Tipos Abstrato de Dados

Profª. Rose Yuri Shimizu

Roteiro

Tipos Abstratos de Dados - TAD

- **Estrutura de dados** com ações/operações particulares
  - Servindo de **modelo/tipo** para dados que se enquadrem nessas operações
- TAD(classe):
  - características/dados (atributos) + operações/comportamentos (métodos)
- É um **tipo de dado** que é acessada por uma **interface**:
  - Para usar: saber o que faz, e não, necessariamente, como faz
  - Programas clientes (que usam os dados) não acessam diretamente os valores
  - Acessam via funções fornecidas pela interface
  - Ocultamento de informação (caixa preta)
    - ★ Escondendo as estruturas de dados e a lógica de implementação
- **Tipos**: pilhas, filas, árvores

Roteiro

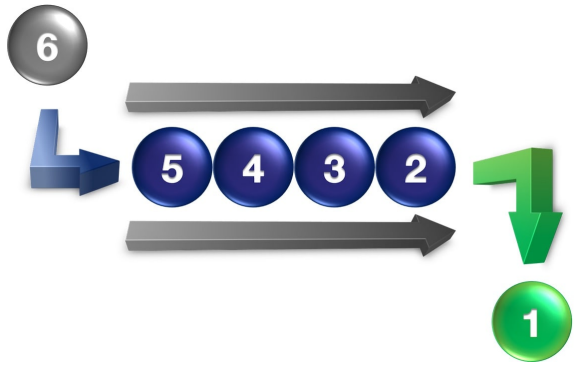
TAD Fila

O que é uma FILA?

Alinhamento de uma série de indivíduos ou objetos em sequência, de modo que um esteja imediatamente atrás do outro.

Processamento/atendimento de uma FILA?

Os dados que estão na frente são processados primeiro.



TAD Fila

FIFO (first-in first-out)

- Primeiro a entrar, é o primeiro a sair
- Justo: ordem de chegada/enfileiramento
  - Processamento de dados obedecendo a ordem de chegada
    - Sistema de inscrições
    - Julgadores automáticos
  - Fila de impressão
  - Fila de processos no sistema operacional
  - Gravação de mídias (ordem dos dados importa)
  - Busca: varredura pelos mais próximos primeiro

TAD Fila

FIFO (first-in first-out)

- Inserções no fim, remoções no início
- COMPLEXIDADE CONSTANTE
- Operações básicas:
  - vazia
  - tamanho
  - primeiro - busca_inicio
  - ultimo - busca_fim
  - enfileira - insere_fim
  - desenfileira - remove_inicio

Roteiro

TAD Fila - FIFO (first-in first-out) - lista estática

Implementação com lista estática

- `https://www.ime.usp.br/~pf/algoritmos/aulas/fila.html`
- Exemplo de uma implementação
- Operações constantes:
  - REMOÇÃO NO INÍCIO DA FILA
  - INSERÇÃO NO FIM DA FILA

TAD Fila - FIFO (first-in first-out) - lista estática

```
1
2  fila[p..u-1]     7 posições
3
4
5  inicio da fila          fim da fila
6         |                     |
7         p                     u
8    0    1     2     3     4     5     6     7     8
9  ------------------------------------------------------------
10 |    | 11  | 22  | 55  | 99  |     |     |  1  |  5  |     |
11 ------------------------------------------------------------
12                                          p     u
13
14
```

TAD Fila - FIFO (first-in first-out) - lista estática

- CRIAÇÃO DA FILA

```c
#define N 7

int fila[N];
int p, u;

void criar_fila ()
{
    p = u = 0;
}
```

```
 fila[p..u-1]     7 posições


 inicio e fim da fila
   |
  p=u
    0    1     2     3     4     5     6      7      8
 ---------------------------------------------------------
 |    |     |     |     |     |     |     |  0   |  0   |      |
 ---------------------------------------------------------
                                           p      u
```

TAD Fila - FIFO (first-in first-out) - lista estática

- FILA VAZIA

```c
#define N 7

int fila[N];
int p, u;

int vazia ()
{
    return p == u;
}
```

```
        início e fim da fila
                |
              p=u
    0     1     2     3     4     5     6     7     8
 ---------------------------------------------------------
 |     |     |     |     |     |     |     |     |     |
 ---------------------------------------------------------
```

TAD Fila - FIFO (first-in first-out) - lista estática

- REMOÇÃO NO INÍCIO DA FILA - desenfileirar
- Início da fila **p** é deslocado para mais próximo do fim
  - "removendo" o elemento da fila

```
1
2  fila[p..u-1]:
3     remover o elemento fila[p]
4
5
6  início da fila           fim da fila
7          |                    |
8          p = p+1              u
9    0     1     2     3     4     5     6     7     8
10 ---------------------------------------------------------
11 |    | 11 | 22 | 55 | 99 |     |     | 1+1 |  5  |     |
12 ---------------------------------------------------------
13                                      p     u
```

TAD Fila - FIFO (first-in first-out) - lista estática

- REMOÇÃO NO INÍCIO DA FILA - desenfileirar
- Início da fila **p** é deslocado para mais próximo do fim
  - "removendo" o elemento da fila

```
1  #define N 7
2
3  int fila[N];
4  int p, u;
5
6  int desenfileira()
7  {
8      return fila[p++]; //p++ ou ++p?
9  }
10
```

TAD Fila - FIFO (first-in first-out) - lista estática

- INSERÇÃO NO FIM DA FILA - enfileirar
- Elemento é colocado na posição indicada por **u**
  - fim da fila é deslocado

```
1
2  fila[p..u-1]:
3     inserir o elemento fila[u] = 88
4
5
6        início da fila       fim da fila
7                 |                 |
8                 p                 u = u+1
9     0     1     2     3     4     5     6     7     8
10 ---------------------------------------------------------
11 |     | 11  | 22  | 55  | 99  | 88  |     |  2  | 5+1 |     |
12 ---------------------------------------------------------
13                                        p     u
14
```

TAD Fila - FIFO (first-in first-out) - lista estática

- INSERÇÃO NO FINAL DA FILA - enfileirar
- Elemento é colocado na posição indicada por **u**
  - fim da fila é deslocado

```
1  #define N 7
2
3  int fila[N];
4  int p, u;
5
6  void enfileira (int y)
7  {
8      fila[u++] = y; //u++ ou ++u?
9  }
10
```

TAD Fila - FIFO (first-in first-out) - lista estática

- Fila cheia
  - u == N

```
1
2  fila[p..u-1] : N = 7
3
4
5        início da fila                    fim da fila - "fila cheia"
6                 |                              |
7                 p                              u
8     0    1     2     3     4     5     6     7      8
9  ---------------------------------------------------------------
10 |    |     | 22  | 55  | 99  | 88  | 33  |  2   |  7   |      |
11 ---------------------------------------------------------------
12                                     N-1    p      u
13
14
```

