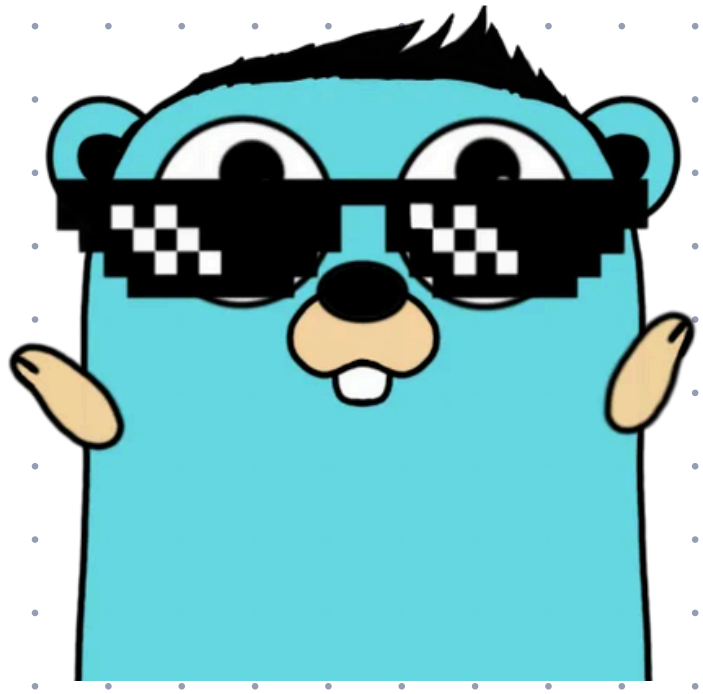
Análise de algoritmos

PESQUISA EM LARGURA

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



/evelyncristinioliveira

Análise de algoritmos

PESQUISA EM LARGURA

A pesquisa em largura é conhecida como *Breadth-First Search*, é algoritmo que responde caso exista caminho ou não e qual é o menor caminho entre dois objetos. Por meio do Problema do Caminho Mínimo, a estrutura de grafos permite que seja solucionado fazendo uso de seu conjunto de conexões, formado por vértices (pontos de conexão) e arestas (ligações desses pontos). Quando há conexão de vértices com outros vértices, são chamados de vizinhos e é uma forma de modelarmos eventos conectados.

Imagine que você se encontra no meio de um labirinto buscando uma saída junto ao seu grupo de amigos dispostos a te ajudar. Dividindo em grupos menores, marcando o caminho percorrido e comunicando entre si, acontecará de se encontrarem e seguirem juntos explorando todas as possibilidades. Com essa estratégia coordenada, o labirinto seria explorado de maneira que fossem evitados becos sem sída e fossem concentrados esforços nos caminhos mais prováveis de se encontrar a saída.

Análise de algoritmos

PESQUISA EM LARGURA

Podemos representar as relações desse grafo fazendo uso da tabela *hash* (estrutura estudada anteriormente), mapeando um vértice a todos os seus vizinhos na estrutura chave-valor e não importando a ordem que são adicionados. Podem existir também um vértice que não possui vizinhos, denominado de dígrafo (grafo não direcionado).

Quando temos nós que aparecem mais de uma vez se relacionando no grafo, uma estratégia ao persistir é marcar o campo como verificado e evitar que haja *loop* infinito, criando uma lista de nós já verificados.

Referente aos tempos de execução Big O é de O(número de arestas) ao buscar uma saída no labirinto por todo o grafo. Se optado por ter uma lista de caminhos verificados, terá tempo de O(número de caminhos percorridos) no total, já a pesquisa em largura possui tempo de O(número de caminhos percorridos + número de arestas), escrito como **O**(**V** (número de vértices) + **A** (número de arestas)).

Análise de algoritmos

PESQUISA EM LARGURA

Quando optamos por uma lista de verificação, a estrutura que possui comportamento compatível é a fila (FIFO), onde os itens adicionados primeiro na lista são desenfileirados e verificados primeiro.

