Complexidade de Algoritmos

Prof. Diego Buchinger
diego.buchinger@outlook.com
diego.buchinger@udesc.br

Prof. Cristiano Damiani Vasconcellos
cristiano.vasconcellos@udesc.br

Abordagens para Resolução de Problemas

Programação Dinâmica
(*Dynamic Programming*)

Uma abordagem auxiliar para reduzir a computação em problemas de otimização.

Consiste em dividir o problema original em subproblemas mais simples, resolvê-los, armazenar os resultados em memória e consultar estes resultados posteriormente a fim de reduzir o espaço de busca.

Resultados são calculados apenas uma vez (nunca são recalculados por outra iteração)

Programação Dinâmica
(*Dynamic Programming*)

Alguns problemas que podem ser resolvidos com programação dinâmica:

- Fibonacci
- Problema do Menor Troco não canônico
- Problema da Mochila Binária (0/1)
- Maior Subsequência Comum
- Multiplicação de Cadeias de Matrizes
- Sokoban

Fibonacci

O modelo recursivo de Fibonacci pode ser extremamente ruim dependendo da implementação:

```
int fib(int n){
    if( n<=1 ) return n;
    return fib(n-1) + fib(n-2);
}
```

Fibonacci

Para entender a programação dinâmica, talvez o problema da sequencia de Fibonacci seja o exemplo mais simples para demonstrar o seu uso.

A ideia é utilizar uma estrutura auxiliar para armazenar os resultados e utilizá-los posteriormente

Fibonacci

Abordagem *top-down* (*Memoization*)

O vetor **F** é inicializado antes da chamada da função com um valor padronizado que nunca será registrado [ex: -1]

```
Fib(n)
    se n ≤ 1 então
        retorne n
    senão
        se F[n] está indefinido
        F[n] ← Fib(n − 1) + Fib(n − 2)
        retorne F[n]
```

Fibonacci
Abordagem *bottom-up*

```
Fib(n)
    F[0] = 0
    F[1] = 1
    para i ← 2 até n
        F[i] = F[i − 2] + F[i − 1]
    retorne F[n]
```

Problema do Melhor Troco não canônico

Como visto anteriormente, para sistemas monetários **não canônicos** a abordagem gulosa pode não resultar no melhor resultado!

Nesse caso, podemos usar programação dinâmica como um método eficiente para encontrar o melhor troco:

Exemplo Trivial: Moedas disponíveis: 1, 3, 4

Valor do troco: $ 0.06

Qual é o número mínimo de moedas para o troco?

Problema do Melhor Troco não canônico

Exemplo Trivial: Moedas disponíveis: 1, 3, 4

Valor do troco: $ 0.06

Considere as moedas como um vetor: M = { 1, 3, 4 }

Inicialmente assumimos que o troco para $ 0.00 precisa de zero moedas

T(0) = 0

E seguimos iterativamente até o troco desejado usando a fórmula:

T(n) = min(1+T(n-M[0]) , 1+T(n-M[1]) , 1+T(n-M[2]))

Problema do Melhor Troco não canônico

Exemplo Trivial: Moedas disponíveis: 1, 3, 4

Valor do troco: $ 0.06

$$T(0) = 0$$

$$T(n) = \min(\ 1+T(\ n-M[0]\ )\ ,\ 1+T(\ n-M[1]\ )\ ,\ 1+T(\ n-M[2]\ )\ )$$

$$T(1) = \min(\ 1+T(1-1)\ ,\ 1+\cancel{T(1-3)}\ ,\ 1+\cancel{T(1-4)}\ ) = 1$$

$$T(2) = \min(\ 1+T(2-1)\ ,\ 1+\cancel{T(2-3)}\ ,\ 1+\cancel{T(2-4)}\ ) = 2$$

$$T(3) = \min(\ 1+T(3-1)\ ,\ 1+T(3-3)\ ,\ 1+\cancel{T(3-4)}\ ) = 1$$

(...)

Complexidade de tempo => Θ (n * |M|)

Complexidade de espaço => Θ (n)

Problema da Mochila Binária (0/1)

Dark Version: Um ladrão está assaltando uma loja e possui uma mochila que pode carregar até **P** kg sem arrebentar. Sabendo os pesos e valores dos itens, qual é o maior valor possível que o ladrão conseguirá roubar?