- E se inserir em lista cheia?
  - Ocorre o transbordamento

TAD Fila - FIFO (first-in first-out) - lista estática

- Inserção em fila cheia
- Transbordamento

```
1
2  fila[p .. u-1] : N = 7
3
4  fila[u] = 4
5  fila[7] = 4, como 7 > N-1, ocorre transbordamento
6
7
8        início da fila                      fim da fila
9                 |                             |
10                p                             u   =   u+1
11    0     1     2     3     4     5     6      7        8
12 ---------------------------------------------------------------
13 |    |     | 22 | 55 | 99 | 88 | 33 | 2 -> 4 | 7+1 |      |
14 ---------------------------------------------------------------
15                                    N-1     p         u
16                                            |
17                                    invadiu a área de p
18
```

TAD Fila - FIFO (first-in first-out) - lista estática

- Inserção em fila cheia
- Transbordamento: resultado inesperado

```
1
2  fila[p..u-1] : N = 7
3
4  fila = [99, 88, 33, 4, 8] errada
5
6
7                  inicio da fila            fim da fila
8                          |                     |
9                          p                     u
10   0     1     2     3     4    5     6      7      8
11 ---------------------------------------------------------
12 |    |     | 22 | 55 | 99 | 88 | 33 |   4   |   8   |      |
13 ---------------------------------------------------------
14                                  N-1    p      u
15
16
```

TAD Fila - FIFO (first-in first-out)

- Problema: e se fila cheia, u == N, com espaços livres?
- Solução: chegou ao fim, volta para o primeiro (circular)

```
1
2  fila[p..u-1] : N = 7
3
4  fila[u] = 33
5
6
7        inicio da fila          fim da fila
8                |                     |
9                p                     u -> (u+1==N?u=0)
10    0     1     2     3     4     5     6     7     8
11 ---------------------------------------------------------------
12 |    |     | 22 | 55 | 99 | 88 | 33 |  2  | 6 -> 0 |     |
13 ---------------------------------------------------------------
14                                    N-1   p       u
15
16
```

TAD Fila - FIFO (first-in first-out)

lista estática circular

- Problema: e se fila cheia, u == N, com espaços livres?
- Solução: chegou ao fim, volta para o primeiro (circular)

```
1
2  void enfileira (int y)
3  {
4      fila[u++] = y;
5      if (u == N) u = 0; ⟸
6  }
7
8  int desenfileira()
9  {
10     int x = fila[p++];
11     if (p == N) p = 0; ⟸
12     return x;
13 }
```

TAD Fila - FIFO (first-in first-out)

lista estática circular - vazia x cheia

- Decisão: posição anterior a **p** fica vazio (diferenciar cheia e vazia)
  - Fila cheia:
    - u+1==p ou (u+1==N e p==0)
    - ou seja, se (u+1) % N == p

      (0+1)%7=1 | (1+1)%7=2 | (2+1)%7=3 |
      (3+1)%7=4 | (4+1)%7=5 | (5+1)%7=6 | (6+1)%7=0

  - Fila vazia: u==p

```
1
2  fila[p..u-1] : N = 7
3  cheia (1+1) % 7 = 2 = p
4
5              fim    início
6               |       |
7               u   = u+1=p
8      0        1       2      3      4     5      6      7      8
9    ---------------------------------------------------------------
10   | 44   |       | 22 | 55 | 99 | 88 | 33 | 2 |   1   |       |
11   ---------------------------------------------------------------
12                                          N-1    p      u
13
```

TAD Fila - FIFO (first-in first-out) - lista estática

Implementação com lista estática - possibilidade de ter várias filas

```
1  typedef int Item;
2  typedef struct {
3      Item *item;
4      int primeiro, ultimo;
5  }Fila;
6
7  Fila *criar( int maxN ){
8      Fila *p = malloc(sizeof(Fila));
9      p->item = malloc(maxN*sizeof(Item));
10     p->primeiro = p->ultimo = 0;
11     return p;
12 }
13
14 int vazia( Fila *f ){
15     return f->primeiro == f->ultimo;
16 }
17
18 int desenfileira(Fila *f) {
19     return f->item[f->primeiro++];
20 }
21
22 void enfileira (Fila *f, int y) {
23     f->item[f->ultimo++] = y;
24 }
```

TAD Fila - FIFO (first-in first-out) - lista estática

```
1  void imprime(Fila *f) {
2    printf("\nFILA p=%d e u=%d\n", f->primeiro, f->ultimo);
3    for(int i=f->primeiro; i<f->ultimo; i++)
4      printf(" F[%d] |", i);
5    printf("\n");
6
7    for(int i=f->primeiro; i<f->ultimo; i++)
8      printf(" %3d  |", f->item[i]);
9    printf("\n");
10 }
11
12 int main() {
13   printf("\n\nCriando a fila e enfileirando 10 elementos\n");
14   Fila *fila1 = criar(100);
15   for(int i=0; i<10; i++) enfileira(fila1, i);
16   imprime(fila1);
17
18   printf("\n\nDesenfileirando os 3 primeiros elementos\n");
19   for(int i=fila1->primeiro; i<3; i++) desenfileira(fila1);
20   imprime(fila1);
21
22   Fila *fila2 = criar(400);
23   ...
24
25   return 0;
26 }
27
```

TAD Fila - FIFO (first-in first-out)

lista estática com redimensionamento

- Problema: fila cheia, u == N, com espaços livres na fila
- Solução: redimensionamento da lista que armazena a fila

```
1  void redimensiona (void) {
2      N *= 2;
3      fila = realloc (fila, N * sizeof (int));
4  }
5
6  //reajustar as variáveis p e u de acordo
7  void redimensiona () {
8      N *= 2; //evitar novos redimensionamentos
9      int *novo = malloc (N * sizeof (int));
10
11     int j=0;
12     for (int i = p; i < u; i++, j++)
13         novo[j] = fila[i];
14
15     p = 0;
16     u = j;
17
18     free (fila);
19     fila = novo;
20 }
```

Roteiro

TAD Fila - FIFO (first-in first-out)

Implementação com lista encadeada

- INSERÇÕES NO FINAL DA FILA
- REMOÇÕES NO INÍCIO DA FILA
- COMPLEXIDADE CONSTANTE: **possível com listas encadeadas?**

TAD Fila - FIFO (first-in first-out) - lista encadeada

- `Item desenfileira(cabeca *lista)`
  - Remover o elemento mais velho $\Longrightarrow$ início da fila
  - Acesso ao primeiro elemento $\Longrightarrow$ constante
- `void enfileira(cabeca *lista, no *novo)`
  - Inserir no fim da lista
  - Acesso ao último elemento $\Longrightarrow$ constante

TAD Fila - FIFO (first-in first-out) - lista encadeada

- Acesso ao primeiro elemento: todas as listas $\Longrightarrow$ constante
- Acesso ao último elemento:
  - Lista simplesmente/duplamente encadeada com cabeça:
    - metadados $\rightarrow$ `lista->ultimo`
  - Lista duplamente encadeada circular:
    - `no *ultimo = lista->prox->ant` (anterior do primeiro)
  - Lista simplesmente encadeada circular modificada:
    - último elemento apontar para a cabeça (mesmo tipo dos outros nós)
    - inserir na cabeça
    - criar uma nova cabeça que aponta para o que velha cabeça aponta
    - velha cabeça/novo elemento: aponta para ??
    - Implementem!
  - Lista simplesmente encadeada com cauda:
    - Podemos utilizar um apontador direto para a cauda

