Filas

Estrutura de Dados — QXD0010

Prof. Atílio Gomes Luiz
gomes.atilio@ufc.br

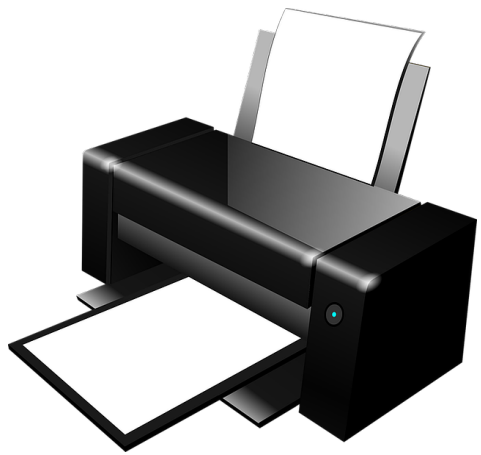
Universidade Federal do Ceará
Campus Quixadá

2º semestre/2022

Introdução

Filas

- Uma impressora é compartilhada em um laboratório
- Alunos enviam documentos quase ao mesmo tempo



Como gerenciar a lista de tarefas de impressão?

Filas

- São `listas lineares` que adotam a política FIFO para a manipulação de elementos.
- FIFO (*first-in first-out*): o primeiro que entra é o primeiro que sai. Remove primeiro objetos inseridos há mais tempo

Filas

- São `listas lineares` que adotam a política FIFO para a manipulação de elementos.
- FIFO (*first-in first-out*): o primeiro que entra é o primeiro que sai. Remove primeiro objetos inseridos há mais tempo

Operações básicas:

- Enfileira (*push*): adiciona item no "fim"

Filas

- São `listas lineares` que adotam a política FIFO para a manipulação de elementos.
- FIFO (*first-in first-out*): o primeiro que entra é o primeiro que sai. Remove primeiro objetos inseridos há mais tempo

Operações básicas:

- Enfileira (*push*): adiciona item no "fim"
- Desenfileira (*pop*): remove item do "início"

Filas

- São `listas lineares` que adotam a política FIFO para a manipulação de elementos.
- FIFO (*first-in first-out*): o primeiro que entra é o primeiro que sai. Remove primeiro objetos inseridos há mais tempo

Operações básicas:

- Enfileira (*push*): adiciona item no "fim"
- Desenfileira (*pop*): remove item do "início"
- A consulta na fila é feita desenfileirando elemento a elemento até encontrar o elemento desejado ou chegar ao final da fila.

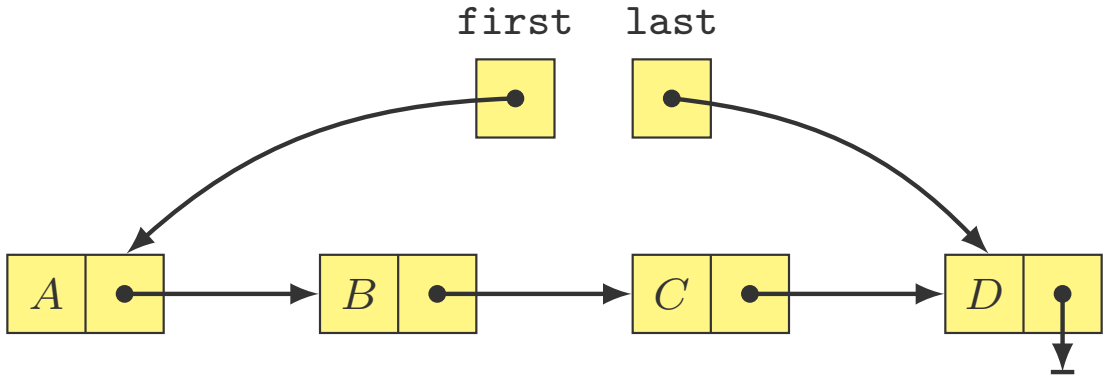
Filas — Operações

- Construir uma fila vazia
- Testar se a fila está vazia
- Retornar o número de elementos na fila
- Acessar o primeiro da fila
- Acessar o último elemento da fila
- Inserir um elemento
- Remover o próximo elemento

