Listas Encadeadas - Aplicação: Ordenação Topológica

Algoritmos e Programação 2

Prof. Dr. Anderson Bessa da Costa
Universidade Federal de Mato Grosso do Sul

Ordenação Topológica

- Problema que pode ser caracterizado como uma aplicação de listas lineares
- Importante
  - Uso potencial todas as vezes em que o problema abordado envolve uma ordem parcial.

Figura: Ordem parcial e ordenação topológica

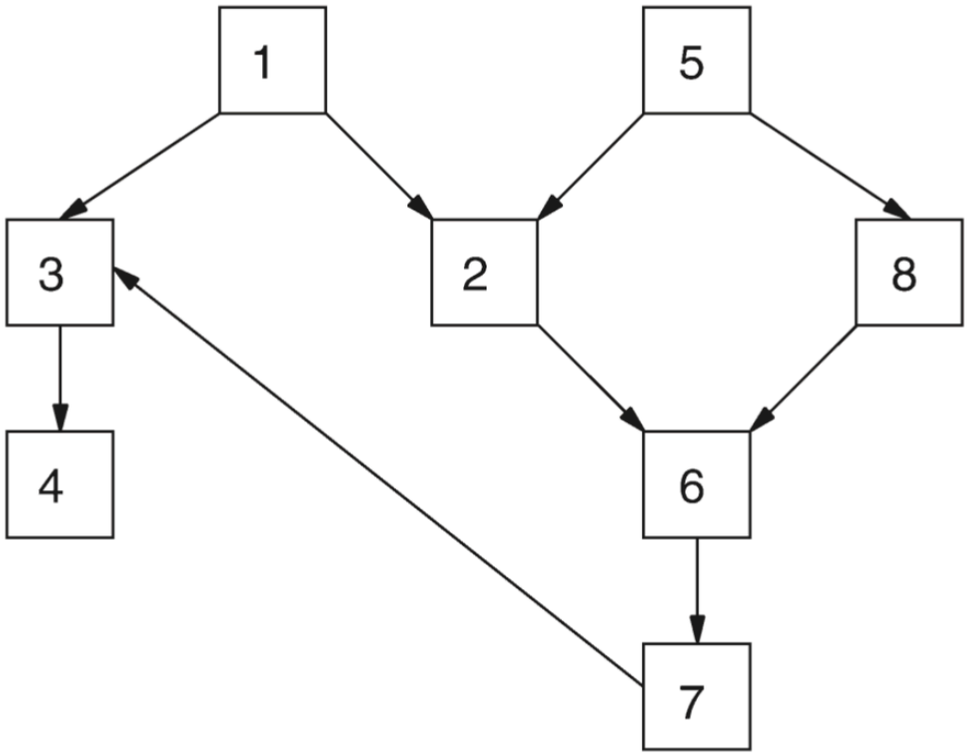

Figura 2.13 Representação de ordem parcial.

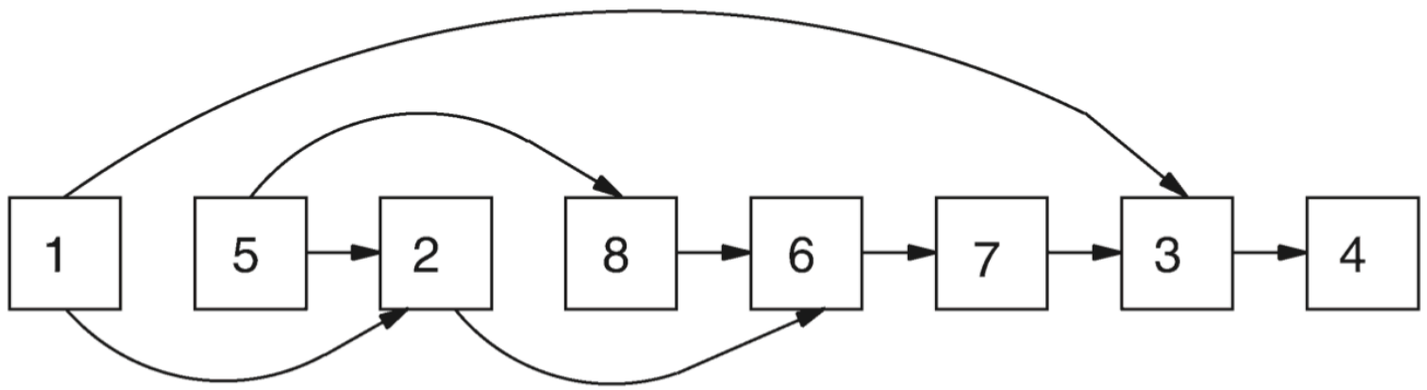

Figura 2.14 Ordenação topológica.