TAD Fila - FIFO (first-in first-out) - lista encadeada

```
1  //criar e enfileirar
2  void enfileira(cabeca *lista, Item x) {
3      no *novo = criar_no(x); ⟸
4      if(novo){
5          if(vazia(lista)) lista->prox = novo;
6          else lista->ultimo->prox = novo;
7
8          lista->ultimo = novo; ⟸ //inserir_fim
9          novo->prox = NULL;
10
11         lista->tam++;
12     }
13 }
```

TAD Fila - FIFO (first-in first-out) - lista encadeada

```
1  //desenfileira e devolve o item
2   Item desenfileira(cabeca *lista)
3   {
4       no *lixo = lista->prox; ⟸   //remover_inicio
5       lista->prox = lixo->prox;
6
7       if(vazia(lista)) lista->ultimo = NULL;
8       lista->tam--;
9
10      Item x = lixo->info; ⟸
11      free(lixo); ⟸
12      return x; ⟸
13  }
```

TAD Fila - FIFO (first-in first-out) - exemplo

- Vivo ou morto(`https://br.spoj.com/problems/VIVO/`)
  - Vários participantes (números inteiros) em uma fila única
  - Ordem: "vivo" (1 - levantar); "morto" (0 - abaixar)
  - O participante que não seguir a ordem, é retirado fazendo a "fila andar"
  - O jogo continua até que uma rodada tenha apenas um participante (vencedor)
- Entrada:
  - P e R: número inicial de participantes(códigos de 1 a P) e rodadas de ordens
  - P códigos dos participantes na ordem em que estão na fila
  - Cada uma das R linhas seguintes
    - ★ Um número inteiro N indicando o número de participantes da rodada
    - ★ Um número inteiro inteiro J (0/morto ou 1/vivo) representando a ordem dada
    - ★ N números inteiros representando a ação (0/morto ou 1/vivo) do participante na i-ésima posição na fila
  - Saída: código do vencedor
- Exemplo:

```
Entrada:
5 4
3 2 1 4 5
5 1 1 1 1 1 1
5 0 0 1 0 1 0
3 0 0 1 0
2 1 0 1
```

```
Saída:
5
```

TAD Fila - FIFO (first-in first-out) - exemplo

```
criar_fila();

while(participantes--){
    scanf("%d", &i);
    enfileira(i)
}

while(rodadas--){
    scanf("%d %d", &n, &e);
    while(n--){
        scanf("%d", &s);
        x = desenfileira();
        if(s == e) enfileira(x);
    }
}
printf("%d\n", desenfileira());

```

TAD Fila - FIFO (first-in first-out) - exemplo

-	-	-	1	2	1	-	-	-
-	-	1	1		1	1	-	-
-	-	1				3	-	-
-	-	1				2	-	-
-	2	1				1	-	-
-	1					1	-	-
-	1		[l, c]			1	2	-
-	1						1	-
-	2	1					1	-
-	-	3	1		2	1	1	-
-	-	-	1	1	1	-	-	-
-	-	-	-	-	-	-	-	-

- Campo minado - abrir casas

TAD Fila - FIFO (first-in first-out) - exemplo

```
        ?       ?       ?       ?       ?
        ?   [-l,-c][-l, c][-l,+c]   ?
        ?   [ l,-c][ l, c][ l,+c]   ?
        ?   [+l,-c][+l, c][+l,+c]   ?
        ?       ?       ?       ?       ?
```

- Campo minado - abrir casas
- A partir da posição `l` e `c`, abrir os adjacentes
- Enfileirar os adjacentes para posterior tratamento

```c
int abrir_mapa(int m, int n, struct area campo[m][n], int l, int c) {
  campo[l][c].visivel = 1;
  if(campo[l][c].item==-1) return campo[l][c].item; //-1 perdeu
```

```c
int abrir_mapa(int m, int n, struct area campo[m][n], int l, int c) {
  campo[l][c].visivel = 1;
  if(campo[l][c].item==-1) return campo[l][c].item; //-1 perdeu

  struct head fila;
  struct celula *no = malloc(sizeof(struct celula));
  no->l = l;
  no->c = c;

  STAILQ_INIT(&fila);
  STAILQ_INSERT_TAIL(&fila, no, prox);
```

```c
int abrir_mapa(int m, int n, struct area campo[m][n], int l, int c) {
  campo[l][c].visivel = 1;
  if(campo[l][c].item==-1) return campo[l][c].item; //-1 perdeu

  struct head fila;
  struct celula *no = malloc(sizeof(struct celula));
  no->l = l;
  no->c = c;

  STAILQ_INIT(&fila);
  STAILQ_INSERT_TAIL(&fila, no, prox);

  while(!STAILQ_EMPTY(&fila)) {
```

```c
int abrir_mapa(int m, int n, struct area campo[m][n], int l, int c) {
  campo[l][c].visivel = 1;
  if(campo[l][c].item==-1) return campo[l][c].item; //-1 perdeu

  struct head fila;
  struct celula *no = malloc(sizeof(struct celula));
  no->l = l;
  no->c = c;

  STAILQ_INIT(&fila);
  STAILQ_INSERT_TAIL(&fila, no, prox);

  while(!STAILQ_EMPTY(&fila)) {
    struct celula *no2 = STAILQ_FIRST(&fila);
    STAILQ_REMOVE_HEAD(&fila, prox);
```

```c
int abrir_mapa(int m, int n, struct area campo[m][n], int l, int c) {
  campo[l][c].visivel = 1;
  if(campo[l][c].item==-1) return campo[l][c].item; //-1 perdeu

  struct head fila;
  struct celula *no = malloc(sizeof(struct celula));
  no->l = l;
  no->c = c;

  STAILQ_INIT(&fila);
  STAILQ_INSERT_TAIL(&fila, no, prox);

  while(!STAILQ_EMPTY(&fila)) {
    struct celula *no2 = STAILQ_FIRST(&fila);
    STAILQ_REMOVE_HEAD(&fila, prox);

    if(campo[no2->l][no2->c].item==0) {
```

```c
int abrir_mapa(int m, int n, struct area campo[m][n], int l, int c) {
  campo[l][c].visivel = 1;
  if(campo[l][c].item==-1) return campo[l][c].item; //-1 perdeu

  struct head fila;
  struct celula *no = malloc(sizeof(struct celula));
  no->l = l;
  no->c = c;

  STAILQ_INIT(&fila);
  STAILQ_INSERT_TAIL(&fila, no, prox);

  while(!STAILQ_EMPTY(&fila)) {
    struct celula *no2 = STAILQ_FIRST(&fila);
    STAILQ_REMOVE_HEAD(&fila, prox);

    if(campo[no2->l][no2->c].item==0) {
      for(int i=no2->l-1; i<=no2->l+1; i++) {
        for(int j=no2->c-1; j<=no2->c+1; j++) {
```