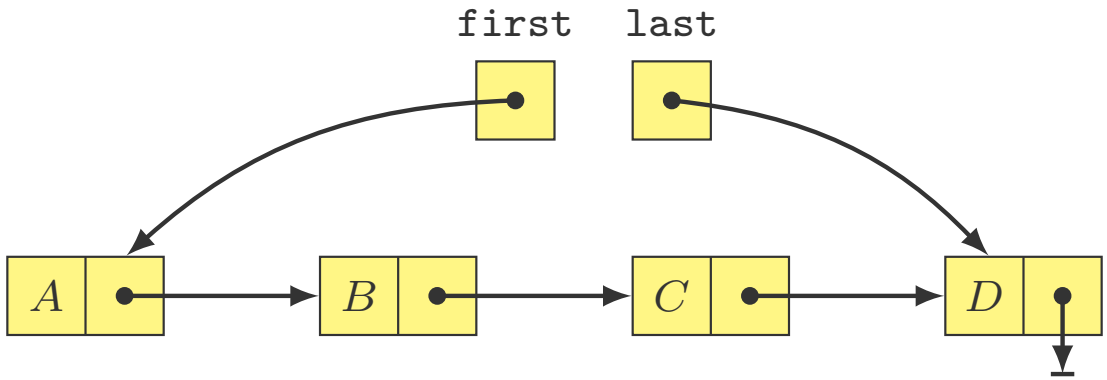
UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

Implementação

Implementação de uma Fila

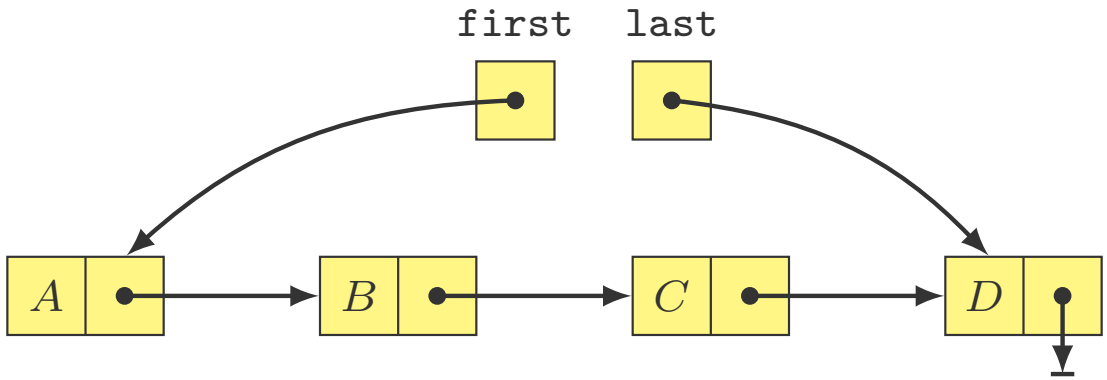


Implementação de uma Fila



- Com relação à alocação de memória, o modo mais natural de implementar uma fila é usando alocação dinâmica.

