Aplicações de listas
Outras estruturas

SCC-202 – Algoritmos e Estruturas de Dados I

Lucas Antiqueira

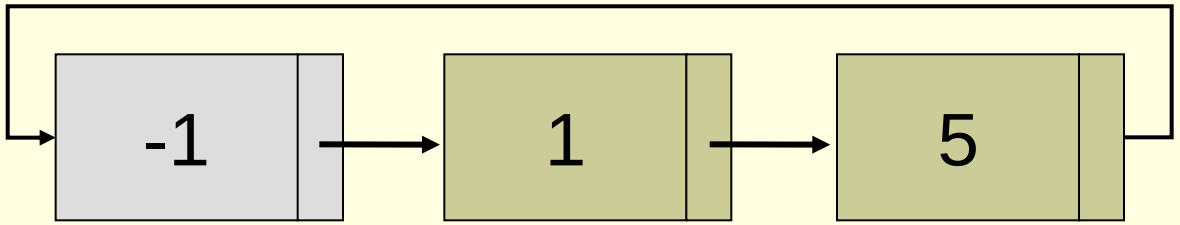
Grandes números

Grandes números

- Problema: lidar com números muito grandes
  - Em C, inteiros (mesmo long int) são limitados
  - Como somar números inteiros maiores do que o tamanho do tipo permite?
    - Listas!

Grandes números

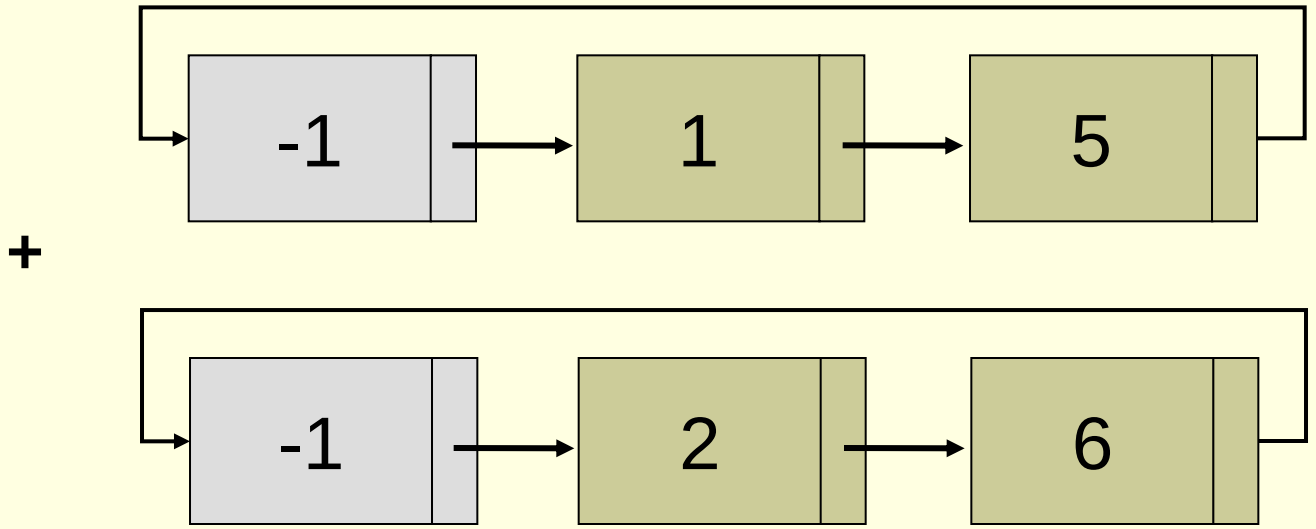
- Representando números como listas
  - 15



Exemplo circular com nó de cabeçalho (sentinela)

Grandes números

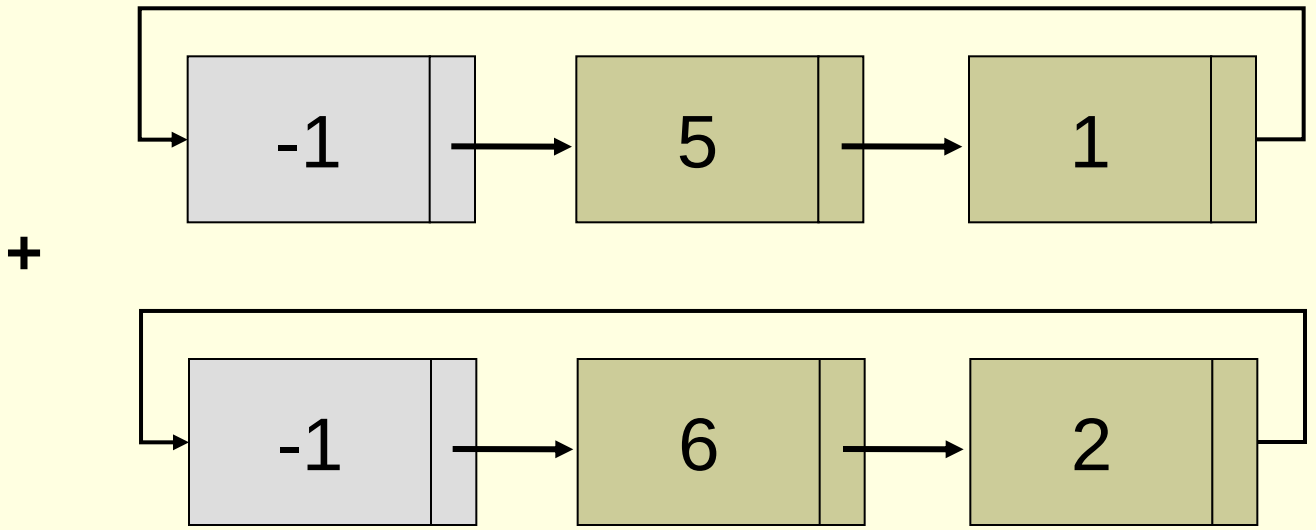
- Soma de dois números
  - Bloco somados dois a dois, da direita para a esquerda
    - Exemplo: 15+26