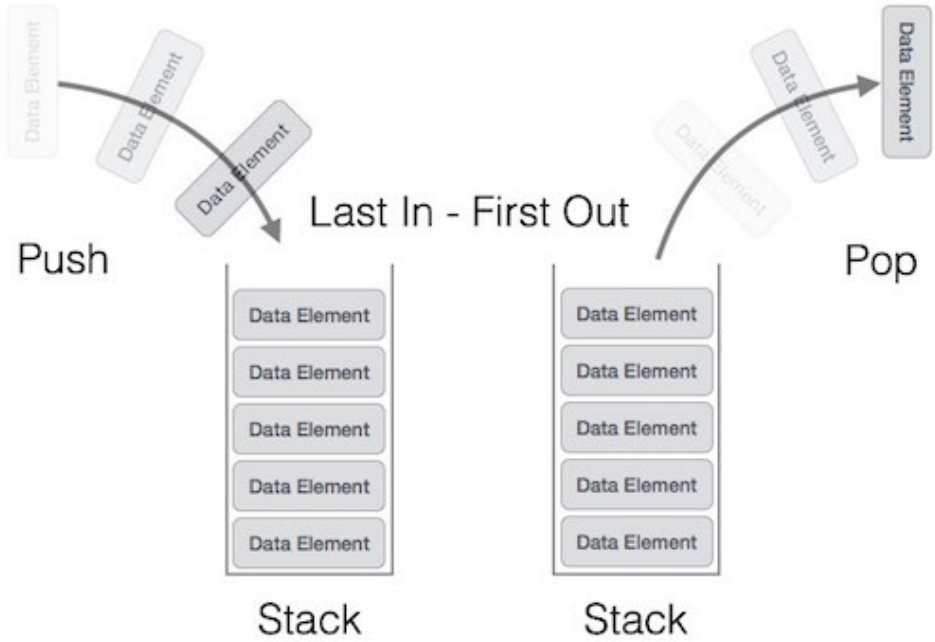
```c
int abrir_mapa(int m, int n, struct area campo[m][n], int l, int c) {
  campo[l][c].visivel = 1;
  if(campo[l][c].item==-1) return campo[l][c].item; //-1 perdeu

  struct head fila;
  struct celula *no = malloc(sizeof(struct celula));
  no->l = l;
  no->c = c;

  STAILQ_INIT(&fila);
  STAILQ_INSERT_TAIL(&fila, no, prox);

  while(!STAILQ_EMPTY(&fila)) {
    struct celula *no2 = STAILQ_FIRST(&fila);
    STAILQ_REMOVE_HEAD(&fila, prox);

    if(campo[no2->l][no2->c].item==0) {
      for(int i=no2->l-1; i<=no2->l+1; i++) {
        for(int j=no2->c-1; j<=no2->c+1; j++) {
          if(i>=0 && i<m && j>=0 && j<n && campo[i][j].visivel==0) {
            campo[i][j].visivel = 1;
            struct celula *no3 = malloc(sizeof no3);
            no3->l = i; no3->c = j;
            STAILQ_INSERT_TAIL(&fila, no3, prox);
          }
        }
      }
    }
    free(no2);
  }
  return 0;
}
```

Roteiro

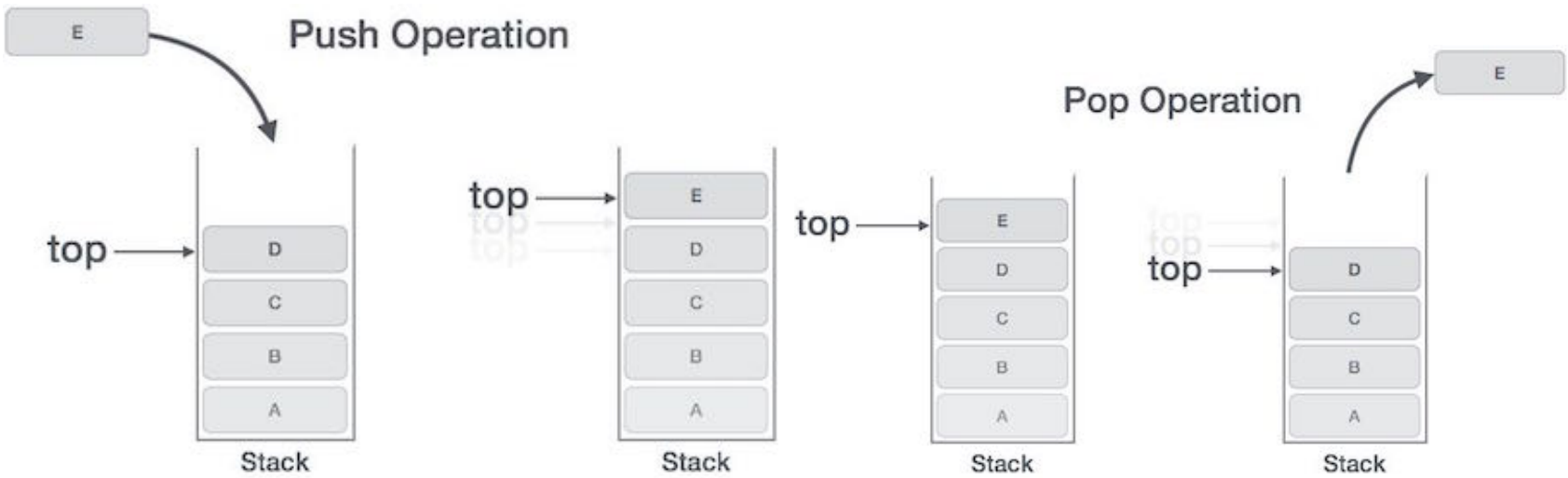
TAD Pilha - LIFO (Last In, First Out)

- Listas com o comportamento **LIFO (Last In, First Out)**: último a entrar, primeiro a sair;

- **Operações** que definem o comportamento de pilha:
  1. **criar**: uma pilha vazia;
  2. **vazia**: verificar se está vazia;
  3. **empilhar**: inserir um item no topo;
  4. **desempilhar** remover o item mais recente;
  5. **espiar** o item do topo.



TAD Pilha - LIFO (Last In, First Out)



TAD Pilha - LIFO (Last In, First Out)

- **Problemas clientes** das pilhas:
  - **Desfazer/Refazer**
  - **Histórico de navegadores**
  - **Gerenciamento de memória**: pilhas de memória são utilizadas para armazenar todas as variáveis de um programa em execução
  - **Recursão**: as chamadas de função são mantidas por pilha de memória
  - **Busca em profundidade**: percorrer uma possibilidade completa antes de analisar o próximo caminho
  - ***Backtracking***: poder voltar a um ponto para refazer uma decisão
  - **Inversão** de strings
  - **Balanceamento de símbolos** ([{}]): verificação sintaxe (compiladores)
  - **Identificador de expressões/palavras** (tokens): verificação léxica (compiladores)
  - **Conversão de expressões**: infixo para prefixo, posfixo para infixo, etc.

Roteiro

TAD Pilha - LIFO (Last In, First Out)

listas estáticas

Implementação com lista estática

- `https://www.ime.usp.br/~pf/algoritmos/aulas/pilha.html`
- Exemplo de uma implementação
- Operações constantes:
  - REMOÇÃO NO TOPO DA PILHA
  - INSERÇÃO NO TOPO DA PILHA
- Decisão: TOPO???