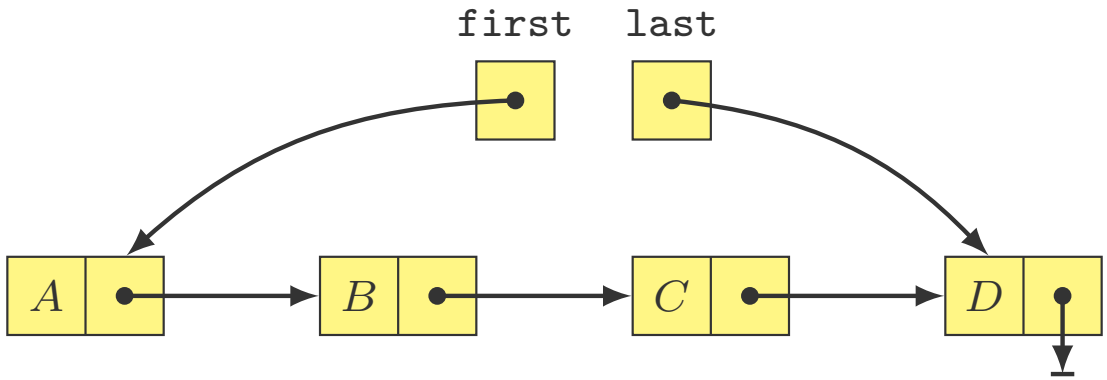
Implementação de uma Fila



- Com relação à alocação de memória, o modo mais natural de implementar uma fila é usando alocação dinâmica.

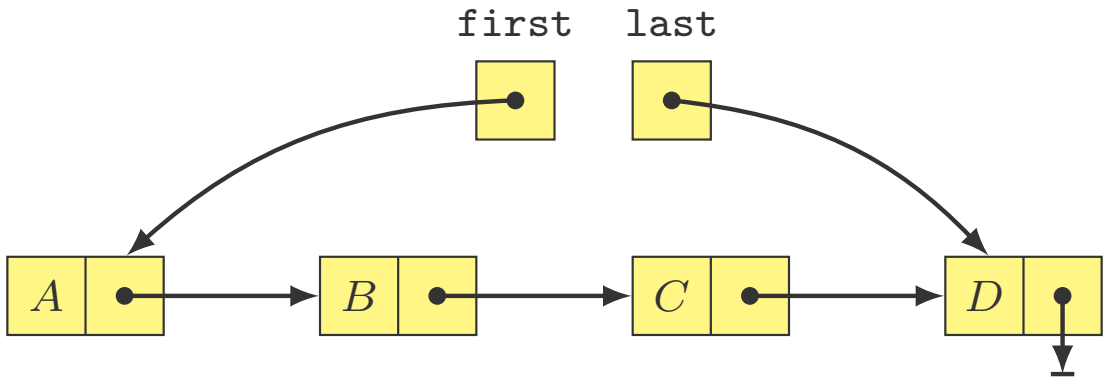
- Vamos implementar uma fila usando uma lista simplesmente encadeada sem nó cabeça com um ponteiro para o início e outro para o fim da lista.

Implementação de uma Fila



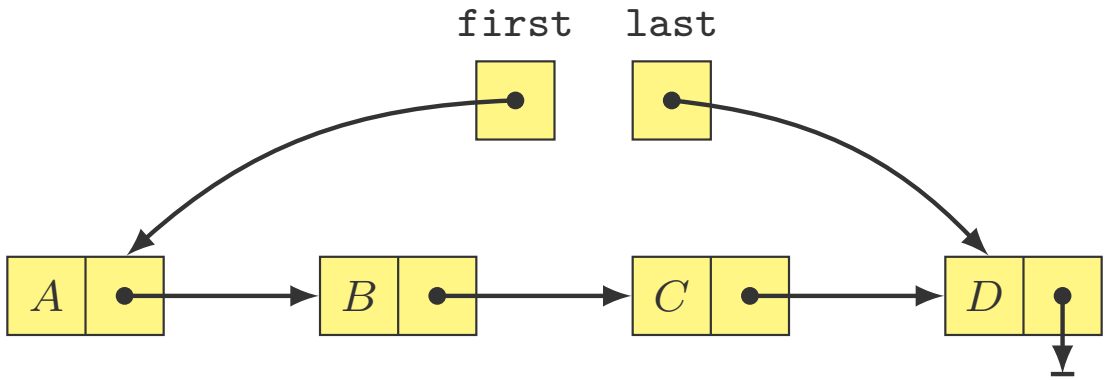
- Com relação à alocação de memória, o modo mais natural de implementar uma fila é usando alocação dinâmica.
- Vamos implementar uma fila usando uma lista simplesmente encadeada sem nó cabeça com um ponteiro para o início e outro para o fim da lista.
- Outras variações de lista podem ser usadas:

