Centro Federal de Tecnologia de Minas Gerais

Algoritmos e estrutura de dados

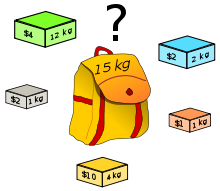
Problema da mochila

Rubio Torres Castro Viana

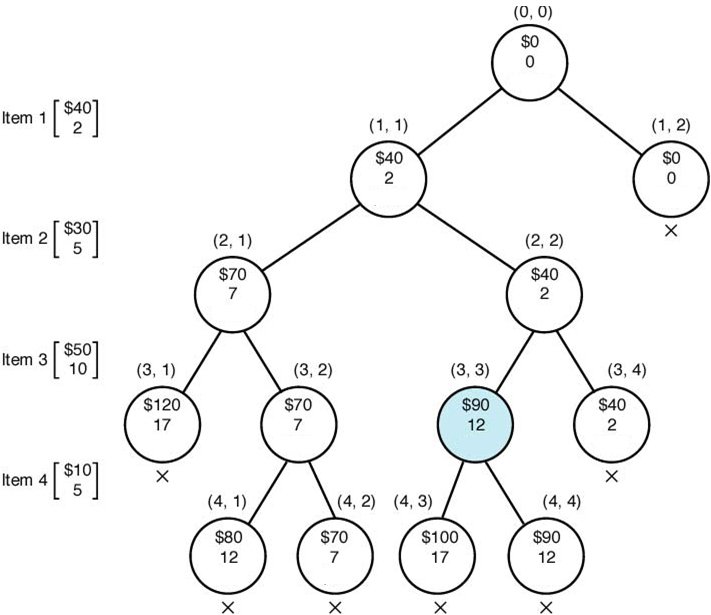
Belo Horizonte, MG

Cccc vc 2017

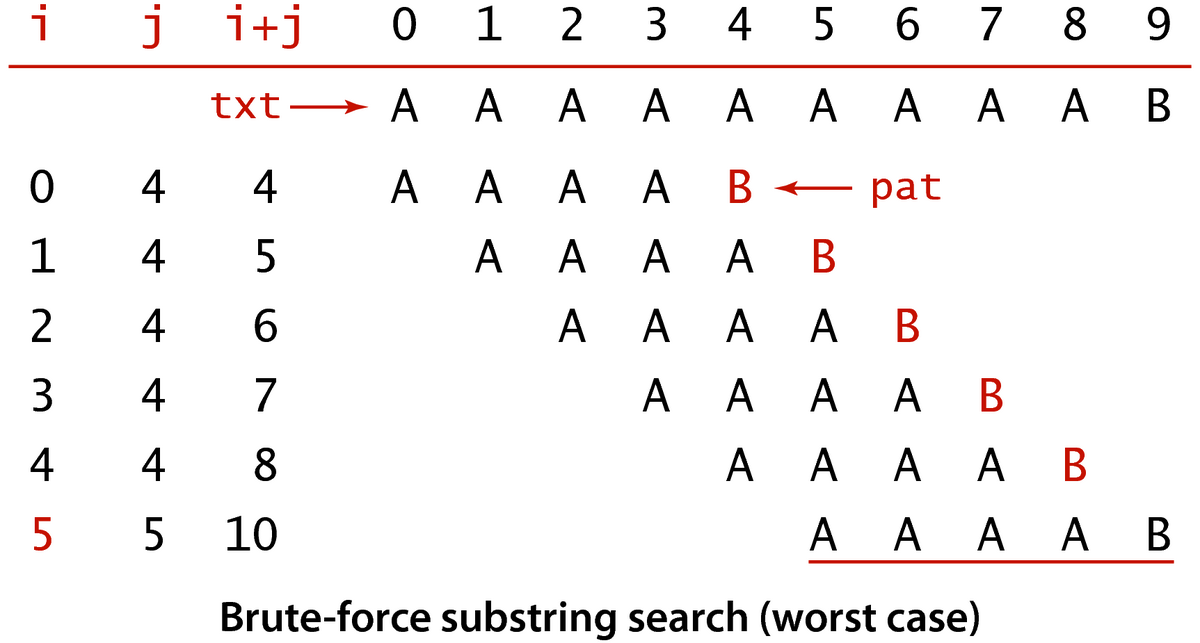
**Resumo**

Neste trabalho apresentarei o problema que é enunciado como um suposto momento no qual você possui uma mochila com uma capacidade W e esteja em um lugar que possui n produtos que você deve pegar, *~~roubar talvez?!~~, sendo que deve ter o maior valor possível dentro do peso W.* Cada produto i tem tamanho wt[i] e valor val[i]. Este trabalho visa produzir algoritmos que tenha o maior valor possível (tentativa e erro) ou aproximações (guloso). 

Esse problema foi exposto em 1972 por Richard Karp e entrou para seus 21 problemas NP-completos.

O modelo é um problema combinatório onde se resolve, geralmente, por enumeração exaustiva tentando todas as soluções e comparando seus valores e pesos. Entretanto, como mostrado na imagem ao lado, sua solução pode ser custosa quando usado a muitos elementos já que criaria muitos subprodutos para enfim gerar um produto. Assim podemos usar também para resolver esse problema, um armazenamento de subprodutos ótimos (programação dinâmica) ou uma aproximação ao problema.