TAD Pilha - LIFO (Last In, First Out)

listas estáticas

- CRIAÇÃO DA PILHA

```
1 int *pilha;
2 int t;
3
4 void criar(int N)
5 {
6     pilha = malloc(N * sizeof(int));
7     t=0;
8 }
```

```
1     pilha[0..t-1] -> tamanho 7
2
3                           topo da pilha
4                                |
5                                t
6     0    1     2     3     4     5     6     7     8
7  ---------------------------------------------------------
8  | 10 | 11 | 22 | 55 | 99 |     |     | 5   |     |     |
9  ---------------------------------------------------------
10                                      t
11
```

TAD Pilha - LIFO (Last In, First Out)

listas estáticas

- INSERÇÃO NO TOPO DA PILHA - empilhar
- Elemento é colocado na posição indicada por **t**
  - topo da pilha é deslocado

```
1     pilha[0..t-1] -> tamanho 7
2     Inserir o elemento pilha[t]
3         atualiza o topo t=t+1
4
5
6                     topo da pilha
7                          |
8                          t -> t++
9     0    1     2     3     4      5     6     7     8
10  ---------------------------------------------------------
11  | 10 | 11 | 22 | 55 | 99 |     |     |  4  |     |     |
12  ---------------------------------------------------------
13                                          t
14
```

TAD Pilha - LIFO (Last In, First Out)

listas estáticas

- INSERÇÃO NO TOPO DA PILHA - empilhar
- Elemento é colocado na posição indicada por **t**
  - topo da pilha é deslocado

```
1 int  *pilha;
2 int  t;
3
4 void  empilha  (Item  y)
5 {
6      pilha[t++]  =  y;
7 }
```

TAD Pilha - LIFO (Last In, First Out)

listas estáticas

- REMOÇÃO NO TOPO DA FILA - desempilhar
- Topo da pilha **t** é deslocado para mais próximo do início
  - "removendo" o elemento da pilha
  - Item indicado pela nova posição do topo é ignorado

```
1  pilha[0..t-1] -> tamanho 7
2  Remover o elemento
3     elemento removido pilha[t-1]
4     atualizar o topo t=t-1
5
6
7                    topo da pilha
8                          |
9                        t-1 <- t
10    0    1     2     3     4      5     6      7      8
11 ------------------------------------------------------------
12 | 10 | 11 | 22 | 55 | 99 |     |     |   5-1  |     |      |
13 ------------------------------------------------------------
14                                            t
15
```

TAD Pilha - LIFO (Last In, First Out)

listas estáticas

- REMOÇÃO NO TOPO DA FILA - desempilhar
- Topo da pilha **t** é deslocado para mais próximo do início
  - "removendo" o elemento da pilha
  - Item indicado pela nova posição do topo é ignorado

```
1 int *pilha;
2 int t;
3
4 Item desempilha()
5 {
6     return pilha[--t]; //--t ou t-- ??
7 }
```

TAD Pilha - LIFO (Last In, First Out)

listas estáticas

- ESPIA e FILA VAZIA

```
1  int *pilha;
2  int t;
3
4  Item espia() {
5      return pilha[t-1];
6  }
7
8  int vazia () {
9      return t == 0;
10 }
```

```
1      pilha[0..t-1] -> tamanho 7
2
3  topo da pilha
4     |
5     t
6     0     1     2     3     4     5     6     7     8
7   ---------------------------------------------------------
8   |     |     |     |     |     |     |  0  |     |
9   ---------------------------------------------------------
10                                          t
```

TAD Pilha - LIFO (Last In, First Out)

listas estáticas

- PROBLEMA: fila cheia, u == N, com espaços livres na fila???

```
    pilha[0..t-1] -> tamanho 7


                                        fim da pilha - pilha cheia
                                                 |
                                                 t
  0     1     2     3     4     5     6     7      8
-------------------------------------------------------------------
| 11 | 44 | 22 | 55 | 99 | 88 | 33 | 7 |      |      |
-------------------------------------------------------------------
                                                 t
```

TAD Pilha - LIFO (Last In, First Out)

```c
typedef char Item;
typedef struct {
    Item *item;
    int topo;
}Pilha;

Pilha *criar( int maxN ) {
    Pilha *p = malloc(sizeof(Pilha));
    p->item = malloc(maxN*sizeof(Item));
    p->topo = 0;
    return p;
}

int vazia( Pilha *p ) {
    return p->topo == 0;
}

void empilhar( Pilha *p, Item item ) {
    p->item[p->topo++] = item;
}

Item desempilhar( Pilha *p ) {
    return p->item[--p->topo];
}

Item espiar( Pilha *p ) {
    return p->item[p->topo-1];
}
```

Roteiro

TAD Pilha - LIFO (Last In, First Out)

Implementação com lista encadeada

- INSERÇÕES NO TOPO DA PILHA
- REMOÇÕES DO TOPO DA FILA
- COMPLEXIDADE CONSTANTE:
  - TOPO: ???
  - **possível com listas encadeadas?**

```
1  //EMPILHA NO TOPO - TOPO??? - inserir_???
2  void empilha(head *lista, Item x)
3  {
4      node *novo = criar_no(x); ⟸
5      if(novo){
6          if(vazia(lista)) lista->ultimo = novo;
7
8          novo->prox = lista->prox;
9          lista->prox = novo;  ⟸
10
11         lista->num_itens++;
12     }
13 }
```

```
1  //DESEMPILHA DO TOPO - TOPO??? - remover_???
2  Item desempilha(head *lista)
3  {
4      node *topo = lista->prox;
5      lista->prox = topo->prox; ⟸
6
7      if(topo == lista->ultimo) lista->ultimo = NULL;
8      lista->num_itens--;
9
10     Item x = topo->info; ⟸
11     free(topo);          ⟸
12     return x;            ⟸
13 }
14
15 Item espia(head *p)
16 {
17     return (p->prox->info);
18 }
```

TAD Pilha - LIFO (Last In, First Out) - exemplo

Problema - Calculadora posfixada

Desenvolva um programa que leia da entrada padrão uma expressão matemática posfixada, compute o resultado e mostre na saída padrão.

Entrada: 5 9 8 + 4 6 * * 7 + *

Saída: 2075

./calcula "5 9 8 + 4 6 * * 7 + *"

```c
int main (int argc, char *argv[]) {
  char *a = argv[1];

  head *pilha = criar_lista();

  for(int i=0; a[i]!='\0'; i++) {

    //operacao do operador sobre os ultimos operandos lidos
    if(a[i] == '+')
      empilha(pilha, desempilha(pilha)+desempilha(pilha));
    if(a[i] == '*')
      empilha(pilha, desempilha(pilha)*desempilha(pilha));

    //colocar zero a esquerda
    if((a[i] >= '0') && (a[i] <= '9')) empilha(pilha, 0);

    //calcular o equivalente numerico de uma
    //  sequencia de caracteres
    while((a[i] >= '0') && (a[i] <= '9'))
      //calcula o decimal, centena ... + valor numerico
      empilha(pilha, 10*desempilha(pilha) + (a[i++]-'0'));
  }
  printf("%d \n", desempilha(pilha));
}
```

TAD Pilha - LIFO (Last In, First Out) - exemplo

Problema - Balanceamento de símbolos

Identificar se a sintaxe dos modificadores de negrito (*), itálico (/), e sublinhado (_) estão corretos.