Grandes números

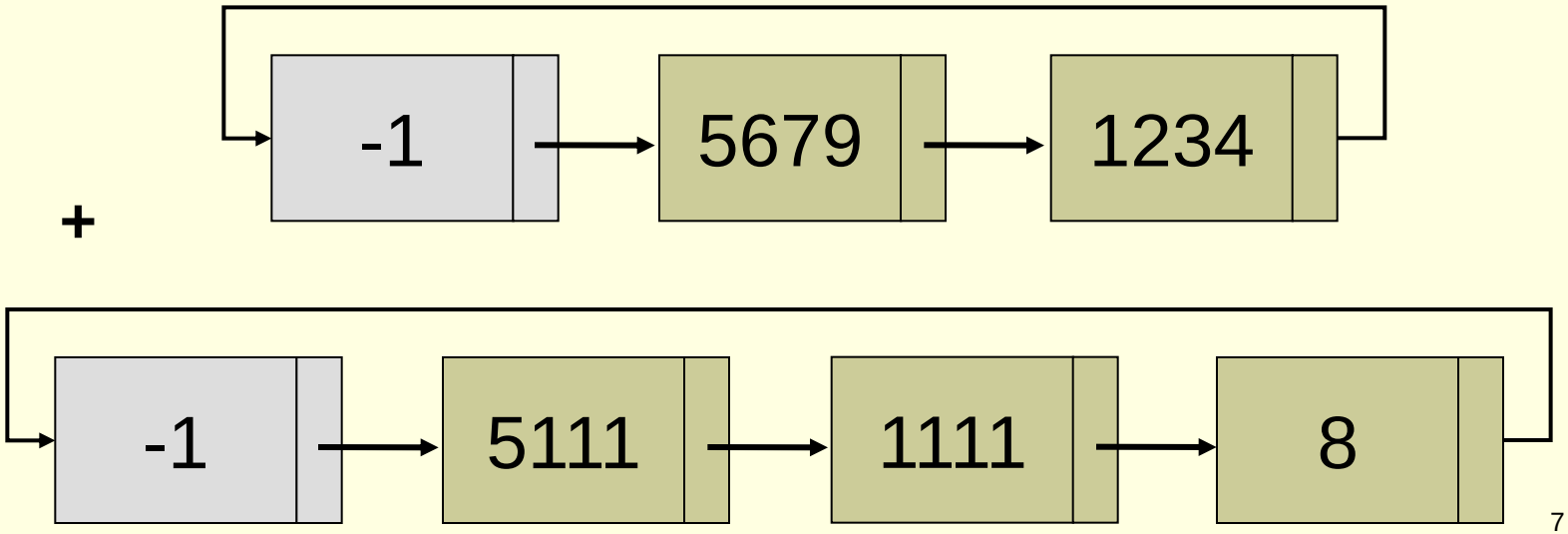
- Para facilitar nossa vida, números podem ser representados ao contrário

  - Exemplo: 15+26



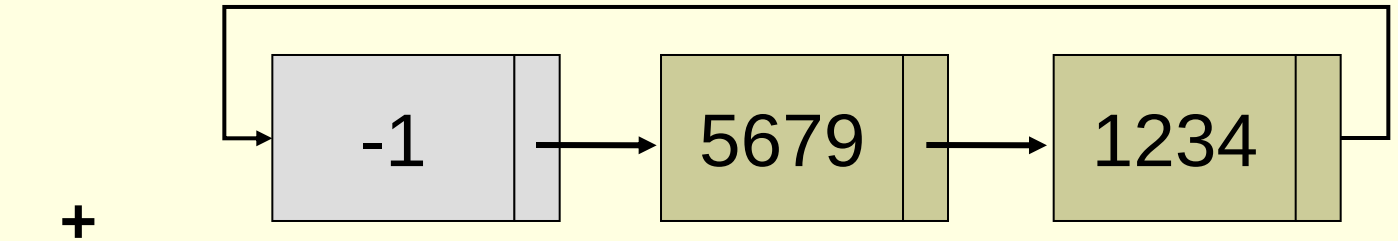
Grandes números

- Como os números tratados por esse mecanismo são muito grandes, pode-se aproveitar melhor o tipo inteiro: uso otimizado de memória
  - Exemplo: 12345679 + 811115111

