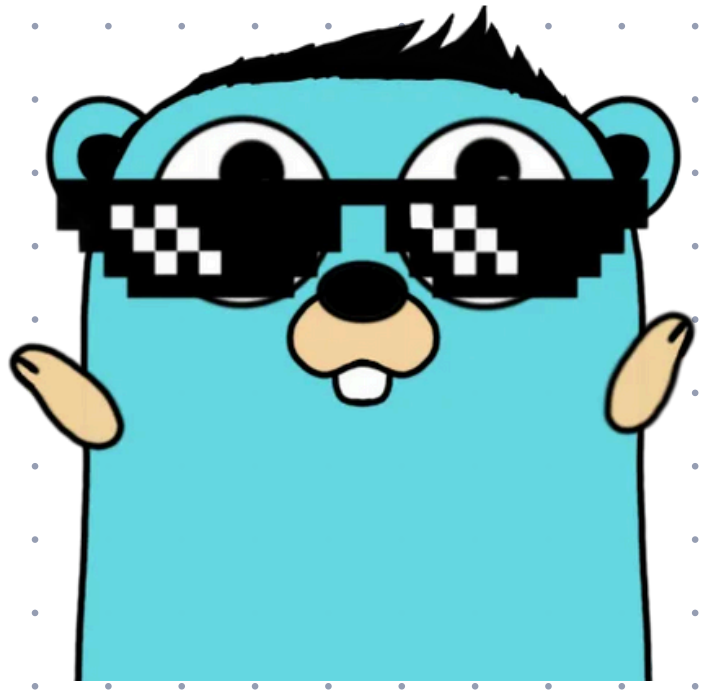
Análise de algoritmos

ALGORITMO GULOSO

Baseado no livro Entendendo Algoritmos, Um guia ilustrado para programadores e outros curiosos. Aditya Y. Bhargava, Novatec, 2017.



/evelynthrossell

Análise de algoritmos

ALGORITMO GULOSO

Você precisa organizar sua bagagem para realizar uma viagem que possui limitação de espaço e itens candidatos a irem na mala, uma vez que cada item possui peso e valor. O objetivo é selecionar itens que maximizem o valor total dentro da capacidade da mala.

Paralelamente, o algoritmo guloso funciona de forma que a melhor escolha é feita dentre as possíveis, baseado em função de avaliação de um critério, nesse caso, monetário.

Os algoritmos gulosos, conhecidos também como algoritmos gananciosos, são simples de escreverem, de fácil compreensão e é tido como uma heurística que dependendo de como for aplicada, pode ficar próxima ou distante da solução ótima. Em termos gerais, o algoritmo atende a:

- Ordenação de itens pelo valor de unidade de peso bem como valor monetário do item e;

Análise de algoritmos

ALGORITMO GULOSO

- Seleção de itens conforme ordenação até que a mala esteja cheia ou não haja mais itens.

O tempo de execução do algoritmo guloso é varíavel para cada contexto, geralmente baseado na estrutura de dados utilizada, heurística utilizada, tamanho de entrada e implementação, apresentando no pior caso o tempo de execução de $O(n^2)$.

Por outro lado, quando tratamos de problemas computacionalmente inviáveis e custosos, os algoritmos de aproximação nos ajudam a encontrar a solução ótima com tempo de execução no pior caso de O(2^n).

Análise de algoritmos

ALGORITMO DE APROXIMAÇÃO

Destaca-se por garantir a solução dentro de um fator de aproximação da solução ótima. Os algoritmos de aproximação são utilizados quando necessita-se de rápida resposta do que aguardar por uma solução ótima, complexidade operacional quando envolve grandes instâncias de um problema e custos elevados ou até mesmo quando um problema é considerado como NP-difícil (quando não há algoritmo eficiente) com tempo de execução polinomial capaz de encontrar a solução exata para todos os casos.

O que normalmente ocorre é que um *array* inicial é recebido e convertido em conjuntos (sem conter itens repetidos) e para representar em uma tabela *hash* para cada estação. Os valores representados pela tabela *hash* são conjuntos associados à uma chave (nome do vértice) e ajudam a compor o cálculo do conjunto final.

Análise de algoritmos

ALGORITMO DE APROXIMAÇÃO

Se o problema for considerado com tempo de execução polinomial, é considerado eficiente, já se o tempo for considerado quase-polinomial, indicará lentidão quando comparado ao polinomial. E, por fim, o tempo exponencial está presente em casos isolados considerados de difícil resolução.

/evelynthrossell

Análise de algoritmos

PROBLEMA NP-COMPLETO

A melhor forma de solucionarmos o problema dos conjuntos, é calculando cada conjunto possível através do problema do caixeiro-viajante, percorrendo por todos os pontos por uma vez e buscando minimizar a distância percorrida. Podemos deduzir que:

- Quando o cálculo de partida é desconhecido, deve-se definir o caminho ideal bem como seu local de partida;

- O cálculo dos conjuntos baseia-se na função fatorial.