Exemplos:
negrito
isso eh negrito e /italico/
erro /e

Entrada: 6 *b/i/*

```c
int  n;
char c;
scanf("%d", &n); //tamanho da expressão

pilha *p = criar(); //criamos a pilha

while(n>0 && scanf(" %c",&c)==1) {

  if(c=='*'||c=='/'||c=='_')
  {
    if(!vazia(p) && espiar(p) == c)
      desempilhar();
    else
      empilhar(c);
  }
  n--;
}

if(vazia(p))
    printf("C\n");
else
    printf("E\n");
```

b/i/

```
[   ]
[   ]
[   ]
[   ]
[   ]
[   ]
```

Entrada: 6 *b/i/*

```
1
2  //ler os n caracteres
3  while(n>0 && scanf(" %c",&c)==1) {
4    //achou o simbolo
5    if(c=='*'||c=='/'||c=='_')
6    {
7      //pilha vazia
8      if(!vazia(p) && espiar(p) == c)
9        desempilhar();
10     else
11       empilhar(c);
12   }
13   n--;
14 }
15
16 if(vazia(p))
17   printf("C\n");
18 else
19   printf("E\n");
```

b/i/

```
1          [    ]
2          [    ]
3          [    ]
4          [    ]
5          [    ]
6          [    ]
7
```

Entrada: 6 *b/i/*

```
1
2  //consumiu a entrada != simbolos
3  while(n>0 && scanf(" %c",&c)==1) {   <=
4
5    if(c=='*'||c=='/'||c=='_')
6    {
7
8      if(!vazia(p) && espiar(p) == c)
9        desempilhar();
10     else
11       empilhar(c);
12   }
13   n--;
14 }
15
16 if(vazia(p))
17   printf("C\n");
18 else
19   printf("E\n");
```

b/i/

```
1          [    ]
2          [    ]
3          [    ]
4          [    ]
5          [    ]
6          [ *  ]
7
```

Entrada: 6 *b/i/*

```
1
2
3  while(n>0 && scanf(" %c",&c)==1) {   ⟸
4
5    if(c=='*'||c=='/'||c=='_')   ⟸
6    {
7      //topo igual a /?
8      if(!vazia(p) && espiar(p) == c)   ⟸
9        desempilhar();
10     else
11       empilhar(c);   ⟸
12   }
13   n--;
14 }
15
16 if(vazia(p))
17   printf("C\n");
18 else
19   printf("E\n");
```

b/i/

```
1          [    ]
2          [    ]
3          [    ]
4          [    ]
5          [    ]
6          [ *  ]
7
```

Entrada: 6 *b/i/*

```
1
2  //consumiu a entrada != simbolos
3  while(n>0 && scanf(" %c",&c)==1) {   <=
4
5    if(c=='*'||c=='/'||c=='_')
6    {
7
8      if(!vazia(p) && espiar(p) == c)
9        desempilhar();
10     else
11       empilhar(c);
12   }
13   n--;
14 }
15
16 if(vazia(p))
17   printf("C\n");
18 else
19   printf("E\n");
```

b/i/

```
1          [    ]
2          [    ]
3          [    ]
4          [    ]
5          [ /  ]
6          [ *  ]
7
```

Entrada: 6 *b/i/*

```
1
2
3  while(n>0 && scanf(" %c",&c)==1) {
4
5    if(c=='*'||c=='/'||c=='_')
6    {
7      //topo igual /?
8      if(!vazia(p) && espiar(p) == c)
9        desempilhar();
10     else
11       empilhar(c);
12   }
13   n--;
14 }
15
16 if(vazia(p))
17   printf("C\n");
18 else
19   printf("E\n");
```

b/i/

```
1          [    ]
2          [    ]
3          [    ]
4          [    ]
5          [ /  ]
6          [ *  ]
7
```

Entrada: 6 *b/i/*

```
1
2
3  while(n>0 && scanf(" %c",&c)==1) {
4
5    if(c=='*'||c=='/'||c=='_')
6    {
7      //topo igual *?
8      if(!vazia(p) && espiar(p) == c)
9        desempilhar();
10     else
11       empilhar(c);
12   }
13   n--;
14 }
15
16 if(vazia(p))
17   printf("C\n");
18 else
19   printf("E\n");
```

b/i/

```
1          [    ]
2          [    ]
3          [    ]
4          [    ]
5          [    ]
6          [ *  ]
7
```

Entrada: 6 *b/i/*

```
1
2
3  while(n>0 && scanf(" %c",&c)==1) {
4
5    if(c=='*'||c=='/'||c=='_')
6    {
7
8      if(!vazia(p) && espiar(p) == c)
9        desempilhar();
10     else
11       empilhar(c);
12   }
13   n--;
14 }
15
16 if(vazia(p))          <=
17   printf("C\n");      <=
18 else
19   printf("E\n");
```

b/i/

```
1   [    ]
2   [    ]
3   [    ]
4   [    ]
5   [    ]
6   [    ]
7
```

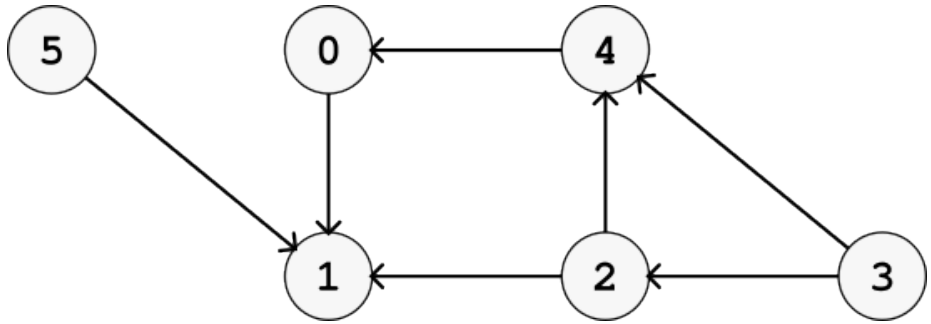
Roteiro

Problema usando **TAD Fila**

Distância das demais cidade

- `https://www.ime.usp.br/~pf/algoritmos/aulas/fila.html`
- Problema:
  - Dada uma cidade c
  - Encontrar a distância (menor número de estradas) de c a cada uma das demais cidades.
- Dado um mapa:
  - A[i][j] vale 1 se existe estrada de i para j e vale 0 em caso contrário

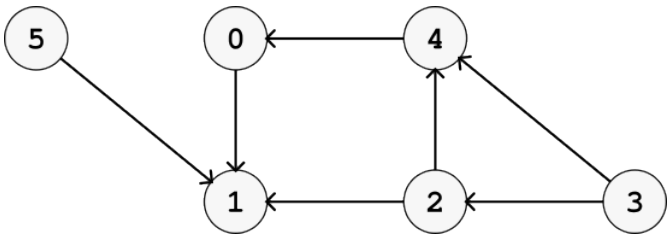
		destinos					
		0	1	2	3	4	5
origens	0	0	1	0	0	0	0
	1	0	0	0	0	0	0
	2	0	1	0	0	1	0
	3	0	0	1	0	1	0
	4	1	0	0	0	0	0
	5	0	1	0	0	0	0