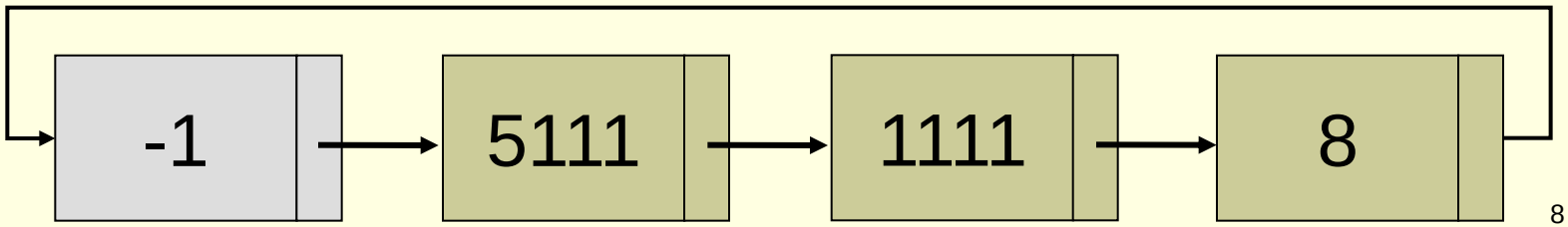


Grandes números

- Como recuperar o número somado para colocar na nova lista?
- Como recuperar o “sobe 1”?