Ordem Parcial

- **Ordem parcial** de conjunto S é uma relação entre objetos de S, representada pelo símbolo "$\preceq$" (precede ou igual)
- Satisfaz as seguintes propriedades para quaisquer objetos x, y e z, não necessariamente distintos em S:
  - (i) se $x \preceq y$ e $y \preceq z$, então $x \preceq z$ (transitiva);
  - (ii) se $x \preceq y$ e $y \preceq x$, então $x = y$ (anti-simétrica);
  - (iii) $x \preceq x$ (reflexiva).

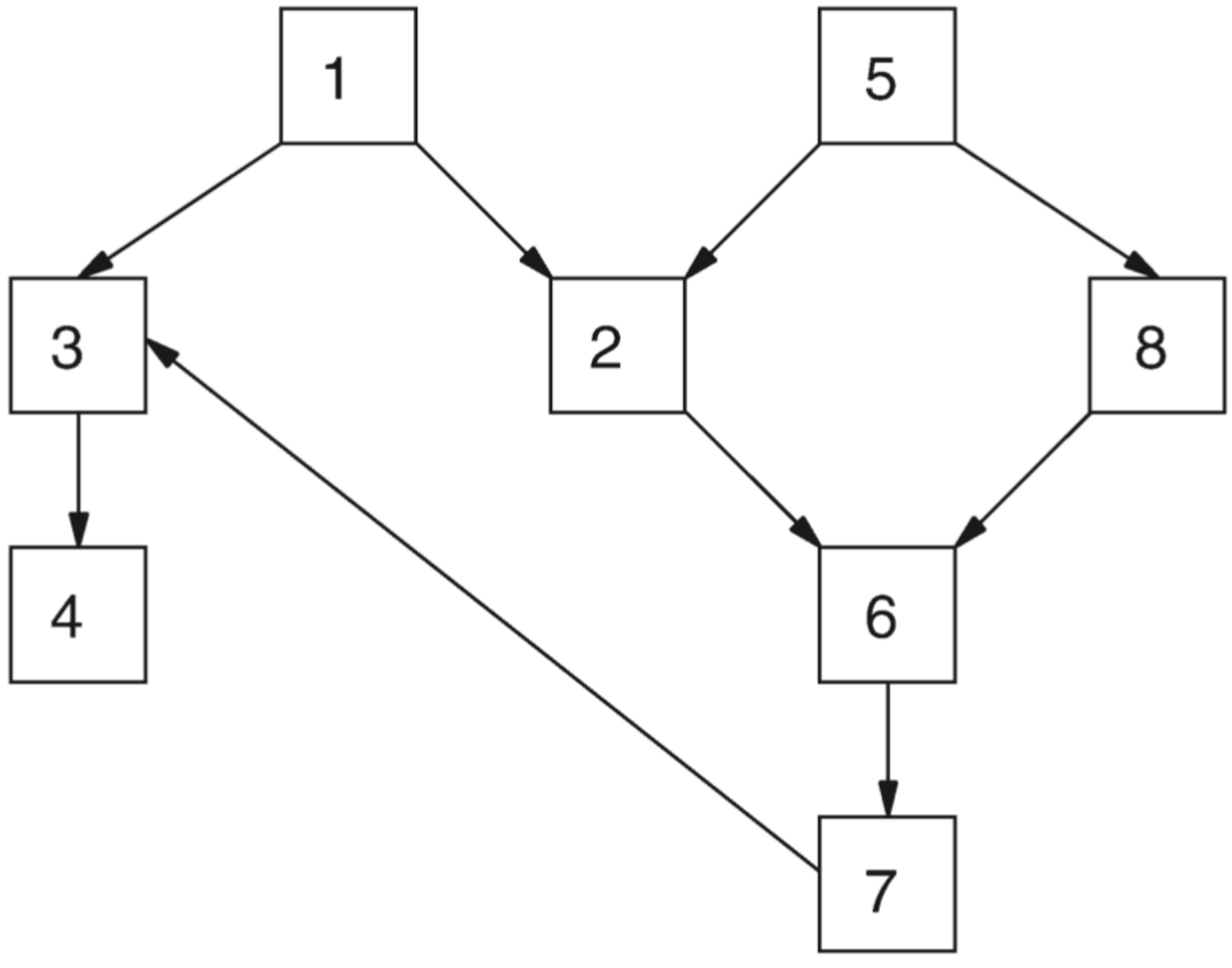

Figura 2.13 Representação de ordem parcial.

X precede y

- $x \preceq y$ pode ser lida "x precede ou igual y"
- Se $x \preceq y$ e $x \neq y$, escreve-se $x \prec y$ e diz-se "x precede y". Tem-se então:

  (i') se $x \prec y$ e $y \prec z$ então $x \prec z$ (transitiva);