Implementação de uma Fila



- Com relação à alocação de memória, o modo mais natural de implementar uma fila é usando alocação dinâmica.
- Vamos implementar uma fila usando uma lista simplesmente encadeada sem nó cabeça com um ponteiro para o início e outro para o fim da lista.
- Outras variações de lista podem ser usadas:
  - Lista circular simplesmente encadeada;

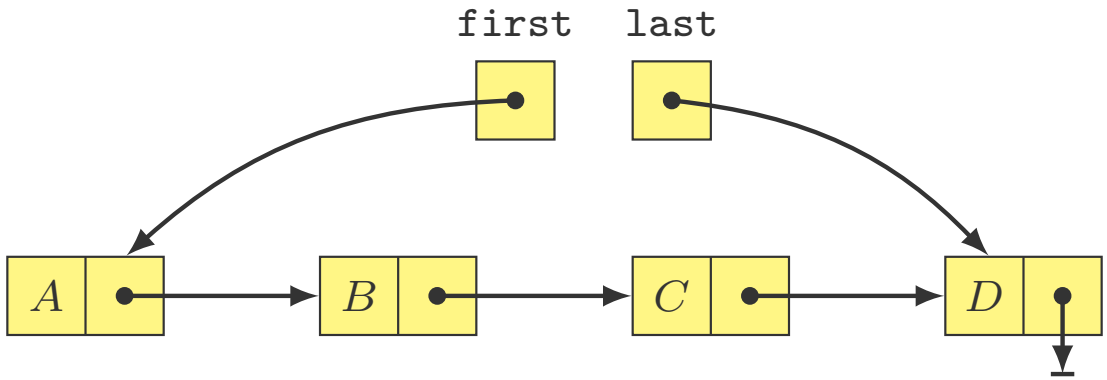
Implementação de uma Fila



- Com relação à alocação de memória, o modo mais natural de implementar uma fila é usando alocação dinâmica.
- Vamos implementar uma fila usando uma lista simplesmente encadeada sem nó cabeça com um ponteiro para o início e outro para o fim da lista.
- Outras variações de lista podem ser usadas:
  - Lista circular simplesmente encadeada;
  - Lista duplamente encadeada;

Implementação de uma Fila

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ



- Com relação à alocação de memória, o modo mais natural de implementar uma fila é usando alocação dinâmica.
- Vamos implementar uma fila usando uma lista simplesmente encadeada sem nó cabeça com um ponteiro para o início e outro para o fim da lista.
- Outras variações de lista podem ser usadas:
  - Lista circular simplesmente encadeada;
  - Lista duplamente encadeada;
  - Lista circular duplamente encadeada, etc.

Implementação de uma Fila

- Nossa fila armazenará qualquer tipo de dado válido. Para isso, usaremos templates!

Implementação de uma Fila

- Nossa fila armazenará qualquer tipo de dado válido.
  Para isso, usaremos templates!

- A nível de implementação, cada nó da lista simplesmente encadeada será representado como uma estrutura (struct) que possui apenas dois campos:

Implementação de uma Fila

- Nossa fila armazenará qualquer tipo de dado válido.
  Para isso, usaremos templates!

- A nível de implementação, cada nó da lista simplesmente encadeada será representado como uma estrutura (struct) que possui apenas dois campos:
  - `value`: guarda o valor.

Implementação de uma Fila

- Nossa fila armazenará qualquer tipo de dado válido.
  Para isso, usaremos templates!

- A nível de implementação, cada nó da lista simplesmente encadeada será representado como uma estrutura (struct) que possui apenas dois campos:
  - `value`: guarda o valor.
  - `next`: ponteiro que aponta para o nó seguinte na lista.

