Instituto Superior Politécnico da Huila
Departamento de Engenharia Informática
Trabalho de Algoritmos e Estrutura de Dados em Engenharia Informática
Tema: Algoritmos gulosos

Artur Cristóvão Muhongo
Nº do SIGU:2022106308.
Agnalda Juliana Binga
Nº do SIGU: 2022156829.
Domingos Jorge Chico Pereira
Nº do SIGU: 2022135437.
Hélio Téofilo Sacopa Soma
Nº do SIGU: 2022124928.
Júlia Lúcia Calumbo Quessongo
Nº do SIGU: 2022145416.
Teodoro Ezequias de Oliveira
Nº do SIGU:2022169258.

Lubango, 2024.

Índice

Introdução

De forma geral, os algoritmos relacionados com otimização lidam com uma sequência de passos, sendo que em cada passo ha um conjunto de escolhas/opções. Para resolver problemas de otimizações, os algoritmos gulosos escolhem as opções que parecem ser a melhor no momento (escolha otima), e esperam que desta forma consiga-se chegar a uma solução otima. Embora nem sempre seja possivel chegar a uma solução otima a partir da utilização de algoritmos gulosos, eles sao eficientes em uma ampla variedade de problemas, conforme poderemos ver neste trabalho.

Objectivos

Este trabalho tem como objetivos: -Compreender o funcionamento e as características dos algoritmos gulosos. -Apresentar exemplos práticos da aplicação desses algoritmos. -Analisar a eficácia e as limitações dos algoritmos gulosos.

Desenvolvimento

Os algoritmos gulosos a sua origem remonta no século XX, com o avanço da matemática e da ciência da computação, os algoritmos gulosos ganharam forma mais definida. Diversos pesquisadores, como **Robert Prim** e **Edmond Chowla**, apresentaram trabalhos pioneiros que lançaram as bases para o estudo formal dessa área pelo mundo tudo.

Exemplos Históricos:

- **Problema das Moedas:** Um dos primeiros problemas a serem solucionados com algoritmos gulosos foi o problema das moedas, onde o objetivo é encontrar o mínimo número de moedas para fazer um troco.
- **Problema do Caixeiro Viajante:** Outro problema clássico que se beneficia da aplicação de algoritmos gulosos é o problema do caixeiro viajante, onde se busca o caminho mais curto para visitar um conjunto de cidades.

Os algoritmos gulosos, como demonstrado por, (Cormen, 1959), (Kruskal, 1956), e (Prim, 1957) são abordagens eficientes e amplamente utilizadas para resolver uma variedade de problemas em grafos ponderados. Eles compartilham a ideia central de fazer escolhas locais ótimas em cada etapa, na esperança de obter uma solução global ótima. (Kruskal, 1956) destaca isso em sua busca pela árvore de abrangência mínima de um grafo, enquanto aplicam princípios semelhantes na busca pelo caminho mais curto e pela árvore de abrangência mínima.

Esses autores demonstram que os algoritmos gulosos podem produzir soluções eficientes e próximas da ótima para certas classes de problemas. No entanto, é importante notar que esses algoritmos podem não ser adequados para todos os problemas e que a escolha de uma estratégia gulosa depende da natureza específica do problema em questão.

Algoritmos Gulosos: é a denominação dada para a técnica de projeto de algoritmos que tenta resolver o problema fazendo a escolha localmente ótima em cada fase com a esperança de encontrar uma correspondência global ótima.

Características dos Algoritmos Gulosos:

- **Tomada de decisões localmente ótimas:** Em cada etapa, o algoritmo escolhe a opção que parece "melhor" no momento, sem considerar o impacto global da decisão.
- **Construção iterativa da solução:** A solução final é construída gradualmente, passo a passo, com base nas decisões anteriores.

- **Simplicidade e rapidez:** Devido à sua natureza incremental, os algoritmos gulosos geralmente são mais fáceis de implementar e executar do que métodos de otimização mais complexos.

Vantagens dos Algoritmos Gulosos:

- **Eficiência**: os algoritmos gulosos são geralmente eficientes em termos de tempo de execução, pois fazem escolhas locais rápidas sem a necessidade de considerar todas as possibilidades. Eles são frequentemente utilizados em problemas de otimização em que é necessário encontrar uma solução próxima à ótima em um curto espaço de tempo.
- **Simplicidade**: conceito intuitivo e fácil de implementar.
- **Versatilidade**: aplicável a uma ampla gama de problemas de otimização.

Desvantagens dos Algoritmos Gulosos:

1. **Nem sempre fornecem a solução ótima**: os algoritmos gulosos geralmente tomam decisões com base em informações locais sem considerar o impacto global. Isso significa que eles podem falhar em encontrar a solução ótima em todos os casos. Em certas situações, uma escolha localmente ótima pode levar a uma solução global subótima.