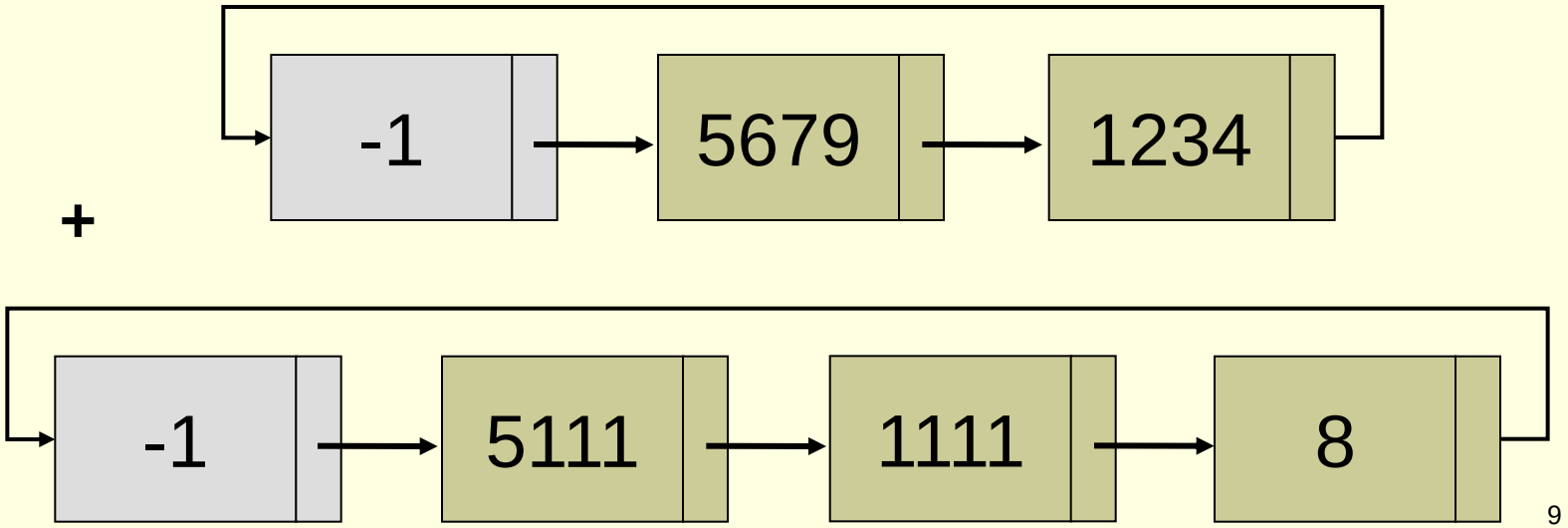


+

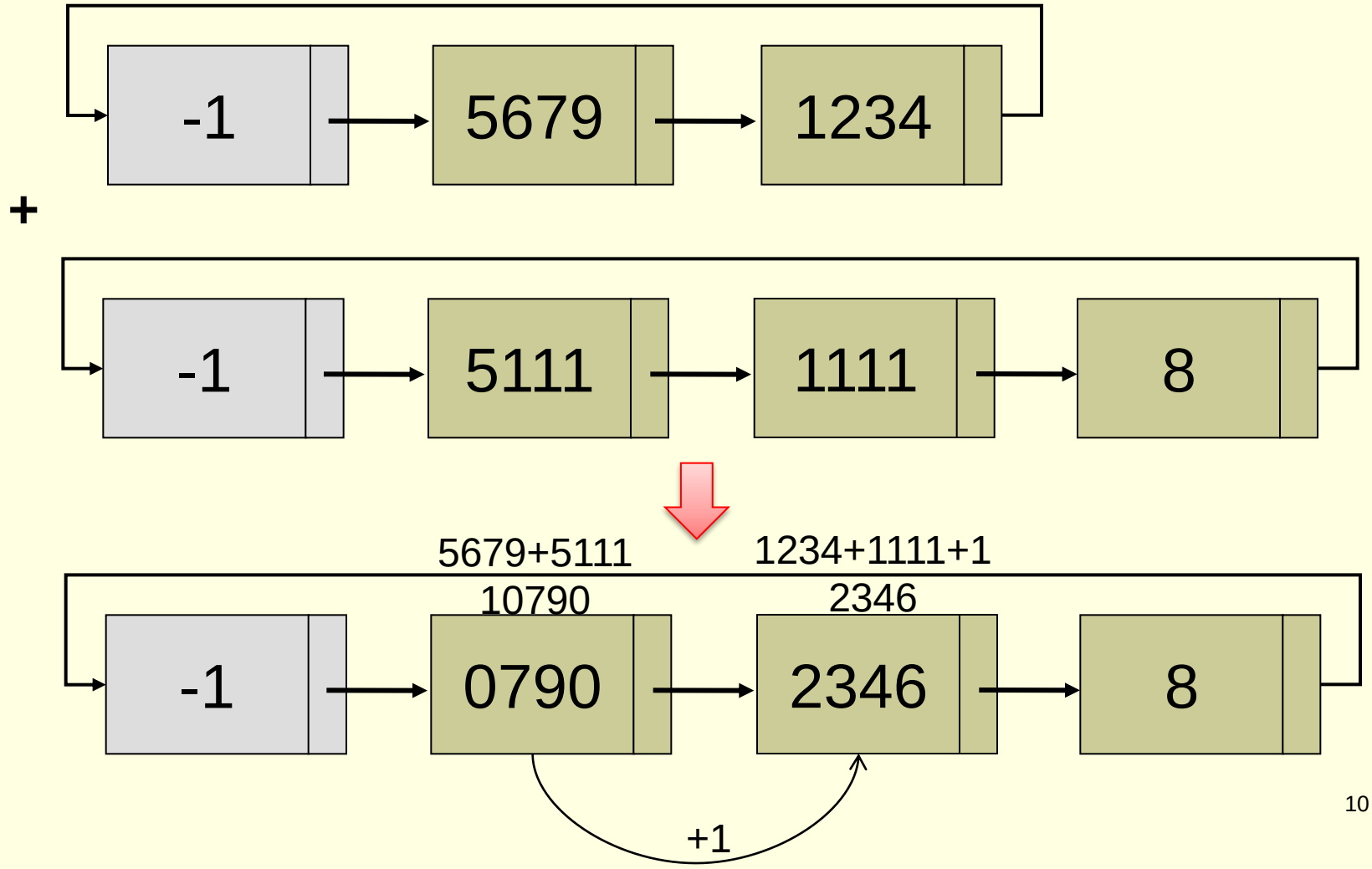


Grandes números

- Como recuperar o número somado para colocar na nova lista?
  - Soma % 10.000 (por que 4 zeros?)
- Como recuperar o “sobe 1”?
  - Soma / 10.000 (divisão inteira)



Grandes números



Grandes números

- Exercício para casa
  - Implemente em C uma sub-rotina para somar dois grandes números utilizando uma lista circular com nó de cabeçalho
    - As duas listas a serem somadas devem ser passadas por parâmetros, sendo que o ponteiro para a nova lista contendo a soma deve ser retornado em um outro ponteiro (por parâmetro também).

Matrizes esparsas

Matrizes

- Matriz é um arranjo (tabela) retangular de números dispostos em linhas e colunas

$$A_{3x4} = \begin{bmatrix} 1 & 0 & 4 & -3 \\ 2 & 5 & 3 & 4 \\ 9 & 8 & -2 & 1 \end{bmatrix} \qquad B_{3x3} = \begin{bmatrix} 3 & 7 & 4 \\ 1 & 0 & 6 \\ 9 & 2 & 8 \end{bmatrix}$$

nº de elementos = nº de linhas * nº de colunas

Matriz = Arranjo bidimensional

Matrizes especiais

$$A_{3x3} = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 3 & 0 \\ 4 & 5 & 6 \end{bmatrix}$$

Triangular inferior

$$B_{4x4} = \begin{bmatrix} 1 & 2 & 0 & 0 \\ 2 & 4 & 5 & 0 \\ 0 & 6 & 7 & 8 \\ 0 & 0 & 9 & 10 \end{bmatrix}$$

Tri-diagonal

$$C_{7x9} = \begin{bmatrix} 1 & 0 & 0 & 3 & 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 & 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 4 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 8 & 0 & 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Matriz esparsa: excessivo nº de elementos nulos (0)

Matrizes esparsas

$$C = \begin{bmatrix} 1 & 0 & 0 & 3 & 0 & 0 & 0 & \cdots & 0 \\ 0 & 2 & 0 & 0 & 0 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 0 & -1 & 4 & 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & & \vdots \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & \cdots & 0 \end{bmatrix}$$

$9.000 x 9.000$

9.000 x 9.000 = 81.000.000 elementos

Considerando int de 4 bytes: 4 * 81.000.000 bytes ~ 310 MB

Matriz esparsa com **15.000** elementos **não nulos**, os quais ocupam aprox. 60 KB

0,02% do total

Matrizes esparsas

- Uso da matriz tradicional
  - Vantagem
    - Ao se representar dessa forma, preserva-se o acesso direto a cada elemento da matriz
      - Algoritmos simples
  - Desvantagem
    - Muito espaço para armazenar zeros

Matrizes esparsas

- Necessidade
  - Método alternativo para representação de matrizes esparsas
- Solução
  - Estrutura de lista encadeada contendo somente os elementos não nulos