APLICAÇÕES

Em estrutura de dados, podemos encontrar o algoritmo de grafos nos seguintes casos:

- Redes sociais: encontrar conexões em comum e sugerir novas conexões;
- Roteamento de rede de computadores: encontrar o caminho mais rápido para a internet (entre computador e servidor) e rotear mensagens instantâneas;
- Jogos: devendar e encontrar o caminho mais curto para o objetivo;
- Busca por palavras em dicionários;
- Simulação de epidemias: simular propagação de doenças, mapeando conexões entre pessoas e prevendo o contágio e;

Análise de algoritmos

PESQUISA EM LARGURA

- Inteligência artificial: planejar o trajeto de um robô autônomo (explorar ambiente, identificar obstáculos e encontrar o caminho mais seguro para alcançar o objetivo) e análise de redes complexas (identifica padrões e relações de elementos da rede).

De forma geral, as vantagens da implementação de um algoritmo de grafo consistem na versatilidade, simplicidade, eficiência e completude. Por outro lado, temos como desvantagens o consumo de memória, ineficiência em alguns grafos e pode não encontrar o caminho ideal para todos os casos.

Para o algoritmo abaixo, teremos um jogo de palavras que você precisa transformar uma palavra em outra, modificando uma letra pro vez. O objetivo é encontrar a sequência de transformações mais curta possível, usando apenas um dicionário fornecido.

Análise de algoritmos

PESQUISA EM LARGURA

Temos também restrições a serem respeitadas como:

- Termos a palavra inicial e final contendo o mesmo tamanho;
- O tamanho do dicionário deve estar entre 1 à 5000;
- Todas as palavras do dicionário deve ter o mesmo tamanho que a palavra inicial e final;
- As palavras do dicionário devem ser únicas e;
- A palavra inicial e final não devem ser iguais.

GRAFOS BFS

```go
package main

import "fmt"

type WordVerifier interface {
  IsValid(word string) bool
}

type wordMapVerifier struct {
  wordMap map[string]bool
}
```

/evelyncristinioliveira

GRAFOS BFS

```go
func (w *wordMapVerifier) IsValid(word string) bool {
	_, ok := w.wordMap[word]
	return ok
}

func createWordVerifier(wordList []string) WordVerifier {
	wordMap := make(map[string]bool)
	for _, word := range wordList {
		wordMap[word] = true
	}
	return &wordMapVerifier{wordMap: wordMap}
}
```

/evelyncristinioliveira

GRAFOS BFS

```go
func buildNewWord(word string, i int, ch rune) string {
    return word[:i] + string(ch) + word[i+1:]
}

func bfs(startWord string, endWord string, verifier
WordVerifier, wordList []string) int {
    if !isValidWordList(wordList) {
        return 0
    }

    queue := []string{startWord}
    steps := 0
```

/evelyncristinioliveira

GRAFOS BFS

```
for len(queue) > 0 {
    for _, word := range queue {
        if word == endWord {
            return steps + 1
        }
        for i := 0; i < len(word); i++ {
            for ch := 'a'; ch <= 'z'; ch++ {
                newWord := buildNewWord(word, i, ch)
                if verifier.IsValid(newWord) {
                    queue = append(queue, newWord)
                }
            }
```

/evelyncristinioliveira

GRAFOS BFS

```
                }
            }
            steps++
            queue = queue[1:]
        }
        return 0
    }

func isValidWordList(wordList []string) bool {
        return len(wordList) > 0
}
```

/evelyncristinioliveira

GRAFOS BFS

```
func ladderLength(beginWord string, endWord string,
wordList []string) int {
	verifier := createWordVerifier(wordList)
	return bfs(beginWord, endWord, verifier, wordList)
}

func main() {
	beginWord := "hit"
	endWord := "cog"
	wordList := []string{"hot", "dot", "dog", "lot", "log",
"cog"}
```

/evelyncristinioliveira

GRAFOS BFS

```
    minLength := ladderLength(beginWord, endWord,
wordList)

    if minLength > 0 {
        fmt.Printf("minimum word ladder size: %d\n",
minLength)
    } else {
        fmt.Println("it is not possible to transform the initial
word into the final one using the list provided.")
  }
}
```

/evelyncristinioliveira

SAÍDA DO PROGRAMA

CENÁRIO OTIMISTA

minimum word ladder size: 5

CENÁRIO PESSIMISTA

it is not possible to transform the initial word into the final one using the list provided.

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.

/evelyncristinioliveira