Arquivo Node.h


```
1  #ifndef NODE_H
2  #define NODE_H
3
4  template<typename T>
5  struct Node {
6      T value;     // valor a ser enfileirado
7      Node *next; // ponteiro para o proximo da fila
8
9      // Construtor
10     Node(const T& val, Node* nxt) {
11         value = val;
12         next = nxt;
13     }
14
15     // Destrutor
16     ~Node() {
17         delete next; // libera o proximo node
18     }
19 };
20
21 #endif
```

Queue.h — Tipo Abstrato de Dado Fila

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

```
1  template<typename T>
2  class Queue {
3  private:
4      Node<T> *m_first {nullptr}; // ponteiro para primeiro
5      Node<T> *m_last {nullptr}; // ponteiro para ultimo
6      int m_size {0}; // numero de elementos na fila
7
8  public:
9      Queue() = default;
10     ~Queue();
11     bool empty() const;
12     int size() const;
13     T& front();
14     const T& front() const;
15     T& back();
16     const T& back() const;
17     void push(const T& val);
18     void pop();
19     Queue(const Queue& q) = delete;
20     Queue& operator=(const Queue& q) = delete;
21 };
```

Queue.h — Implementação da Fila

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

```
1  template<typename T>
2  Queue<T>::~Queue() {
3      delete m_first;
4  }
5
6  template<typename T>
7  bool Queue<T>::empty() const {
8      return m_size == 0;
9  }
10
11 template<typename T>
12 int Queue<T>::size() const {
13     return m_size;
14 }
```

Queue.h — Implementação da Fila


```
1  template<typename T>
2  T& Queue<T>::front() {
3      if(m_size == 0) {
4          throw std::runtime_error("empty queue");
5      }
6      return m_first->value;
7  }
8
9  template<typename T>
10 const T& Queue<T>::front() const {
11     if(m_size == 0) {
12         throw std::runtime_error("empty queue");
13     }
14     return m_first->value;
15 }
```

Queue.h — Implementação da Fila


```
1  template<typename T>
2  T& Queue<T>::back() {
3      if(m_size == 0) {
4          throw std::runtime_error("empty queue");
5      }
6      return m_last->value;
7  }
8
9  template<typename T>
10 const T& Queue<T>::back() const {
11     if(m_size == 0) {
12         throw std::runtime_error("empty queue");
13     }
14     return m_last->value;
15 }
```

Queue.h — Implementação da Fila

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

```
1  template<typename T>
2  void Queue<T>::push(const T& val) {
3      Node<T> *aux = new Node<T>(val, nullptr);
4      if(m_size > 0) {
5          m_last->next = aux;
6          m_last = aux;
7      }
8      else {
9          m_last = m_first = aux;
10     }
11     m_size++;
12 }
```

Queue.cpp — Implementação da Fila

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

```
1  template<typename T>
2  void Queue<T>::pop() {
3      if(m_size != 0) {
4          Node<T> *aux = m_first;
5          m_first = aux->next;
6          aux->next = nullptr;
7          delete aux;
8          m_size--;
9      }
10 }
```

Arquivo main.cpp


```
1  #include <iostream>
2  #include <sstream>
3  #include "Queue.h"
4  using namespace std;
5
6  int main() {
7      Queue<int> fila; // cria fila vazia
8
9      for(int i = 1; i <= 9; i++)
10         fila.push(i);  // enfileira
11
12     while(!fila.empty()) {
13         cout << fila.front() << endl;
14         fila.pop();
15     }
16
17 }
```

Exemplos de aplicações de filas

Algumas aplicações de filas:

- Gerenciamento de fila de impressão
- Buffer do teclado
- Escalonamento de processos
- Comunicação entre aplicativos/computadores
- Percurso de estruturas de dados complexas (grafos etc.)

Exercícios

Exercício 1 (Filas)

Considere o tipo abstrato de dados Queue como definido nesta aula:

- Implemente uma função que receba três filas, `f_res`, `f1` e `f2`, e transfira alternadamente os elementos de `f1` e `f2` para `f_res`.

