Programação Dinâmica

Vagner Seibert[1], Grazziano Borges Fagundes[1], Miguel Silva[1],

[1]Universidade Federal de Pelotas

UFPEL, Julho 2024

SUMÁRIO

- Introdução
- Algoritmos
- Exemplos
- Discussão

INTRODUÇÃO

- Definição

INTRODUÇÃO

- Definição
- Historia

INTRODUÇÃO

- Definição
- Historia
- Paralelos

INTRODUÇÃO

- Definição
- Historia
- Paralelos
- Aplicações

Introdução à Memoization

- **Técnica essencial** em programação dinâmica.
- **Evita trabalho repetido** com armazenamento de resultados.
- **Melhora a eficiência** em problemas recursivos, como Fibonacci.

Explicação Teórica

- Para calcular `fibonacci(n)`, precisamos de `fibonacci(n-1)` e `fibonacci(n-2)`.
- **Sem memoization**: repetição de cálculos e complexidade alta.
- **Com memoization**: resultados armazenados e reutilizados, economizando tempo e recursos.

Exemplo Prático: Sequência de Fibonacci

- A sequência de Fibonacci é um clássico exemplo para ilustrar a programação dinâmica.
- A sequência é definida como:
- Observe a fórmula: $F(n) = F(n-1) + F(n-2)$ é usada para calcular os próximos termos da sequência, a partir dos valores iniciais $F(0) = 0$ e $F(1) = 1$.

$$F(n) = \begin{cases} 0\,, & \text{se } n = 0\,; \\ 1, & \text{se } n = 1; \\ F(n-1) + F(n-2) & \text{outros casos.} \end{cases}$$

Implementação Prática: Fibonacci

Exemplo com recursão simples: `fibonacci(n)`

- Complexidade exponencial devido à repetição de cálculos.

```python
def fibonacci(n):
    if n == 0:
        return 0
    elif n == 1:
        return 1
    else:
        return fibonacci(n-1) + fibonacci(n-2)
```

Implementação com Memoization

Introduzindo memoization: usando um dicionário para armazenar resultados

- Reduz a complexidade para `O(n)`.

```python
def fibonacci_memo(n, memo={}):
    if n in memo:
        return memo[n]
    if n == 0:
        result = 0
    elif n == 1:
        result = 1
    else:
        result = fibonacci_memo(n-1, memo) + fibonacci_memo(n-2, memo)
    memo[n] = result
    return result
```

Implementação Prática: Fibonacci

Exemplo com recursão simples: `fibonacci(n)`

- Complexidade exponencial devido à repetição de cálculos.

- Complexidade exponencial devido à repetição de cálculos.

```python
def fibonacci(n):
    if n == 0:
        return 0
    elif n == 1:
        return 1
    else:
        return fibonacci(n-1) + fibonacci(n-2)
```

Implementação com Memoization

Introduzindo memoization: usando um dicionário para armazenar resultados

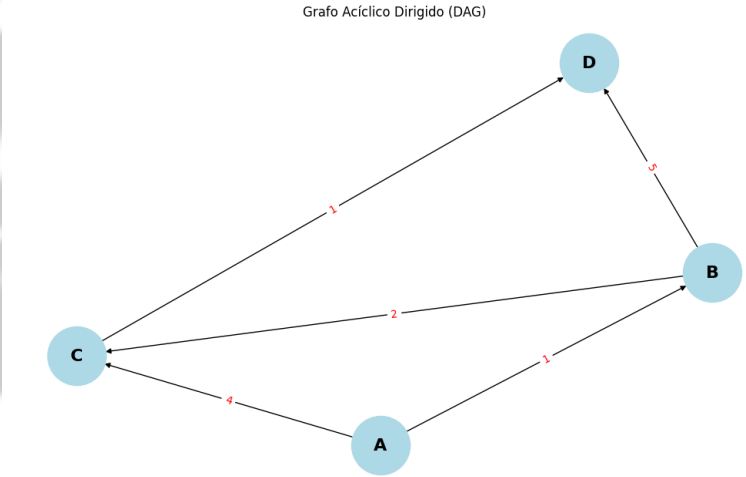
- Reduz a complexidade para `O(n)`.
- A complexidade é reduzida para `O(n)`, armazenando resultados já calculados.

```python
def fibonacci_memo(n, memo={}):
    if n in memo:
        return memo[n]
    if n == 0:
        result = 0
    elif n == 1:
        result = 1
    else:
        result = fibonacci_memo(n-1, memo) + fibonacci_memo(n-2, memo)
    memo[n] = result
    return result
```