  (ii') se $x \prec y$, então $y \not\prec x$ (assimétrica);

  (iii') $x \not\prec x$ (irreflexiva).

- Assume-se que S é um conjunto finito, uma vez que se deseja trabalhar no computador.

Exemplo Ordem Parcial

- Execução de um conjunto de tarefas necessárias, por exemplo, à montagem de um automóvel
- Cada caixa uma tarefa
- Cada tarefa numerada arbitrariamente
- Se existe existe a indicação de um caminho da caixa x para a caixa y, isso significa que a tarefa x deve ser executada antes da tarefa y

Ordenação Topológica

- **Ordenação topológica** trata de imergir a ordem parcial em uma ordem linear
  - Rearrumar os objetos numa sequência $a_1, a_2, \ldots, a_n$ tal que sempre que $a_j < a_k$, tem-se $j < k$

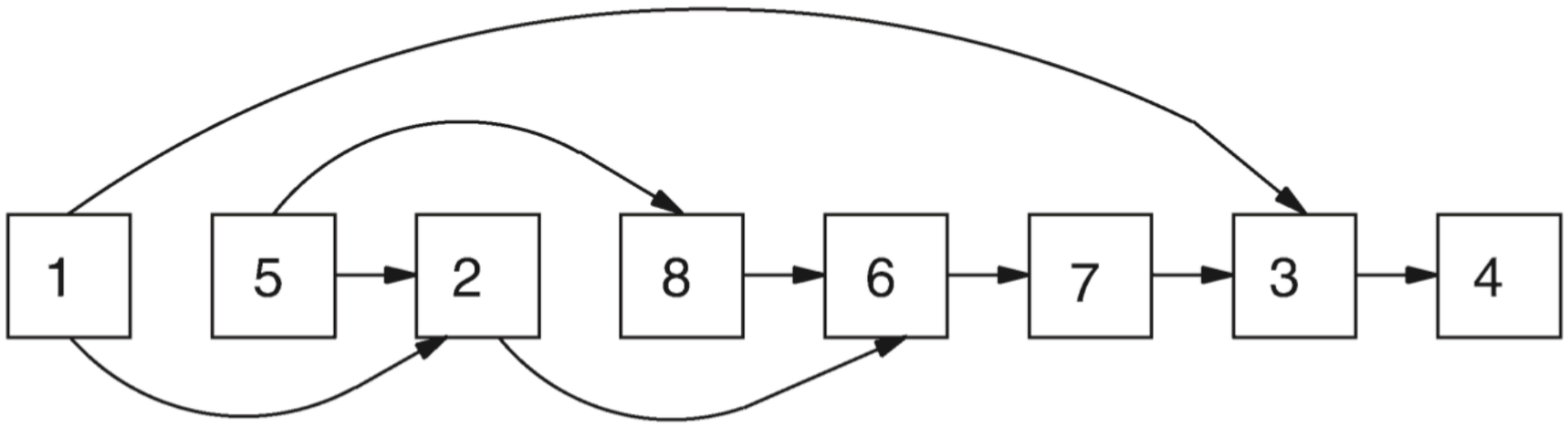

Figura 2.14 Ordenação topológica.