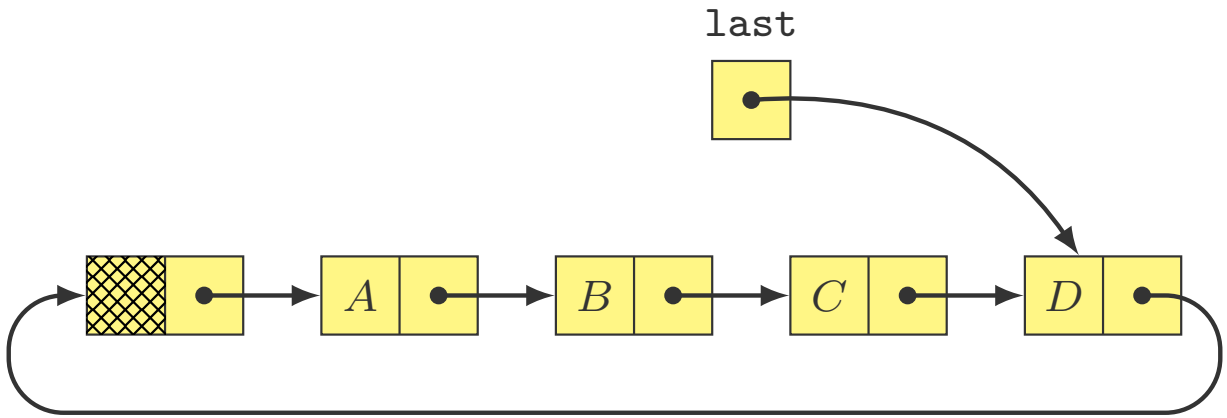
- Note que, ao final dessa função, as filas `f1` e `f2` vão estar vazias, e a fila `f_res` vai conter todos os valores originalmente em `f1` e `f2` (inicialmente `f_res` pode ou não estar vazia).

- Essa função deve obedecer ao protótipo:
  `void combina_filas(Queue& f_res, Queue& f1, Queue& f2)`

Exercício 2 — Implementação Alternativa (Filas)

Universidade Federal do Ceará
Campus Quixadá

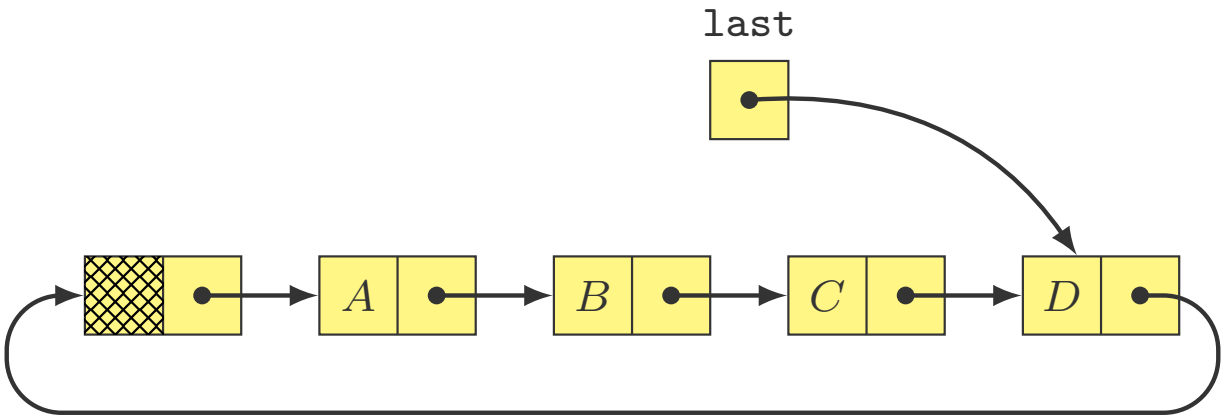
Exercício: implemente uma fila em uma lista encadeada circular com nó cabeça.



Exercício 2 — Implementação Alternativa (Filas)

Universidade Federal do Ceará
Campus Quixadá

Exercício: implemente uma fila em uma lista encadeada circular com nó cabeça.