TAD Fila - FIFO (first-in first-out) - Exemplo

Exemplo: distâncias da cidade **3**

```
                                    partidas   cidades alcançáveis
                                     3 ------- 3 (3 para 3 = 0 estrada)
FILA [ 3 ]

mapa[origens][destinos]
      0  1  2  3  4  5
 0    0  1  0  0  0  0
 1    0  0  0  0  0  0
 2    0  1  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
```

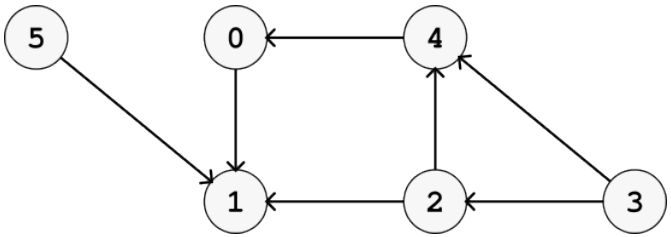


TAD Fila - FIFO (first-in first-out) - Exemplo

Exemplo: distâncias da cidade **3**

```
FILA [ 2 4 ]

mapa[origens][destinos]
      0   1   2   3   4   5
 0    0   1   0   0   0   0
 1    0   0   0   0   0   0
 2    0   1   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
```

```
partidas    cidades alcançáveis
 3 -------  3 (3 para 3 = 0 estrada)

 3 -------  2 4 (3 para 2 = 1 estrada)
                (3 para 4 = 1 estrada)
```



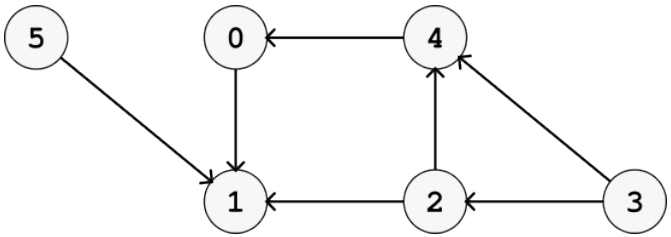
TAD Fila - FIFO (first-in first-out) - Exemplo

Exemplo: distâncias da cidade **3**

```
FILA [ 4 1 ]
```

```
mapa[origens][destinos]
     0  1  2  3  4  5
 0   0  1  0  0  0  0
 1   0  0  0  0  0  0
[2]  0 [1] 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
```

```
partidas   cidades alcançáveis
 3 ------- 3 (3 para 3 = 0 estrada)

 3 ------- 2 4 (3 para 2 = 1 estrada)
              (3 para 4 = 1 estrada)

 2 ------- 1 4 (3 para 2 = 1 estrada e
               2 para 1 = 1 estrada, portanto,
               3 para 1 = 2 estradas)
              (4 já visitada - rota ignorada)
```

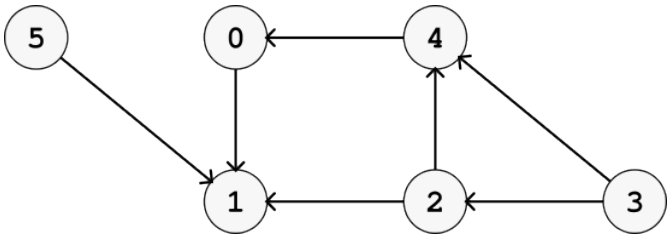


TAD Fila - FIFO (first-in first-out) - Exemplo

Exemplo: distâncias da cidade **3**

```
FILA [ 1 0 ]

mapa[origens][destinos]
     0  1  2  3  4  5
 0   0  1  0  0  0  0
 1   0  0  0  0  0  0
[2]  0 [1] 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
```

```
partidas   cidades alcançáveis
 3 ------- 3 (3 para 3 = 0 estrada)

 3 ------- 2 4 (3 para 2 = 1 estrada)
              (3 para 4 = 1 estrada)

 2 ------- 1 4 (3 para 2 = 1 estrada e
                2 para 1 = 1 estrada, portanto,
                3 para 1 = 2 estradas)
               (4 já visitada - rota ignorada)

 4 ------- 0 (3 para 4 = 1 estrada e
               4 para 0 = 1 estrada, portanto,
               3 para 0 = 2 estradas)
```

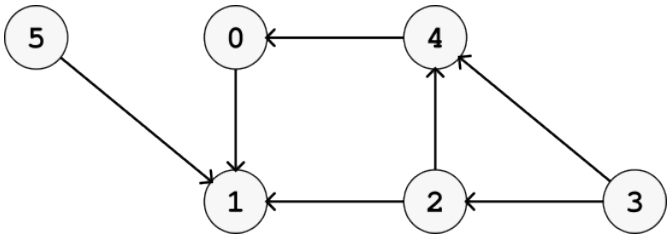


TAD Fila - FIFO (first-in first-out) - Exemplo

Exemplo: distâncias da cidade **3**

```
FILA [ 0 ]
```

```
mapa[origens][destinos]
      0   1   2   3   4   5
 0    0   1   0   0   0   0
[1]   0   0   0   0   0   0
[2]   0 [1]   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
```

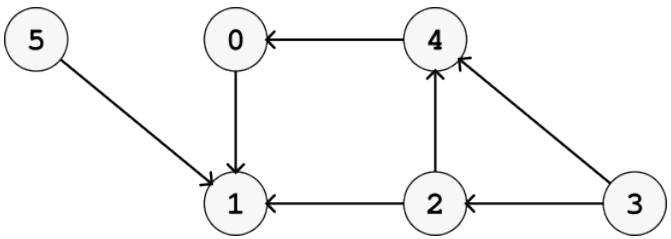



```
partidas    cidades alcançáveis
 3 -------  3 (3 para 3 = 0 estrada)

 3 -------  2 4 (3 para 2 = 1 estrada)
                (3 para 4 = 1 estrada)

 2 -------  1 4 (3 para 2 = 1 estrada e
                 2 para 1 = 1 estrada, portanto,
                 3 para 1 = 2 estradas)
                (4 já visitada - rota ignorada)

 4 -------  0 (3 para 4 = 1 estrada e
                4 para 0 = 1 estrada, portanto,
                3 para 0 = 2 estradas)

 1 -------  X
```

TAD Fila - FIFO (first-in first-out) - Exemplo

Exemplo: distâncias da cidade **3**

```
FILA [ ]

mapa[origens][destinos]
      0   1   2   3   4   5
[0]   0 [1]  0   0   0   0
[1]   0   0  0   0   0   0
[2]   0 [1]  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
```




```
partidas    cidades alcançáveis
 3 -------  3 (3 para 3 = 0 estrada)

 3 -------  2 4 (3 para 2 = 1 estrada)
                (3 para 4 = 1 estrada)

 2 -------  1 4 (3 para 2 = 1 estrada e
                 2 para 1 = 1 estrada, portanto,
                 3 para 1 = 2 estradas)
                (4 já visitada - rota ignorada)

 4 -------  0 (3 para 4 = 1 estrada e
                4 para 0 = 1 estrada, portanto,
                3 para 0 = 2 estradas)

 1 -------  X

 0 -------  1 (já visitada - rota ignorada)
```

TAD Fila - FIFO (first-in first-out) - Exemplo

1. Percorrer as cidades alcançáveis
2. Calcular as distâncias para cada cidade
3. Não revisitar cidades

TAD Fila - FIFO (first-in first-out) - Exemplo

1. Percorrer as cidades alcançáveis
   - **Fila das cidades**
   - Que possuem ligações com um ponto de partida
   - Cidades que devem ser analisadas
2. Calcular as distâncias para cada cidade
3. Não revisitar cidades