- Na versão da mochila binária os elementos só podem ser pegos como um todo (não podemos pegar metade de um item – ou pega, ou não pega).
- **Exemplo**: Mochila com capacidade 9 kg

Objetos	1	2	3	4	5	6
Peso	1	3	2	5	7	9
Valor	3	2	4	7	9	12

Problema da Mochila Binária (0/1)

Solução Matricial

Montamos uma matriz para armazenar as melhores soluções intermediárias e as usamos para gerar as próximas soluções;

Objetos	1	2	3	4	5	6
Peso	1	3	2	5	7	9
Valor	3	2	4	7	9	12

UDESC

Problema da Mochila Binária (0/1)

Solução Matricial

Objetos	**1**	**2**	**3**	**4**	**5**	**6**
Peso	1	3	2	5	7	9
Valor	3	2	4	7	9	12

Item / Capacidade	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
1	0	3	3	3	3	3	3	3	3	3
2	0	3	3	3	5	5	5	5	5	5
3	0	3	4	7	7	7	9	9	9	9
4	0	3	4	7	7	7	10	11	14	14
5	0	3	4	7	7	7	10	11	14	14
6	0	3	4	7	7	7	10	11	14	**14**

```
m[item][k] = max( m[item-1][k], valor + m[item-1][k-peso] )
```

Problema da Mochila Binária (0/1)

Solução Matricial

Complexidade de tempo => Θ (peso * qtd-itens)

Complexidade de espaço => Θ (peso * qtd-itens)

Maior Subsequência Comum

Longest Common Subsequence

Dadas duas strings S e T, qual é maior subsequencia (sequencia não necessariamente contígua), da esquerda para a direita, entre estas strings?

- **Exemplo**: S = ABAZDC

  T = BACBAD

Resposta ?

Resolução similar a abordagem matricial para resolver o problema da mochila.

Maior Subsequência Comum

Longest Common Subsequence

Solução Matricial

pA / pB		A	B	A	Z	D	C
	0	0	0	0	0	0	0
B	0						
A	0						
C	0						
B	0						
A	0						
D	0						

1 – Preenche linha e coluna fictícia

2 – usar regra:

```
m[i][j] = max( m[i-1][j],
               m[i][j-1] )

SE pA[i] == pB[j]:
    m[i][j] = max( m[i][j],
                   m[i-1][j-1] + 1 )
```

Maior Subsequência Comum

Longest Common Subsequence

Solução Matricial

pA / pB		A	B	A	Z	D	C
	0	0	0	0	0	0	0
B	0	0	1	1	1	1	1
A	0	1	1	2	2	2	2
C	0	1	1	2	2	2	3
B	0	1	2	2	2	2	3
A	0	1	2	3	3	3	3
D	0	1	2	3	3	4	4

Multiplicação de Cadeias de Matrizes

Matrix Chain Multiplication

Dadas uma sequencia de *n* matrizes, qual é a melhor ordem para multiplicá-las a fim de realizar menos multiplicações.

OBS: a multiplicação entre matrizes é associativa, logo (AB)(CD) = A(B(CD)) = (AB)C)D = (A(BC))D = A((BC)D)

- **Exemplo**: São dadas 4 matrizes com os seguintes tamanhos:

$M_1 = (2,3)$	$M_1 * M_2 = 2 * 3 * 6 = 36$
$M_2 = (3,6)$	$M_2 * M_3 = 3 * 6 * 4 = 72$
$M_3 = (6,4)$	$M_3 * M_4 = 6 * 4 * 5 = 120$
$M_4 = (4,5)$	

Multiplicação de Cadeias de Matrizes

Matrix Chain Multiplication

- **Exemplo**: São dadas 4 matrizes com os seguintes tamanhos:

$M_1 = (2,3)$ $\quad\quad$ $M_1 * M_2 = 2 * 3 * 6 = 36$

$M_2 = (3,6)$ $\quad\quad$ $M_2 * M_3 = 3 * 6 * 4 = 72$

$M_3 = (6,4)$ $\quad\quad$ $M_3 * M_4 = 6 * 4 * 5 = 120$

$M_4 = (4,5)$

MATRIZ	1	2	3	4
1	0	36		
2		0	72	
3			0	120
4				0

MATRIZ	1	2	3	4
1	0	1		
2		0	2	
3			0	3
4				0

Multiplicação de Cadeias de Matrizes

Matrix Chain Multiplication

- **Exemplo**: São dadas 4 matrizes com os seguintes tamanhos:

$M_1 = (2,3)$ $\quad (M_1 * M_2)\, M_3 = 36 + 2 * 6 * 4 = 84$

$M_2 = (3,6)$ $\quad M_1\, (M_2 * M_3) = 2 * 3 * 4 + 72 = 96$

$M_3 = (6,4)$ $\quad (M_2 * M_3)\, M_4 = 72 + 3 * 4 * 5 = 132$

$M_4 = (4,5)$ $\quad M_2\, (M_3 * M_4) = 3 * 6 * 5 + 120 = 210$

MATRIZ	1	2	3	4
1	0	36	84	
2		0	72	132
3			0	120
4				0

MATRIZ	1	2	3	4
1	0	1	2	
2		0	2	3
3			0	3
4				0

Multiplicação de Cadeias de Matrizes

Matrix Chain Multiplication

- **Exemplo**: São dadas 4 matrizes com os seguintes tamanhos:

$M_1 = (2,3)$ $(M_1 M_2 M_3) M_4 = 84 + 2 * 4 * 5 = 124$

$M_2 = (3,6)$ $(M_1 M_2)*(M_3 M_4) = 36+2*6*5+120 = 216$

$M_3 = (6,4)$ $M_1 (M_2 M_3 M_4) = 2 * 3 * 5 + 132 = 162$

$M_4 = (4,5)$

MATRIZ	1	2	3	4
1	0	36	84	124
2		0	72	132
3			0	120
4				0

MATRIZ	1	2	3	4
1	0	1	2	3
2		0	2	3
3			0	3
4				0

Multiplicação de Cadeias de Matrizes

Matrix Chain Multiplication

PSEUDOCÓDIGO

```
MATRIX-CHAIN-MULT(   v[n]  )
    para  i de 1 a n:                      // zera a primeira linha
        m[ i , i ] = 0
    para  c de 2 a n:                      // para todas as linhas posteriores
        para  i de 1 a (n - c + 1):        // para todos os elem. da diagonal
            j = i + c - 1
            m[ i , j ] = ∞
            para  k de i a (j - 1):        // para todas as permutações
                q = m[ i , k ] + m[ k+1 , j ] + v[ i-1 ] * v[ k ] * v[ j ]
                se  q < m[ i , j ]:
                    m[ i , j ] = q
                    s[ i, j] = k
```

Multiplicação de Cadeias de Matrizes

Matrix Chain Multiplication

PSEUDOCÓDIGO

```
PRINT-EQUACAO (  s[n][n] , i , j  )
    se i == j
        print  "M" + i
    senão
        print  "("
        PRINT-EQUACAO(   s , i , s[ i , j ] )
        PRINT-EQUACAO(   s , s[ i , j ] + 1 , j )
        print  ")"
```

MATRIZ	1	2	3	4
1	0	1	2	3
2		0	2	3
3			0	3
4				0

```
P-E (1,4)
    P-E(1,3)
        P-E(1,2)
            P-E(1,1) => "M1"
            P-E(2,2) => "M2"
        P-E(3,3) => "M3"
    P-E(4,4) => "M4"
```

