193	15490	15490
4000	434890	246	53521	53521
5000	468233	312	58010	58010

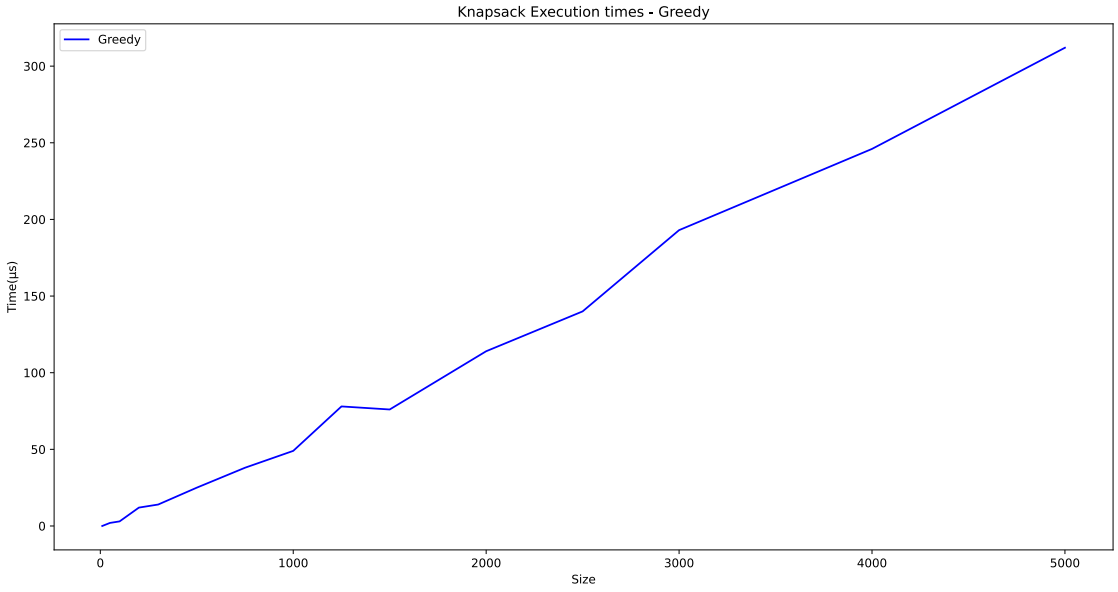
Em uma primeira análise, pode se ver que o método guloso obteve algumas vezes resultados diferentes do método de programação dinâmica.

Além do mais, pode-se ver que o tempo de execução do método de programação dinâmica foi muito maior do que a sua contraparte gulosa.

Seguindo, o gráfico de **tempo** x **tamanho** para cada abordagem do problema pode ser visto a seguir:

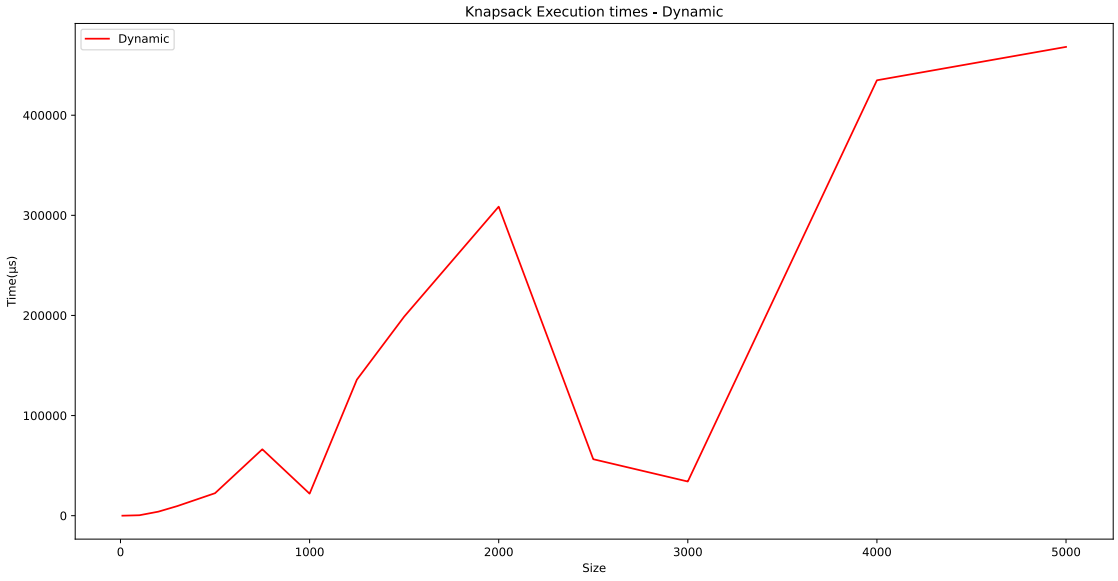
1.5.1 Abordagem Gulosa

[5]:
```
picoplot(
    "Knapsack Execution times - Greedy",
    [
        [i[2] for i in data],
    ],
    [[i[0] for i in data]],
    ["Greedy"],
    ["blue"],
    "Size",
    "Time( s)",
)
```



1.5.2 Abordagem dinamica

[6]:
```
picoplot(
     "Knapsack Execution times - Dynamic",
     [
         [i[1] for i in data],
     ],
     [[i[0] for i in data]],
     ["Dynamic"],
     ["red"],
     "Size",
     "Time( s)",
)
```

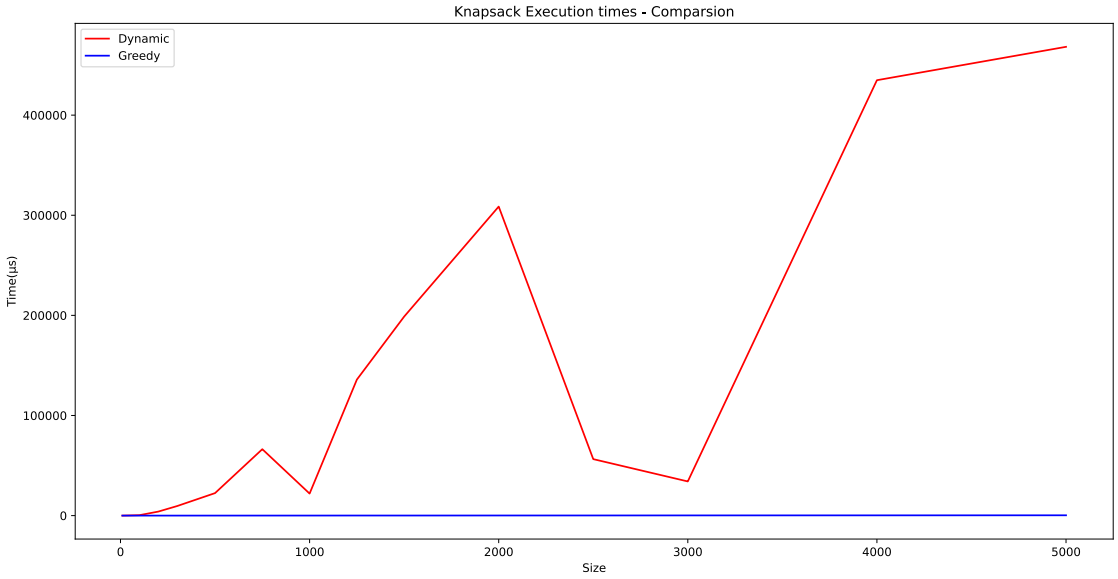


Observando os dois gráficos, e possível notar que, apesar das duas abordagens terem uma sequência de crescimento parecida, a abordagem **dinâmica teve tempos muito discrepantes em relação a abordagem gulosa**. Essa observação pode ser confirmada ao comparar os dois gráficos:

1.5.3 Comparação

[7]:
```
picoplot(
    "Knapsack Execution times - Comparsion",
    [
        [i[1] for i in data],
        [i[2] for i in data],
    ],
    [[i[0] for i in data], [i[0] for i in data]],
    ["Dynamic", "Greedy"],
```

```
    ["red", "blue"],
    "Size",
    "Time( s)",
)
```



1.6 Conclusões

Observando a tabela de resultados, afirma-se o caráter heurístico da estratégia gulosa, ou seja, o fato dela não garantir a obtenção de um resultado ótimo, apenas uma aproximação.

Apesar disso, ao observar a tabela de dados e a tendência de crescimento dos tempos nos gráficos, fica muito claro que, para uma instância de tamanho n e capacidade w, a estratégia gulosa foi muito mais répida, provando a sua complexidade algorítmica de **O(nlogn)** contra os resuldados muito maiores da estratégia dinamica, que por sua vez possui complexidade **O(nw)**.