O ponto de intersecção entre o problema do caixeiro-viajante e o problema dos conjuntos é que são calculadas cada solução possível e escolhe-se a menor, podendo ser considerados como NP-completos. É sabido da dificuldade de identificar quando um problema é fácil de ser resolvido de um problema NP-completo e para isso, vamos listar alguns indícios de que um problema poderá ser categorizado como NP-completo:

Análise de algoritmos

PROBLEMA NP-COMPLETO

- Executar rapidamente para certos itens, enquanto roda lentamente conforme aumento dos itens;
- Referir-se ao problema como todas as combinações de algo;
- Quando não é possível dividir o problema em subproblemas menores;
- O problema apresentar uma sequência (problema do caixeiro-viajante) de difícil resolução e for baseado em conjuntos e;
- Se for possível reescrever o problema de cobertura mínima de conjunto ou o problema do caixeiro-viajante.

APLICAÇÕES

Amplamente empregado por buscar a solução ideal dentre de inúmeras possibilidades por meio dos algoritmos de aproximação, os problemas NP-completos encontram-se nos seguintes setores:

Análise de algoritmos

PROBLEMA NP-COMPLETO

OTIMIZAÇÃO

- Logística: roteamento de veículos, planejamento de produção, gerenciamento de estoque;
- Engenharia: circuitos integrados, corte de materiais, alocação de recursos;
- Finanças: portfólio de investimentos e otimização de carteiras;
- Planjeamento urbano: roteamento de ônibus, localização de serviços públicos.

INTELIGÊNCIA ARTIFICIAL

- Aprendizados de máquina: seleção de características, otimização de hiperparâmetros;
- Visão computacional: segmentação de imagens, reconhecimento de padrões;
- Processamento de linguagem natural: tradução automática, análise de sentimentos.

Análise de algoritmos

PROBLEMA NP-COMPLETO

BIOINFORMÁTICA

- Alinhamento de sequências: comparação de sequências genéticas;
- Dobramento de proteínas: predição de estrutura tridimensional de proteínas.

TEORIA DOS GRAFOS

- Redes sociais: análise de comunidades, detecção de anomalias;
- Redes de comunicação: roteamento de pacotes, projetos de redes.

CRIPTOGRAFIA

- Quebra de senhas: chave de descriptografia de dados;
- Ataques a criptossitemas: exploração de vulnerabilidades de sistemas com criptografia.

/evelynthrossell

Análise de algoritmos

PROBLEMA NP-COMPLETO

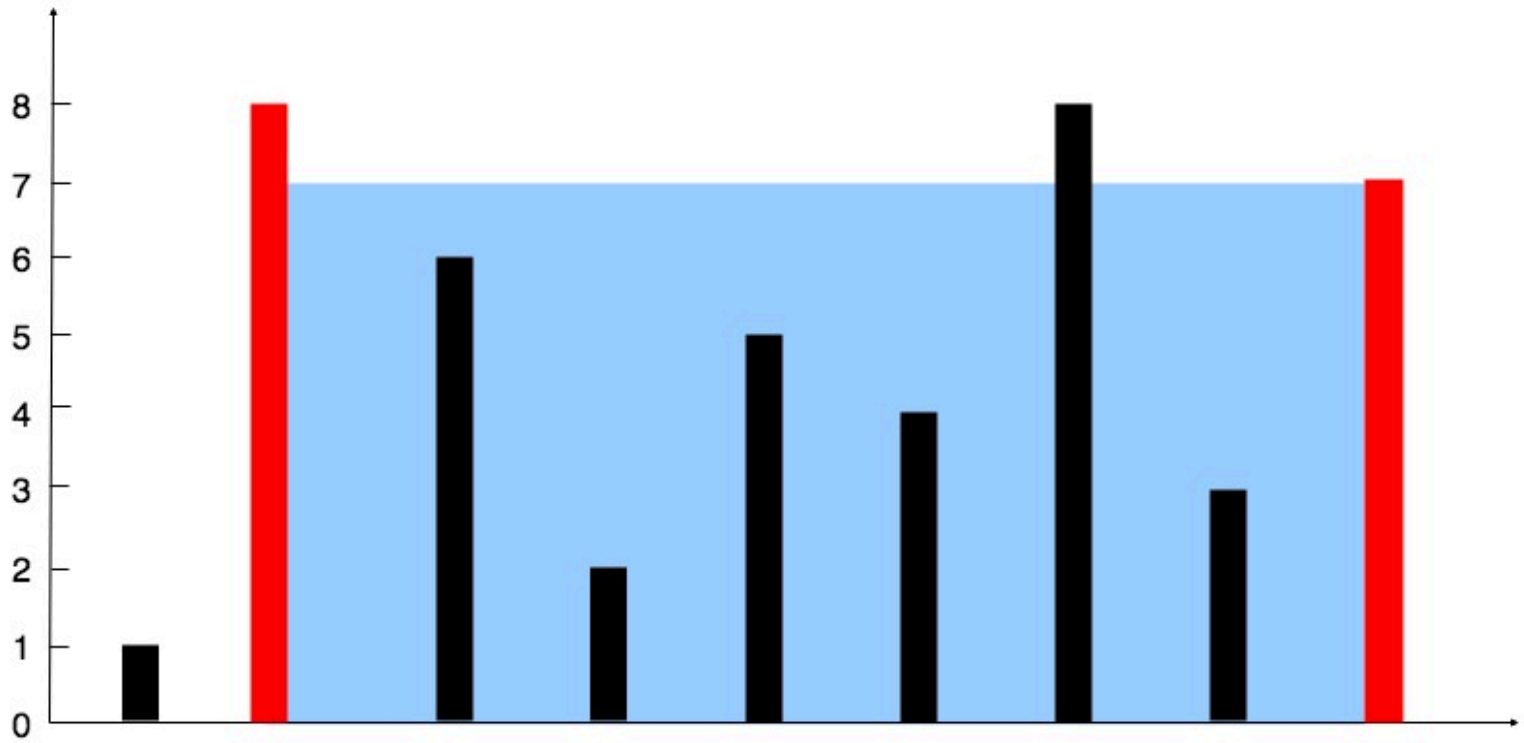
Veremos à seguir um problema que envolve o uso do algoritmo guloso de solução ótima dentro do fator de aproximação.