2. **Dependência da escolha gulosa**: a eficácia dos algoritmos gulosos depende fortemente da condição de escolha gulosa. Se essa condição não for satisfeita para um determinado problema, o algoritmo guloso pode não fornecer uma solução adequada.

3. **Necessidade de prova de correção**: para garantir que um algoritmo guloso forneça a solução correta, é necessário fornecer uma prova de que a estratégia gulosa leva à solução ótima. Isso nem sempre é fácil e pode exigir uma análise cuidadosa da estrutura do problema. 4. Sensibilidade aos dados de entrada: alguns problemas podem ser mais adequados para abordagens gulosas do que outros. A eficácia dos algoritmos gulosos pode variar dependendo dos dados de entrada e das características específicas do problema.

Aplicação pratica do tema

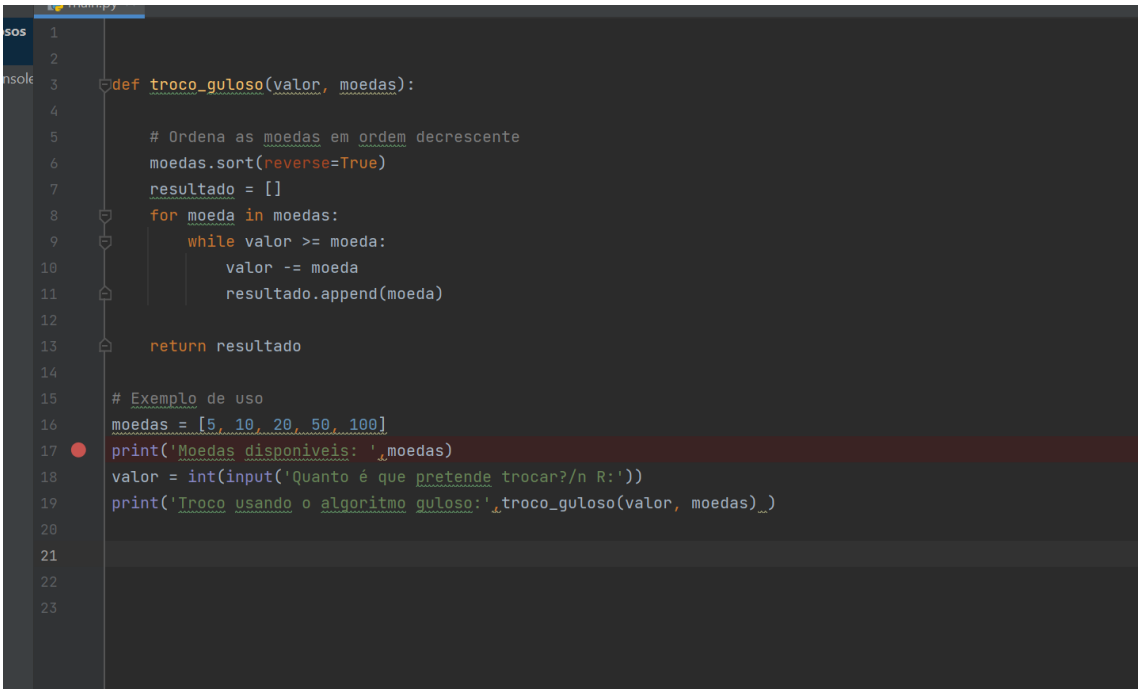
O presente tema algoritmos gulosos tem como parte pratica desenvolvida, resolver o problema de fazer troco utilizando moedas.

Problema: você é um caixa em uma loja e precisa dar o troco para os clientes. Você tem um conjunto de moedas disponíveis, cada uma com um valor diferente. Seu objetivo é encontrar a combinação de moedas que minimize o número total de moedas dadas como troco.

Detalhes:

- Você tem as seguintes moedas disponíveis: 5, 10, 20, 50 e 100 kwanzas.
- Um cliente comprou um item e pagou com uma nota de X kwanzas.
- Sua tarefa é calcular o troco ótimo usando o algoritmo guloso.