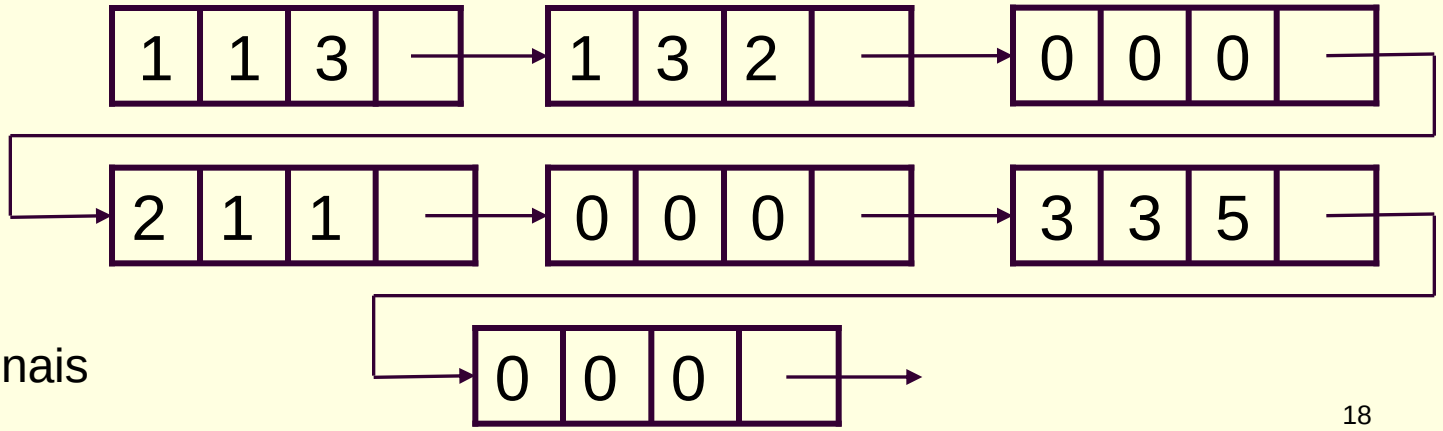
Matrizes esparsas - solução 1

- Listas simples encadeadas

$$A_{3x3} = \begin{bmatrix} 3 & 0 & 2 \\ 1 & 0 & 0 \\ 0 & 0 & 5 \end{bmatrix}$$

linha	coluna	valor	prox

Estrutura de um nó:
- linha, coluna: posição
- valor: ≠ zero
- prox: próximo nó



Nós zerados opcionais para auxiliar na divisão de linhas

Matrizes esparsas - solução 1

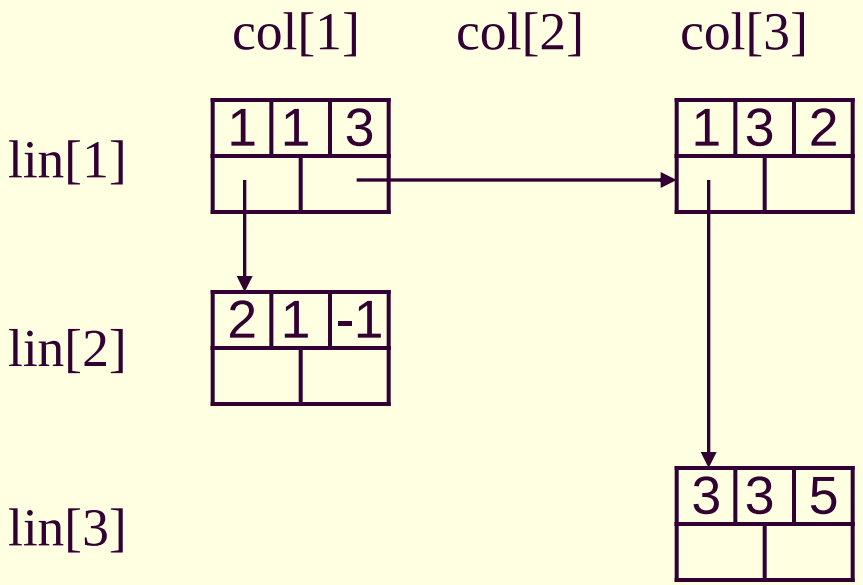
- Desvantagens
  - Perda da natureza bidimensional de matriz
  - Acesso ineficiente à linha
    - Para acessar o elemento na i-ésima linha, deve-se atravessar as i-1 linhas anteriores
  - Acesso ainda mais ineficiente à coluna
    - Para acessar os elementos na j-ésima coluna, tem que se passar por outras linhas antes

- Questão
  - Como organizar essa lista, preservando a natureza bidimensional de matriz?

Matrizes esparsas - solução 2

- Listas cruzadas
  - Para cada matriz, usam-se dois vetores com N ponteiros para as linhas e M ponteiros para as colunas

$$A_{3x3} = \begin{bmatrix} 3 & 0 & 2 \\ -1 & 0 & 0 \\ 0 & 0 & 5 \end{bmatrix}$$



Estrutura de um nó:

linha	coluna	valor
proxlin		proxcol

Matrizes esparsas - solução 2

- Listas cruzadas
  - Cada elemento não nulo é mantido simultaneamente em duas listas
    - Uma para sua linha
    - Uma para sua coluna

Matrizes esparsas

- Listas cruzadas vs. matriz tradicional
  - Em termos de espaço
    - Supor que inteiro e ponteiro para inteiro ocupam um bloco de memória
    - Listas cruzadas: tamanho do vetor de linhas (nl) + tamanho do vetor de colunas (nc) + n elementos não nulos * tamanho do nó (5)
      - nl+nc+n*5
    - Matriz tradicional bidimensional
      - nl*nc

Matrizes esparsas

- Listas cruzadas vs. matriz tradicional
  - Em termos de tempo
    - Operações mais lentas em listas cruzadas: acesso não é direto

Matrizes esparsas

- Listas cruzadas vs. matriz tradicional
  - Necessidade de avaliação tempo-espaço para cada aplicação
  - Em geral, usam-se listas cruzadas quando no máximo (aproximadamente) 1/5 dos elementos forem não nulos
    - De onde vem isso?