Dado um *array* de inteiros *height* de comprimento *n*, existem *n* linhas verticais desenhadas de modo que os dois pontos finais da *i*-ésima linha são *(i,0)* e *(i, height[i])*. Encontre duas linhas que, juntamente com o eixo *x*, formem um contêiner, de modo que o contêiner contenha a maior quantidade de água.

Retorne a quantidade máxima de água que um contêiner pode armazenar. Observe que você não pode inclinar o contêiner.

Análise de algoritmos

PROBLEMA NP-COMPLETO



Extraído do *Leetcode Container With Most Water*.

Entrada: **height** = [1,8,6,2,5,4,8,3,7]
Saída: 49
Explicação: As linhas verticais são representadas pelo *array* [1,8,6,2,5,4,8,3,7]. Nesse caso, a área máxima de água (área em azul) que o contêiner por conter é 49.

Análise de algoritmos

PROBLEMA NP-COMPLETO

Entrada: **height** = [1,1]
Saída: 1

Restrições:

- n == height.length
- 2 <= n <= 105
- 0 <= height[i] <= 104

/evelynthrossell

PROBLEMA NP-COMPLETO

```
package main

import (
        "log"
        "math"
)

func max(x, y int) int {
    return int(math.Max(float64(x), float64(y)))
}
```

/evelynthrossell

PROBLEMA NP-COMPLETO

```go
func min(x, y int) int {
    return int(math.Min(float64(x), float64(y)))
}

func maximumContainerArea(height []int) int {
    left, right := 0, len(height) -1
    maxArea := 0

    for left < right {
        width := right - left
        currentHeight := min(height[left], height[right])
        currentArea := width * currentHeight
```

/evelynthrossell

PROBLEMA NP-COMPLETO

```
        log.Printf("Current Area: %d, Left: %d, Right:
%d\n", currentArea, left, right)

        maxArea = max(maxArea, currentArea)

        if height[left] < height[right] {
            left++
        } else { right-- }
    }
    return maxArea
}
```

/evelynthrossell

PROBLEMA NP-COMPLETO

```go
func main() {
    height := []int{1, 8, 6, 2, 5, 4, 8, 3, 7}
    maxArea := maximumContainerArea(height)

    log.Printf("Maximum Area: %d\n", maxArea)
}
```

/evelynthrossell

SAÍDA DO PROGRAMA

CENÁRIO OTIMISTA

2024/12/31 13:31:51 Current Area: 8, Left: 0, Right: 8
2024/12/31 13:31:51 Current Area: 49, Left: 1, Right: 8
2024/12/31 13:31:51 Current Area: 18, Left: 1, Right: 7
2024/12/31 13:31:51 Current Area: 40, Left: 1, Right: 6
2024/12/31 13:31:51 Current Area: 16, Left: 1, Right: 5
2024/12/31 13:31:51 Current Area: 15, Left: 1, Right: 4
2024/12/31 13:31:51 Current Area: 4, Left: 1, Right: 3
2024/12/31 13:31:51 Current Area: 6, Left: 1, Right: 2
2024/12/31 13:31:51 Maximum Area: 49

Análise de algoritmos

CONSIDERAÇÕES

Para o exemplo apresentado, seguiu-se as boas práticas de desenvolvimento com *clean code* e SOLID, na tentativa de simular um cenário otimista.
Essa iniciativa vai de encontro com a ideia de trazer conteúdos relevantes altamente abordados em processos seletivos e desmitificar a ideia de algoritmos e estrutura de dados. Espero que seja de bom proveito e bons estudos.

/evelynthrossell