Algoritmo proposto (linguagem Phyton):



```
def troco_guloso(valor, moedas):

    # Ordena as moedas em ordem decrescente
    moedas.sort(reverse=True)
    resultado = []
    for moeda in moedas:
        while valor >= moeda:
            valor -= moeda
            resultado.append(moeda)

    return resultado

# Exemplo de uso
moedas = [5, 10, 20, 50, 100]
print('Moedas disponiveis: ', moedas)
valor = int(input('Quanto é que pretende trocar?/n R:'))
print('Troco usando o algoritmo guloso:', troco_guloso(valor, moedas) )
```

O algoritmo apresentado é um exemplo do paradigma de programação denominado Algoritmo Guloso (Greedy Algorithm). Esse paradigma consiste em resolver problemas tomando sempre a decisão que parece ser a melhor no momento, sem considerar as consequências futuras. No caso do algoritmo de troco, ele escolhe sempre a maior moeda disponível que seja menor ou igual ao valor restante a ser dado como troco, garantindo assim que o número total de moedas utilizado seja o menor possível.

A representação do algoritmo segue a estrutura de um algoritmo guloso, onde iteramos sobre as moedas disponíveis e, para cada moeda, escolhemos a maior possível que ainda caiba no valor restante do troco. Essa abordagem simplifica o problema e, na maioria dos casos, produz soluções aceitáveis, mas não necessariamente a melhor solução em todos os casos.

Explicação do código linha por linha:

- Definimos uma função **def troco_guloso** que recebe o valor do troco a ser dado e uma lista de moedas disponíveis.
- Criamos uma lista vazia chamada resultado para armazenar as moedas do troco.
- Ordenamos a lista de moedas em ordem decrescente para garantir que pegaremos as moedas de maior valor primeiro **moedas.soert (reverse = True)**.
- Começamos a iterar sobre as moedas disponíveis. Para cada moeda, enquanto o valor restante for maior ou igual ao valor da moeda atual, adicionamos essa moeda ao troco, subtraímos seu valor do valor restante e incrementamos o contador de moedas.
- Retornamos o troco e o total de moedas utilizadas.
- As moedas disponíveis para são exibidas.
- No exemplo de uso, o utilizador introduz o valor que deseja trocar.
- Chamamos a função **troco_guloso** com esses valores e imprimimos o resultado.

Analise do algoritmo

Função de Complexidade T(n): a complexidade do tempo depende do número de tipos diferentes de moedas e da quantidade total a ser trocada. No pior cenário, se tivermos n tipos diferentes de moedas, por causa da ordenação das moedas a complexidade será:

$$\mathbf{T(n) = nlogn}$$

• Melhor Caso (Omega(Ω)): o melhor caso ocorre quando o valor a ser trocado é exatamente divisível pela maior denominação disponível. Nesse caso, o tempo de execução seria dominado pela ordenação inicial das denominações das moedas, resultando em uma complexidade

$$\mathbf{\Omega(nlogn)}$$

• Pior Caso (BigO): o pior caso ocorre quando temos que usar o maior número possível de denominações menores para fazer o troco. Isso ainda seria limitado pela complexidade da ordenação inicial:

$$\mathbf{O(nlogn)}$$

• Caso Médio (theta(Θ)): como este é um algoritmo guloso e depende muito dos valores específicos envolvidos (tanto as denominações das moedas quanto o valor específico a ser trocado), é difícil definir um "caso médio" preciso sem mais informações. No entanto, em média, podemos esperar que a complexidade seja dominada pela ordenação inicial, ou seja, Θ(nlogn).

Normalmente, esperamos que os casos: melhor, pior e médio sejam diferentes. No entanto, para este algoritmo específico de troco guloso, a complexidade do tempo é dominada pela etapa de ordenação das moedas, independentemente do valor específico a ser trocado. Isso ocorre porque a ordenação é feita antes do início do processo de dar o troco e, portanto, é um custo fixo.

Conclusão

Ao concluir este trabalho sobre algoritmos gulosos, é evidente que esses métodos oferecem uma abordagem eficiente para resolver uma variedade de problemas de otimização. Sua simplicidade e capacidade de encontrar soluções aproximadas de forma rápida os tornam ferramentas valiosas em diversos domínios, desde computação até economia e logística. No entanto, é crucial reconhecer as limitações desses algoritmos, especialmente em situações onde a solução ótima é indispensável. Portanto, este estudo destaca não apenas as aplicações bem-sucedidas dos algoritmos gulosos, mas também a importância de combinar diferentes técnicas de resolução de problemas para alcançar resultados ainda mais robustos e precisos.

Bibliografia

Cormen, L. R. (1959). *Introduction to Algorithms.* MIT Press.

Kruskal, J. (1956). Graph problem. *Proceedings of the American Society*, 48-50.

Prim. (1957). *Redes de Conexões mais curtas e algumas generalizações.*

https://pt.wikipedia.org/wiki/Algoritmo_guloso

https://www.ime.usp.br/~is/ddt/mac5711-99/

https://www.ime.usp.br/~pf/analise_de_algoritmos/aula/guloso.html