Matrizes esparsas

- Listas cruzadas vs. matriz tradicional
  - Necessidade de avaliação tempo-espaço para cada aplicação
  - Em geral, usam-se listas cruzadas quando:

$$nl+nc+5n < nl*nc$$
$$5n < nl*nc - nl - nc$$
$$n < 1/5*(nl*nc - nl - nc)$$
$$n < 1/5*[(nl-1)*(nc-1) - 1]$$

Nº de elementos não nulos

Matrizes esparsas - operações

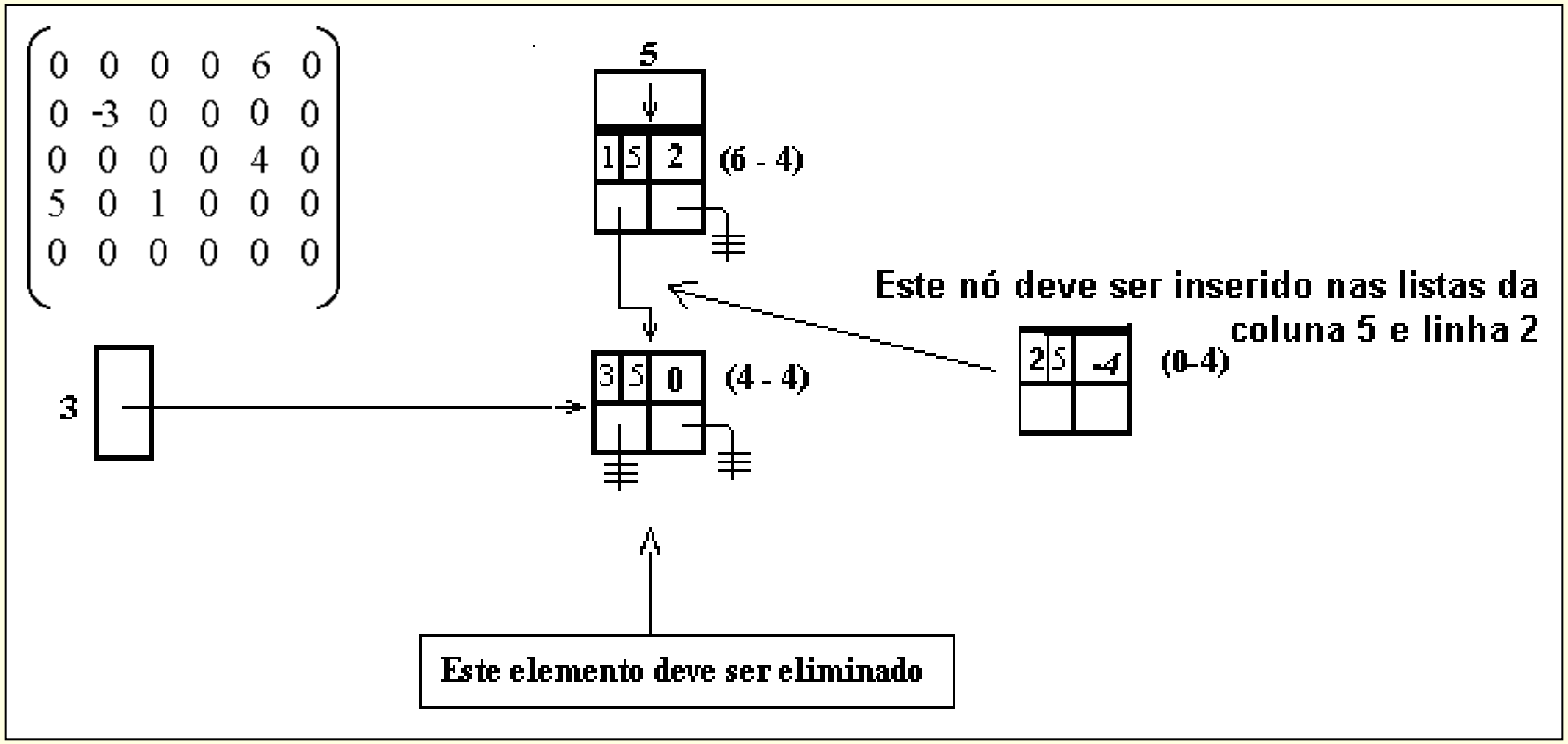
- Em geral
  - Multiplicar uma dada linha ou coluna por uma constante
  - Somar uma constante a todos os elementos de uma linha ou coluna
  - Somar duas matrizes esparsas de igual dimensão
  - Multiplicar matrizes esparsas
  - Transpor matrizes esparsas
  - Inserir, remover ou alterar elementos
  - Etc.