Ideia Ordenação Topológica

- Inicialmente, considera-se um objeto que não é precedido por nenhum outro na ordem parcial
- Esse objeto é o primeiro na saída, isto é, na ordem final
- Agora, remova o objeto do conjunto S
- O conjunto resultante obedece novamente a uma ordem parcial
- Repetir até que todo o conjunto esteja ordenado

Ideia Ordenação Topológica (cont.)

- Esse algoritmo só falharia se, em algum momento, a ordem parcial de um conjunto fosse tal que todos os elementos tivessem um predecessor
- Ora, isso é impossível, porque contraria as propriedades (i) e (ii)

Implementação

- Desempenho desse algoritmo depende de sua implementação
- Objetos a serem ordenados são numerados de 1 a n, em qualquer ordem, e alocados sequencialmente na memória
- Entrada do algoritmo são os pares (j, k), significando que o objeto j precede o objeto k
- Número de objetos n e o número de pares m também são fornecidos

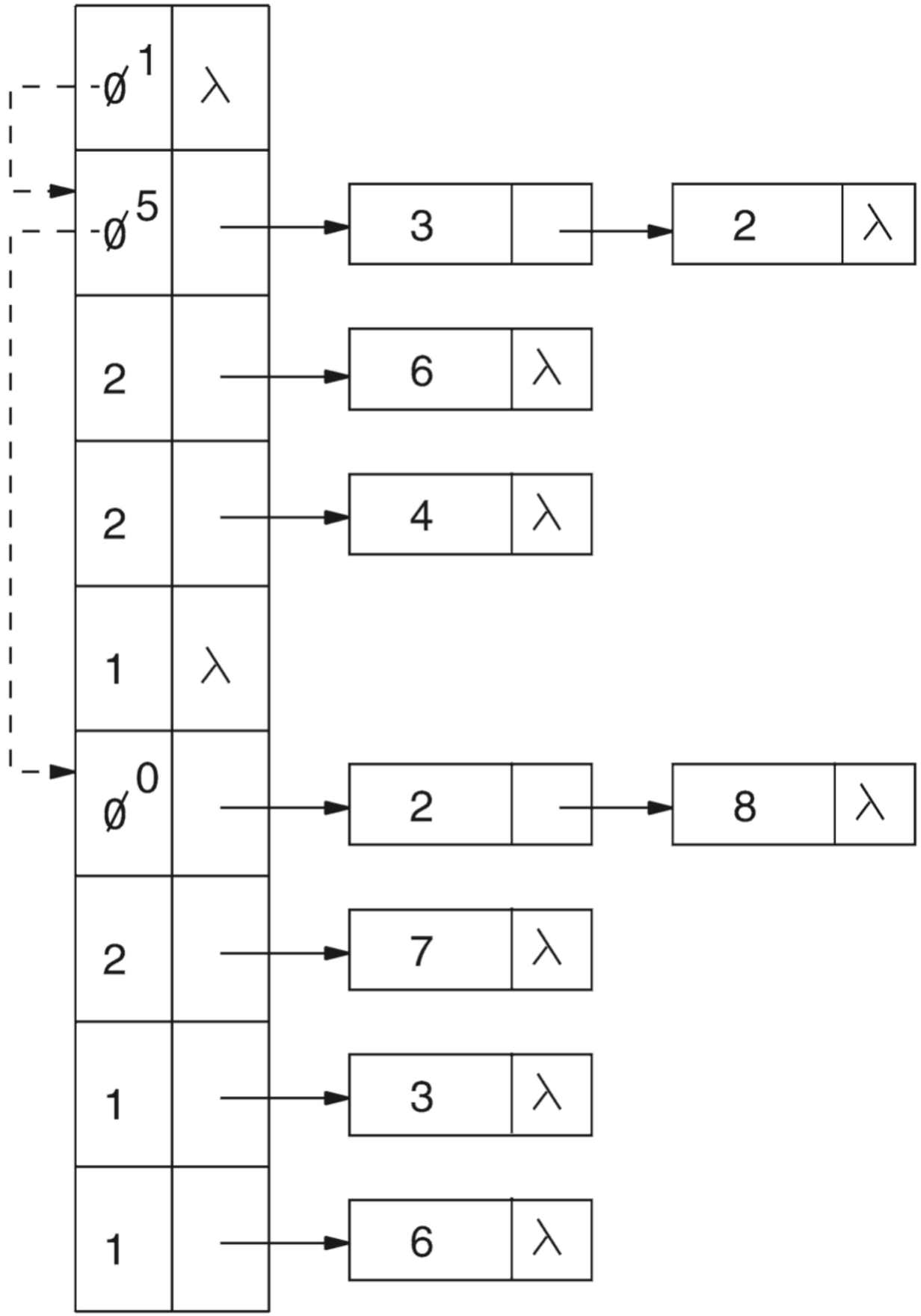

Figura 2.15 Armazenamento para a ordenação topológica.

```c
#include <stdio.h>
#include <stdlib.h>

/* Definicoes de tipos */
struct cel {
    int contador;
    struct cel *seg;
};

typedef struct cel celula;

/* Prototipos */
celula * inicialize(int n, int m);
void ordene_topologicamente(int n, int m);

int main() {
    int n, m;

    // leia n
    printf("n:\n");
    scanf("%d", &n);

    // leia m
    printf("m:\n");
    scanf("%d", &m);

    ordene_topologicamente(n, m);

    return 0;
}
```

```c
celula * inicialize(int n, int m) {
    int i;
    celula *cb;

    // aloque um vetor de n celulas cabeca
    cb = (celula *) malloc (sizeof(celula) * n + 1);

    // inicialize as n listas
    for (i = 0; i <= n; i++) {
        cb[i].contador = 0;
        cb[i].seg = NULL;
    }

    for(i = 1; i < m; i++) {
        celula *p;
        int j, k;

        // leia o par (j, k) - j precede k
        scanf("%d%d", &j, &k);