**Força bruta**



Um algoritmo por força bruta, ou enumeração exaustiva, como dito acima é uma estratégia que busca resultados a um problema que pode ser modulado em uma árvore possibilidades. De fato essa busca tende a ser exponencial, por isso sua eficiência vai depender da quantidade de elementos ou de sua limitações, usando por exemplo programação dinâmica.

Para essa aplicação foi usado um arquivo de formato de entrada onde, os n itens da loja serão fornecidos na primeira linha W (capacidade da mochila), na segunda linha n (número de itens) e nas linhas seguintes os itens, cada um em uma linha (contendo o peso seguido do valor).

O algoritmo foi montado pensando em otimizar ao máximo o algoritmo de força bruta, o algoritmo percorre todas as possibilidades de solução para o problema da mochila e as compara para obter uma solução ótima. Desse modo, temos um algoritmo que garantidamente nos dá uma solução ótima, comparando o peso quem vem no vetor “wt” e os valores no vetor “val”.

Algoritmo em português estruturado ou similar:

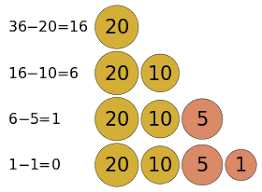
**

Esse algoritmo é de complexidade exponencial Θ( ), ele está implementado no arquivo “*forca\_bruta.c” que pode ser compilado em linha de terminal pelo comando “./forca”*

*Seu arquivo de saída contém um arquivo texto contendo, para cada item escolhido para a solução, o número do item, seu peso e seu valor (uma linha por item), o somatório dos pesos dos itens escolhidos e o somatório dos valores dos itens; e a impressão na tela do tempo total de execução do programa, que carrega sempre a solução ótima embora muitas vezes não seja tão eficiente.*

**

**Algoritmo guloso**

Muitas vezes chamado de míope, o algoritmo guloso é uma técnica de aproximação que escolhe a solução ótima local que pode vir a ser o produto ótimo global. 

O algoritmo possui o “não arrependimento”, o algoritmo não volta atrás em suas decisões, uma vez escolhido o produto, o mesmo não muda.

De fato, embora menos precisa, esse algoritmo tem a ordem de complexidade menor para algumas instâncias e mais possíveis de receber o resultado de alguns problemas que para outros algoritmos têm tempos e complexidades inimagináveis.

Para essa aplicação foi usado um arquivo de formato de entrada onde, os n itens da loja serão fornecidos na primeira linha W (capacidade da mochila), na segunda linha n (número de itens) e nas linhas seguintes os itens, cada um em uma linha (contendo o peso seguido do valor).

O algoritmo foi montado pensando em otimizar ao máximo o algoritmo guloso utilizando a ordenação do vetor de valores “val” no qual deixaremos os maiores valores nas posições iniciais do vetor, após isso colocamos os itens na mochila até estourar o limite da mochila, após isso retornaremos a mochila binária “mochila\_r”, se o elemento estiver na mochila vai ser representado por número 1, se não estiver, 0.

Algoritmo em português estruturado ou similar:

3-0

2-1

4-0

1-1

< 0,1 0,1 0,1 ,>



Esse algoritmo é de complexidade polinomial Θ(), ele está implementado no arquivo “*guloso.c” que pode ser compilado em linha de terminal pelo comando “./guloso”*

*Seu arquivo de saída contém um arquivo texto contendo, para cada item escolhido para a solução, o número do item, seu peso e seu valor (uma linha por item), o somatório dos pesos dos itens escolhidos e o somatório dos valores dos itens; e a impressão na tela do tempo total de execução do programa, que carrega nem sempre a solução ótima mas que muitas vezes é muitas vezes é mais eficiente.*

**Considerações**

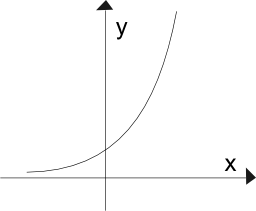
Os algoritmos foram implementados em C e suas entradas foram geradas pelo algoritmo implementado também em c no arquivo gerador\_entradas.c “./gerador” com testes com n variando de 10, 20, 30, 100, 300 e 500 e a razão Capacidade Peso total não ultrapassando 50%.O tempo e a diferença apontada no valor total entre os dois algoritmos foram anotados.



A partir de n=50 força bruta começou a dar problema com o tempo, o crescimento do tempo de execução do algoritmo do força bruta é notável ao contrário do o Guloso, que se mantêm quase que constante durante os testes e quando n chegasse a 300 ou seja 10\*30 estaria na faixa do , todavia o guloso não retornou a solução ótima em quase nenhuma vez.

**Conclusão**

É possível notar pelo gráfico de O(), que sua curva em relação ao tempo cresce muito mais que O() chegando a valores incalculáveis.



Fica claro assim que a eficiência que o algoritmo guloso compensa sua solução mediana diante do tempo exponencial da força bruta sendo assim necessário uma análise sobre seu problema e as possíveis soluções possíveis a ser aplicadas sobre ele.

*Instruções de compilação*

*Gerador:*

*gcc gerador\_entradas.c -o gerador*

*./gerador*

*Força Bruta:*

*gcc forca\_bruta.c -o forca*

*./forca*

*Guloso:*

*gcc guloso.c -o guloso*

*./guloso*

***Referências***

***[1]http://cse.unl.edu/~goddard/Courses/CSCE310J/Lectures/Lecture8-DynamicProgramming.pdf***

***[2]http://www.geeksforgeeks.org/knapsack-problem/***