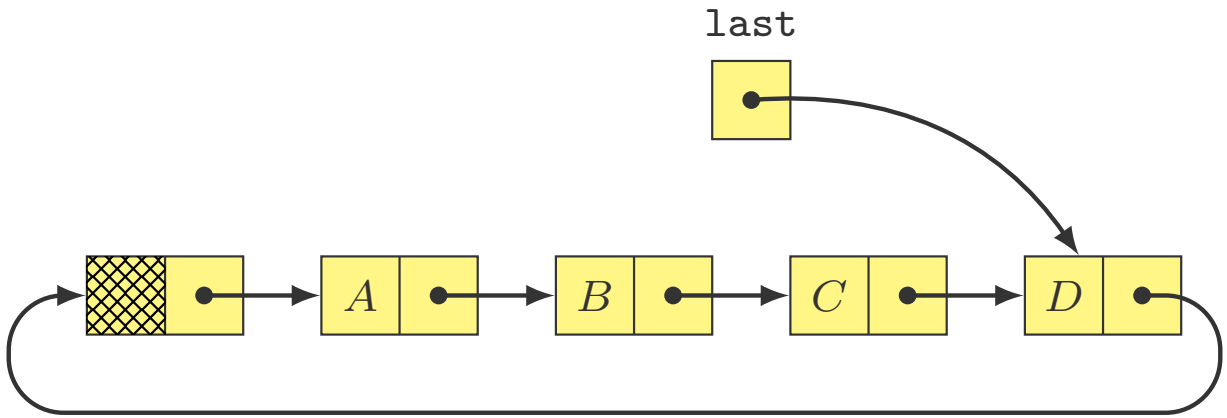
Enfileira:

- Atualizar o campo `next` de `last`
- Mudar `last` para apontar para o novo nó

Exercício 2 — Implementação Alternativa (Filas)

Universidade Federal do Ceará
Campus Quixadá

Exercício: implemente uma fila em uma lista encadeada circular com nó cabeça.



Enfileira:

- Atualizar o campo `next` de `last`
- Mudar `last` para apontar para o novo nó

Desenfileira:

- Basta remover o nó seguinte ao nó auxiliar
  - isto é, `last->next->next`

Uma aplicação de fila

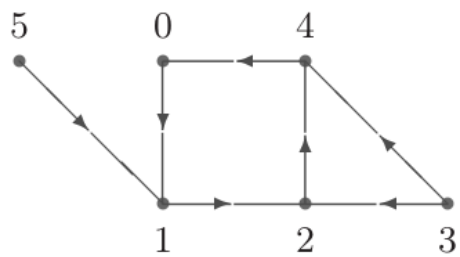
Aplicação: distâncias em uma rede

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

Imagine n cidades numeradas de 0 a $n-1$ e interligadas por estradas de mão única. As ligações entre as cidades são representadas por uma matriz A definida da seguinte maneira:

- $A[i][j] = 1$ se existe estrada da cidade i para a cidade j e $A[i][j] = 0$ em caso contrário.

	0	1	2	3	4	5
0	0	1	0	0	0	0
1	0	0	1	0	0	0
2	0	0	0	0	1	0
3	0	0	1	0	1	0
4	1	0	0	0	0	0
5	0	1	0	0	0	0



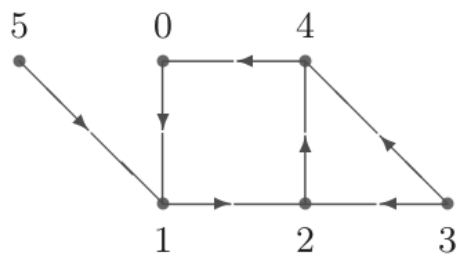
Aplicação: distâncias em uma rede

Universidade Federal do Ceará
Campus Quixadá

Imagine n cidades numeradas de 0 a $n-1$ e interligadas por estradas de mão única. As ligações entre as cidades são representadas por uma matriz A definida da seguinte maneira:

- $A[i][j] = 1$ se existe estrada da cidade i para a cidade j e $A[i][j] = 0$ em caso contrário.