        // crie uma celula e inicialize com o sucessor k
        p = (celula *) malloc (sizeof(celula));
        p->contador = k;

        // adicione na lista j, que contem os sucessores de j
        p->seg = cb[j].seg;
        cb[j].seg = p;

        // incremente contador de k, que contabiliza o numero de vezes que k aparece
como sucessor
        cb[k].contador += 1;
    }
    return cb;
}
```

```c
void ordene_topologicamente(int n, int m) {
    celula *cb;
    int i, fim, objeto, indice;

    cb = inicialize(n, m);
    fim = 0; cb[0].contador = 0;

    // busque objetos sem predecessores
    for (i = 1; i <= n; i++) {
        if (cb[i].contador == 0) {
            cb[fim].contador = i;
            fim = i;
        }
    }

    objeto = cb[0].contador;
    while (objeto != 0) {
        celula *p;

        printf("%d -> ", objeto);  // saida objeto
        p = cb[objeto].seg;  // p recebe a lista de objeto

        while (p != NULL) {
            // armazene em indice o contador da celula (sucessor)
            indice = p->contador;

            // decremente de 1 a quantidade de vezes que indice atua como sucessor
            cb[indice].contador = cb[indice].contador - 1;

            // se agora indice atua 0 vezes como sucessor
            if (cb[indice].contador == 0) {
                cb[fim].contador = indice;
                fim = indice;
            }
            p = p->seg;
        }
        // objeto recebe o proximo objeto do encadeamento
        objeto = cb[objeto].contador;
    }
    printf("FIM \n");
}
```

entrada.txt

```
8
9
1 3
1 2
2 6
3 4
5 2
5 8
6 7
7 3
8 6
```

Terminal

```
% g++ ordenacao_topologica.cpp
% ./a.out < entrada.txt
n:
m:
1 -> 5 -> 8 -> 2 -> 6 -> 7 -> 3 -> 4 -> FIM
```

Referências

- SZWARCFITER, Jayme Luiz; MARKENZON, Lilian. Estruturas de Dados e seus Algoritmos. Edição: 3a. Editora: LTC. 2010