Exemplo: Grafos Acíclicos Direcionados (DAGs)

- Grafos Acíclicos Direcionados (DAGs)
- A programação dinâmica pode ser aplicada usando uma ordenação topológica dos vértices.

Exemplo: Grafos Acíclicos Direcionados (DAGs)

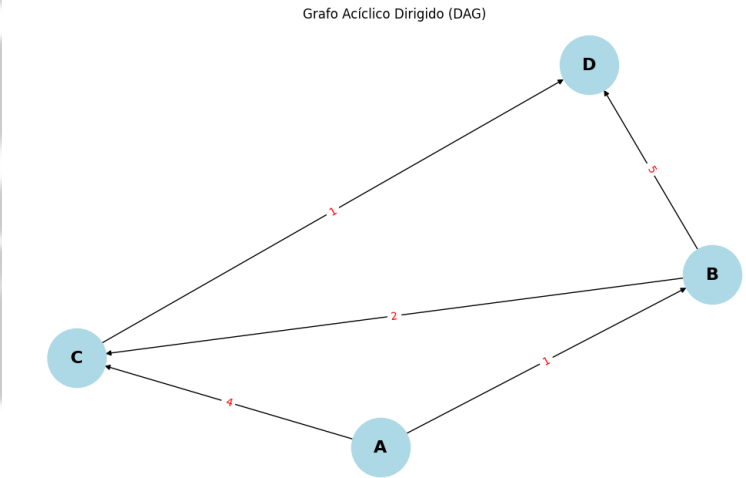


Exemplo Prático

```python
def shortest_path_dag(graph, start):
    # 1. Obtém a ordenação topológica do grafo
    order = topological_sort(graph)

    # 2. Inicializa as distâncias de todos os vértices como infinito
    dist = {v: float("inf") for v in graph}
    dist[start] = 0

    # 3. Processa os vértices na ordem topológica
    for u in order:
        for v in graph[u]:
            # 4. Relaxa a aresta (u, v)
            if dist[v] > dist[u] + graph[u][v]:
                dist[v] = dist[u] + graph[u][v]

    # 5. Retorna as distâncias calculadas
    return dist
```

```python
start_vertex = "A"
graph = {
    "A": {"B": 1, "C": 4},
    "B": {"C": 2, "D": 5},
    "C": {"D": 1},
    "D": {}
}
```

Resultado



Resultado = 'A': 0, 'B': 1, 'C': 3, 'D': 4

DISCUSSÃO

Sobre programação dinâmica

- Permite a solução eficiente de problemas complexos que, de outra forma, seriam computacionalmente inviáveis;
- Com a utilização de técnicas como memoization, é possível evitar redundâncias e reduzir drasticamente o tempo de execução de algoritmos;
- Suas aplicações vão desde problemas de otimização clássicos até questões práticas em engenharia e ciência;

REFERÊNCIAS

[1] Bellman, Richard. ”Dynamic programming.” science 153.3731 (1966): 34-37.
[2] Chen, Xiaoxi. ”A Comparison of Greedy Algorithm and Dynamic Programming Algorithm.” SHS Web of Conferences. Vol. 144. EDP Sciences, 2022.

OBRIGADO!

Dúvidas?