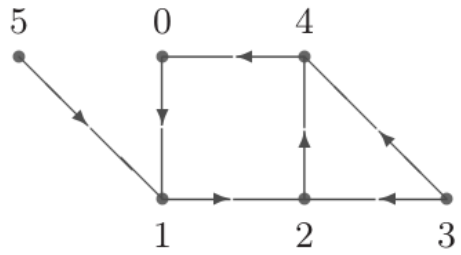
	0	1	2	3	4	5
0	0	1	0	0	0	0
1	0	0	1	0	0	0
2	0	0	0	0	1	0
3	0	0	1	0	1	0
4	1	0	0	0	0	0
5	0	1	0	0	0	0



A **distância** de uma cidade i para uma cidade j é o menor número de estradas que é preciso percorrer para ir de i a j. Nosso problema: determinar a distância de uma dada cidade i a cada uma das outras cidades.

Aplicação: distâncias em uma rede

	0	1	2	3	4	5
0	0	1	0	0	0	0
1	0	0	1	0	0	0
2	0	0	0	0	1	0
3	0	0	1	0	1	0
4	1	0	0	0	0	0
5	0	1	0	0	0	0



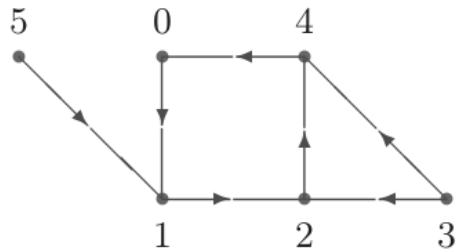
Distância da cidade 3 a cada uma das demais:

`dist = {2,3,1,0,1,-1}`

As distâncias serão armazenadas em um vetor `dist` de tal modo que `dist[x]` seja a distância de i a x. Se for impossível sair de i e chegar em x, dizemos que `dist[x] = -1`.

Aplicação: distâncias em uma rede

	0	1	2	3	4	5
0	0	1	0	0	0	0
1	0	0	1	0	0	0
2	0	0	0	0	1	0
3	0	0	1	0	1	0
4	1	0	0	0	0	0
5	0	1	0	0	0	0



Distância da cidade 3 a cada uma das demais:

`dist = {2,3,1,0,1,-1}`

As distâncias serão armazenadas em um vetor `dist` de tal modo que `dist[x]` seja a distância de i a x. Se for impossível sair de i e chegar em x, dizemos que `dist[x] = -1`.

A seguir mostramos um algoritmo que usa o conceito de fila para resolver nosso problema das distâncias.

Uma cidade é considerada **ativa** se já foi visitada mas as estradas que nela começam ainda não foram exploradas. O algoritmo mantém as cidades ativas numa fila. Em cada iteração, o algoritmo remove da fila uma cidade x e insere na fila todas as vizinhas de x que ainda não foram visitadas.

Criando a matriz


```
1       vector<vector<int>> matrix =
2       {
3           {0,1,0,0,0,0},
4           {0,0,1,0,0,0},
5           {0,0,0,0,1,0},
6           {0,0,1,0,1,0},
7           {1,0,0,0,0,0},
8           {0,1,0,0,0,0}
9       };
```

A matriz de 0s e 1s é criada usando o contêiner `vector` do C++ como acima. Essa matriz é a matriz da figura anterior.

Algoritmo


```
1  vector<int> distancias(vector<vector<int>>& mat, int origem) {
2      int n = mat.size(); // número total de cidades
3      vector<int> dist; // vetor de distancias
4
5      for(int i = 0; i < n; i++)
6          dist.push_back(-1);
7      dist[origem] = 0;
8
9      queue<int> fila_cidades;
10     fila_cidades.push(origem);
11
12     while(!fila_cidades.empty()) {
13         int cidade = fila_cidades.front();
14         fila_cidades.pop();
15         for(int i = 0; i < n; i++) {
16             if(mat[cidade][i] == 1 && dist[i] == -1) {
17                 dist[i] = dist[cidade] + 1;
18                 fila_cidades.push(i);
19             }
20         }
21     }
22     return dist;
23 }
```

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

FIM