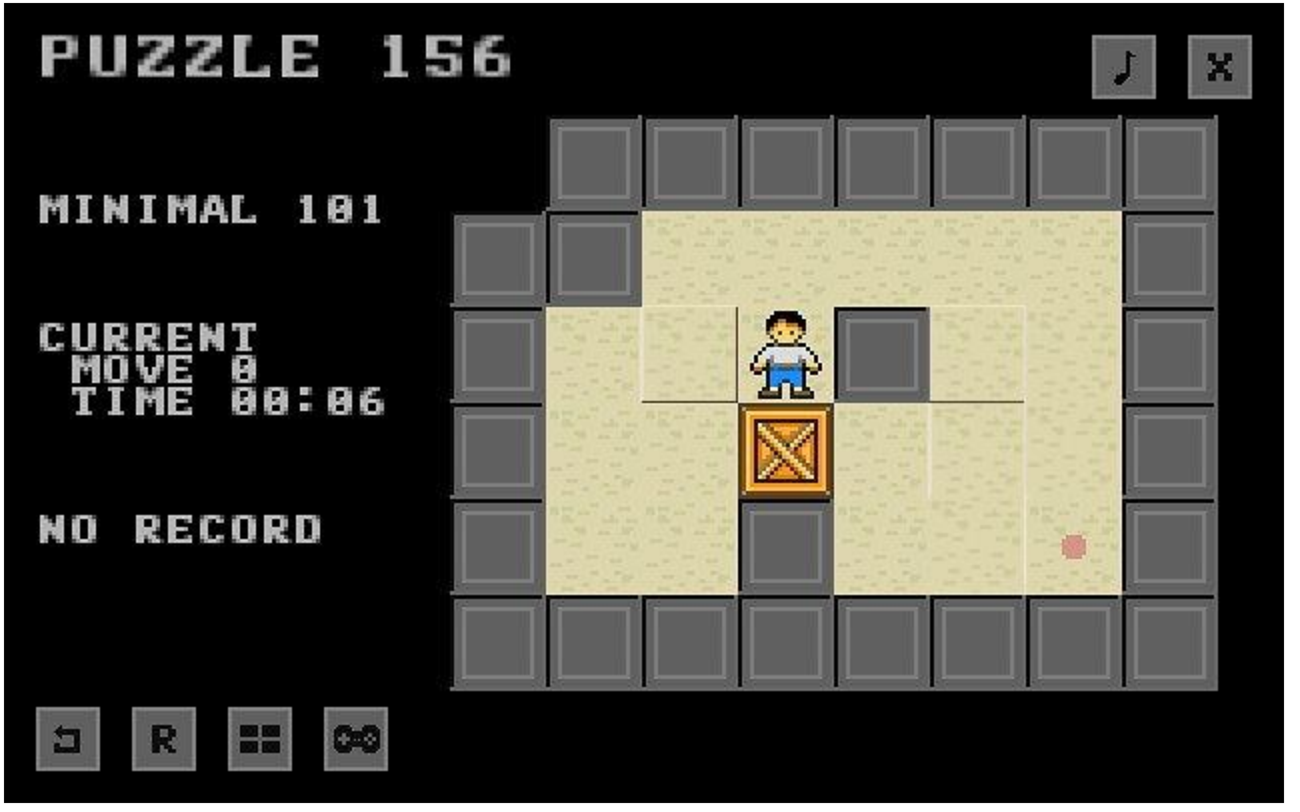
(M1 M2) M3) M4)

Sokoban

Abordagem usando programação dinâmica

Posição do jogador (x,y) | Posição das caixas (x,y)

(1,2) | (2,2) => [1][2][2][2] = 0 (movimentos)



Existem alguns problemas os quais não se conhece uma solução eficiente. Dizemos que estes problemas são “difíceis” ou intratáveis pois, para instâncias grandes, o tempo de processamento seria inviável.

Nestas situações é comum remover a exigência de procurar pela solução ótima e passamos a procurar por uma solução próxima da ótima (solução boa / razoável)

Algoritmos de Aproximação
(*Approximation Algorithms*)

Existem duas abordagens principais:

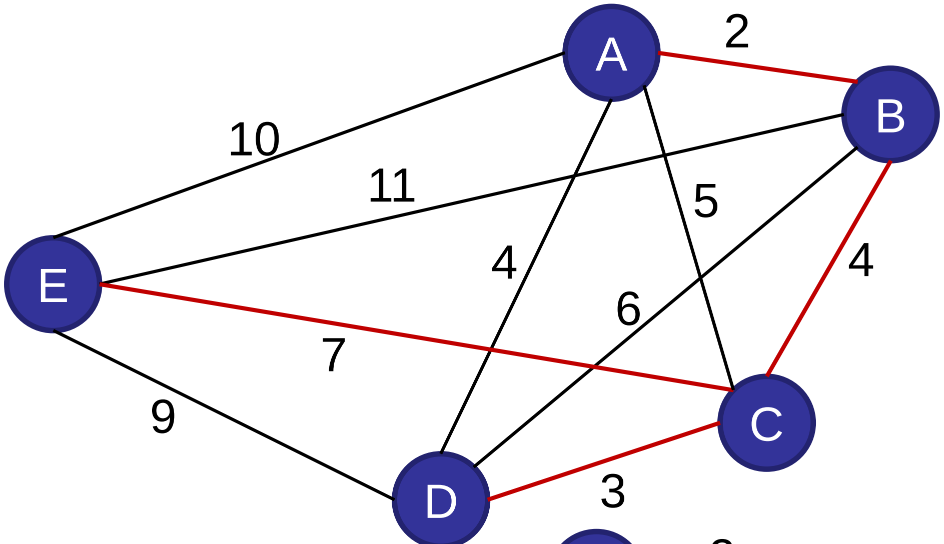
- **Heurística/Metaheurísticas**: algoritmo que buscam soluções próximas a solução ótima. Utilizam alguma informação (ou intuição) sobre a instância do problema para resolvê-lo de forma eficiente. Podem ser genéricas (servem p/ vários problemas).