Matrizes esparsas - operações

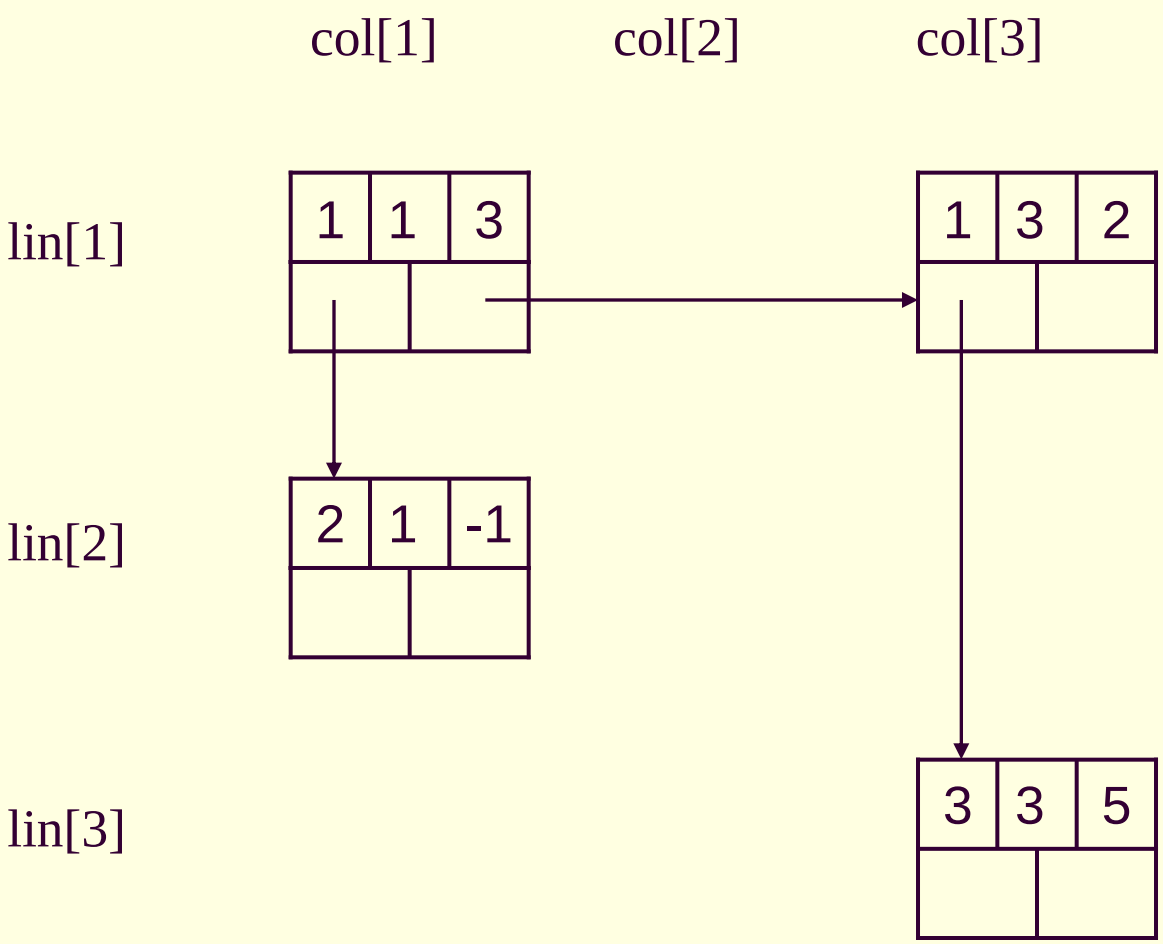
- Após a realização de alguma operação sobre a matriz
  - Quando um elemento da matriz se torna nulo
    - Remoção do elemento
  - Quando algum elemento se torna não nulo
    - Inserção do elemento

Matrizes esparsas - operações

- Por exemplo, ao se somar -4 a coluna 5 do exemplo



Exercício: somar -5 a coluna 3



Estrutura em C

```
#define n 1000      //número de linhas
#define m 1000      //número de colunas

typedef struct no {
  int lin, col, val;
  struct no *proxlin, *proxcol;
} no;

typedef struct {
  no *L[n], *C[m];
} MatrizEsparsa;
```

Matrizes esparsas

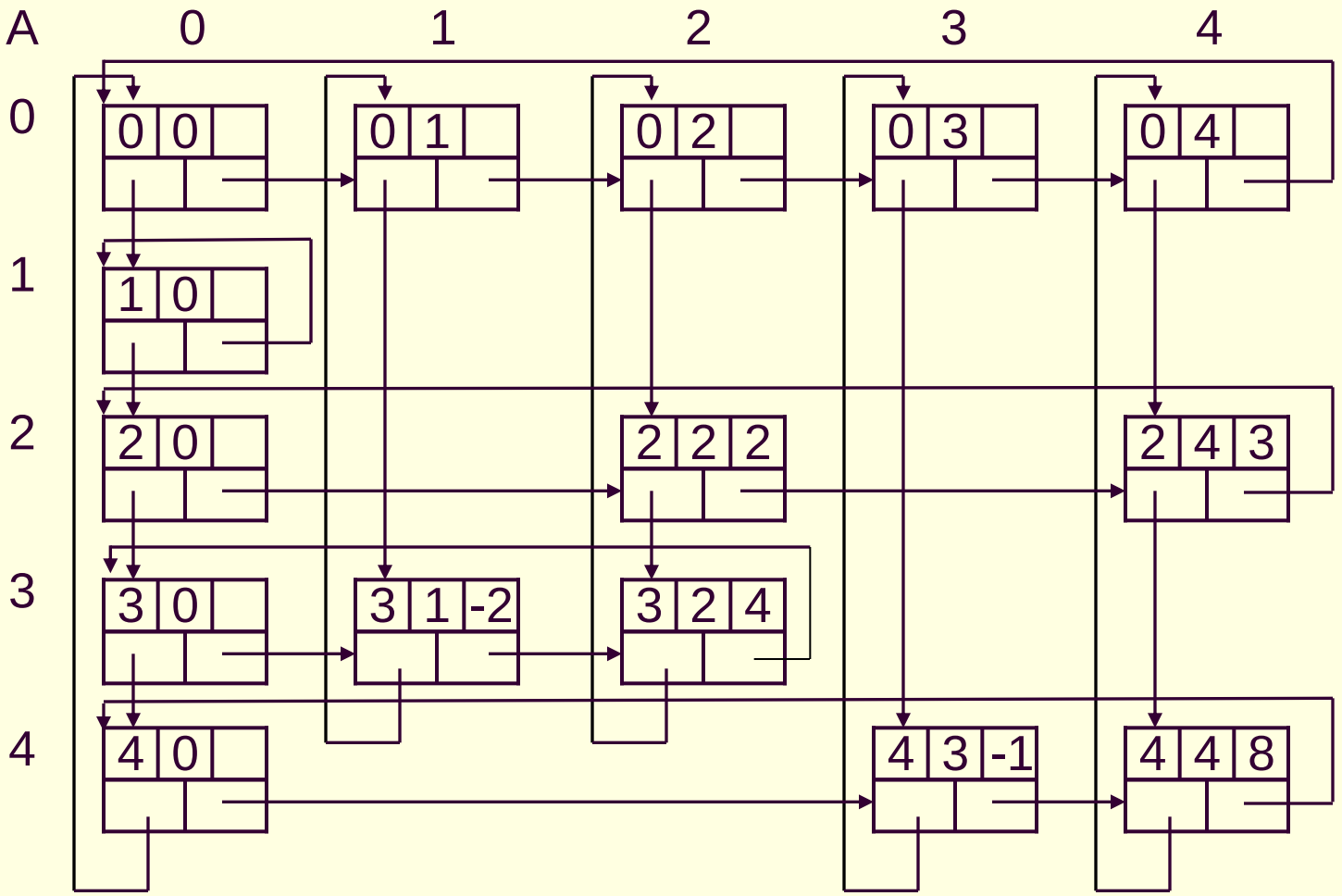
- Exercício para casa:
  - Implementar uma sub-rotina para somar um número K qualquer a uma coluna da matriz
    - Usando listas cruzadas

Matrizes esparsas - outra solução

- Listas circulares com nós de cabeçalho
  - Ao invés de vetores de ponteiros, linhas e colunas são listas circulares com nós de cabeçalho
    - Nós de cabeçalho: reconhecidos por um 0 no campo linha ou coluna
      - 1 único ponteiro para a matriz: navegação em qualquer sentido
  - Exemplo

$$A_{4x4} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 3 \\ -2 & 4 & 0 & 0 \\ 0 & 0 & -1 & 8 \end{bmatrix}$$

Matrizes esparsas - outra solução



Matrizes esparsas - outra solução

- Quais as desvantagens dessa representação?
  - Mais complexa de se manipular
- Quais as vantagens dessa representação?
  - A matriz pode crescer dinamicamente

Fila de prioridade

Fila de prioridade

- Pilha e fila
  - Elementos processados em função da seqüência na qual foram inseridos
- Fila de prioridade
  - Os elementos são processados de acordo com sua importância, não importando o momento em que entraram na fila
    - Por exemplo, atendimento de idosos e gestantes em bancos, importância de processos em sistemas operacionais, substituição de páginas menos utilizadas na memória

Fila de prioridade

- Tipos
  - Ascendente
    - Remove-se o item de menor valor
  - Descendente
    - Remove-se o item de maior valor

Fila de prioridade

- Estruturas
  - Sequencial: é necessário que se desloque elementos para inserção/remoção
  - Encadeada: é mais simples inserir/remover

Fila de prioridade

- Estruturas
  - Listas não ordenadas
    - Vantagens e desvantagens?
  - Listas ordenadas
    - Vantagens e desvantagens?

Fila de prioridade

- Estruturas
  - Listas não ordenadas
    - Inserção é simples, mas remoção exige busca
  - Listas ordenadas
    - Remoção é simples, mas inserção exige busca

Fila de prioridade

- É possível usar um heap?

Fila de prioridade

- É possível usar um heap?
  - Veremos mais adiante, quando estudarmos árvores

Fila de prioridade

- Exercício
  - Implementar o TAD fila de prioridade sobre uma lista ordenada dinâmica e encadeada
    - Declare a estrutura
    - Defina e implemente as sub-rotinas relevantes

Deque

Deque

- Deque = *Double-ended queue*
  - Fila de extremidade dupla
- Estrutura de dados relativamente comum, mas pouco conhecida por esse nome
- Enquanto pilha e fila exigem inserções e remoções em extremidades específicas da estrutura, deque permite inserções e remoções em quaisquer extremidades
  - Se insere em uma extremidade e remove dela: pilha
  - Se insere em uma extremidade e remove da outra: fila
  - Deque permite inserções e remoções dos dois tipos, “misturadas”

Deque

- Implementação do TAD Deque
  - Estrutura de dados tradicional
  - Funções específicas para inserção e remoção em ambas as extremidades

Créditos

- *Material gentilmente cedido pelo Prof. Thiago A. S. Pardo*