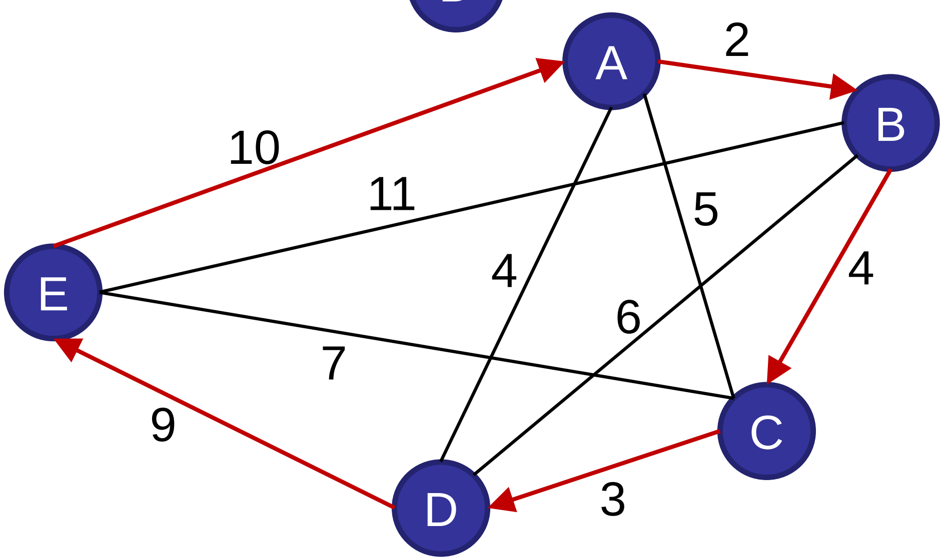
- **Algoritmo Aproximado**: algoritmo que resolve um problema de forma eficiente e ainda garante a qualidade da solução. É necessário provar que a solução está próxima da ótima. Geralmente únicos para cada problema.

Metaheurísticas

- **Algoritmos Bio-inspirados**: algoritmo baseados no comportamento de seres vivos:
  - **Algoritmos genéticos**: gera soluções, escolhe as melhores (seleção natural) e gera um novo ciclo, oferecendo espaço para mutações (alterações nos componentes das soluções).
  - **Colônia de formigas**: modela a geração de soluções baseado no comportamento das formigas com comunicação através do ambiente (como elas sempre encontram um doce?!) .
  - **Enxame de partículas**: se baseia em princípios da revoada de pássaros com influência local (histórico de cada partícula) e global/social (histórico geral)

Algoritmos Aproximados





Caixeiro Viajante

Dado o grafo $G = (V, A)$ e o custo c:

1. Selecione um vértice $r \in V$ para ser o vértice *raiz*.
2. Obtenha a ávore geradora mínima a partir de r.
3. Faça H ser a lista de vertices ordenados de acordo com a primeira visita, considerando um percurso em pré-ordem, retornando a r.

Se a função custo satisfaz a desigualdade de triângulos: $c(u, w) < c(u, v) + c(v, w)$

$$c(H) < 2c(\acute{o}timo)$$

Exercícios

- O pequeno Turing exagerou na quantidade de disciplinas neste semestre. Para passar nas provas finais ele precisa estudar uma quantidade de tempo para cada disciplina. Se ele estudar o tempo necessário com certeza será aprovado, caso contrário, certamente reprovará. Cada disciplina tem a sua importância subjetiva (is) para o menino, sendo que ele quer maximizar este índice de importância uma vez que sabe que não tem tempo viável para estudar para todas as disciplinas. Dada a lista de disciplinas e sua importância subjetiva, descubra qual o melhor cronograma de estudo para Turing a fim de maximizar a importância subjetiva (is) na aprovação (mostre a tabela utilizada na solução).

Exercícios

- Tempo disponível: 16 horas
- Disciplinas:
  - ALP (3h – 5is);
  - CAL (6h – 6is);
  - REC (9h – 5is);
  - TEC (8h – 6is);
  - PAP (5h – 3is);
  - BAN (4h – 4is);
  - PPR (3h – 2is);
  - SOP (4h – 4is);

Exercícios

- Considere as strings *genoma* e *feature* que representam, respectivamente, uma parte do genoma de uma criatura e uma determinada característica de interesse. Deseja-se descobrir qual é a maior subsequência comum (sequencia de caracteres iguais, mas não necessariamente contíguas) entre essas duas cadeias de caracteres:

  genoma: ACCUGTATAUCGUCACTU

  feature: GCAUTTC