TAD Fila - FIFO (first-in first-out) - Exemplo

1. Percorrer as cidades alcançáveis
   - **Fila das cidades**
   - Que possuem ligações com um ponto de partida
   - Cidades que devem ser analisadas
2. Calcular as distâncias para cada cidade
   - **Lista das distâncias**:
     - ★ Índice: representa uma cidade
     - ★ Conteúdo: contador de distância

i	0	1	2	3	4	5
dist[i]	2	3	1	0	1	6

3. Não revisitar cidades

TAD Fila - FIFO (first-in first-out) - Exemplo

1. Percorrer as cidades alcançáveis
   - **Fila das cidades**
   - Que possuem ligações com um ponto de partida
   - Cidades que devem ser analisadas
2. Calcular as distâncias para cada cidade
   - **Lista das distâncias**:
     - ★ Índice: representa uma cidade
     - ★ Conteúdo: contador de distância

i	0	1	2	3	4	5
dist[i]	2	3	1	0	1	6

3. Não revisitar cidades
   - Inicialmente, cada cidade possue um valor inalcançável
   - "Infinito": N (rota máxima - linha reta)
   - Diferente de infinito = cidade já visitada

```
1 void distancias_do_inicio( int mapa[][N], cabeca *fila_cidades,
2                            int inicio, int *distancia ) {
```

```
1 void distancias_do_inicio( int mapa[][N], cabeca *fila_cidades,
2                            int inicio, int *distancia ) {
3     //inicializar todas as cidades como inalcançáveis
```

```
1 void distancias_do_inicio( int mapa[][N], cabeca *fila_cidades,
2                            int inicio, int *distancia ) {
3     //inicializar todas as cidades como inalcançáveis
4     for(int cidade=0; cidade<N; cidade++)
```

```
1 void distancias_do_inicio( int mapa[][N], cabeca *fila_cidades,
2                            int inicio, int *distancia ) {
3     //inicializar todas as cidades como inalcançáveis
4     for(int cidade=0; cidade<N; cidade++)
5         distancia[cidade] = N;
6
7     //colocar a primeira cidade na fila
```

```c
void distancias_do_inicio( int mapa[][N], cabeca *fila_cidades,
                           int inicio, int *distancia ) {
    //inicializar todas as cidades como inalcançáveis
    for(int cidade=0; cidade<N; cidade++)
        distancia[cidade] = N;

    //colocar a primeira cidade na fila
    enfileira(fila_cidades, inicio);

    //calcular a distância da primeira cidade
```

```c
void distancias_do_inicio( int mapa[][N], cabeca *fila_cidades,
                           int inicio, int *distancia ) {
    //inicializar todas as cidades como inalcançáveis
    for(int cidade=0; cidade<N; cidade++)
        distancia[cidade] = N;

    //colocar a primeira cidade na fila
    enfileira(fila_cidades, inicio);

    //calcular a distância da primeira cidade
    distancia[inicio] = 0;

    //percorrer cidades
```

```c
void distancias_do_inicio( int mapa[][N], cabeca *fila_cidades,
                           int inicio, int *distancia ) {
    //inicializar todas as cidades como inalcançáveis
    for(int cidade=0; cidade<N; cidade++)
        distancia[cidade] = N;

    //colocar a primeira cidade na fila
    enfileira(fila_cidades, inicio);

    //calcular a distância da primeira cidade
    distancia[inicio] = 0;

    //percorrer cidades
    while(!vazia(fila_cidades)) {

        //definir a cidade a ser analisada/descoberta
```

```c
void distancias_do_inicio( int mapa[][N], cabeca *fila_cidades,
                           int inicio, int *distancia ) {
    //inicializar todas as cidades como inalcançáveis
    for(int cidade=0; cidade<N; cidade++)
        distancia[cidade] = N;

    //colocar a primeira cidade na fila
    enfileira(fila_cidades, inicio);

    //calcular a distância da primeira cidade
    distancia[inicio] = 0;

    //percorrer cidades
    while(!vazia(fila_cidades)) {

        //definir a cidade a ser analisada/descoberta
        int partida = desenfileira(fila_cidades);

        //verificar cidades
```

```
1  void distancias_do_inicio( int mapa[][N], cabeca *fila_cidades,
2                             int inicio, int *distancia ) {
3      //inicializar todas as cidades como inalcançáveis
4      for(int cidade=0; cidade<N; cidade++)
5          distancia[cidade] = N;
6
7      //colocar a primeira cidade na fila
8      enfileira(fila_cidades, inicio);
9
10     //calcular a distância da primeira cidade
11     distancia[inicio] = 0;
12
13     //percorrer cidades
14     while(!vazia(fila_cidades)) {
15
16         //definir a cidade a ser analisada/descoberta
17         int partida = desenfileira(fila_cidades);
18
19         //verificar cidades
20         for(int cidade=0; cidade<N; cidade++)
21
22             ///se existe estrada e ainda não foi visitada
```

```
1  void distancias_do_inicio( int mapa[][N], cabeca *fila_cidades,
2                             int inicio, int *distancia ) {
3      //inicializar todas as cidades como inalcançáveis
4      for(int cidade=0; cidade<N; cidade++)
5          distancia[cidade] = N;
6
7      //colocar a primeira cidade na fila
8      enfileira(fila_cidades, inicio);
9
10     //calcular a distância da primeira cidade
11     distancia[inicio] = 0;
12
13     //percorrer cidades
14     while(!vazia(fila_cidades)) {
15
16         //definir a cidade a ser analisada/descoberta
17         int partida = desenfileira(fila_cidades);
18
19         //verificar cidades
20         for(int cidade=0; cidade<N; cidade++)
21
22             ///se existe estrada e ainda não foi visitada
23             if( mapa[partida][cidade]==1 && distancia[cidade]>=N ) {
24
25                 //calcular a distância a partir do ponto de partida
```

```c
void distancias_do_inicio( int mapa[][N], cabeca *fila_cidades,
                           int inicio, int *distancia ) {
    //inicializar todas as cidades como inalcançáveis
    for(int cidade=0; cidade<N; cidade++)
        distancia[cidade] = N;

    //colocar a primeira cidade na fila
    enfileira(fila_cidades, inicio);

    //calcular a distância da primeira cidade
    distancia[inicio] = 0;

    //percorrer cidades
    while(!vazia(fila_cidades)) {

        //definir a cidade a ser analisada/descoberta
        int partida = desenfileira(fila_cidades);

        //verificar cidades
        for(int cidade=0; cidade<N; cidade++)

            ///se existe estrada e ainda não foi visitada
            if( mapa[partida][cidade]==1 && distancia[cidade]>=N ) {

                //calcular a distância a partir do ponto de partida
                distancia[cidade] = distancia[partida] + 1;

                //definir que a cidade deve ser analisada
```

```c
void distancias_do_inicio( int mapa[][N], cabeca *fila_cidades,
                           int inicio, int *distancia ) {
    //inicializar todas as cidades como inalcançáveis
    for(int cidade=0; cidade<N; cidade++)
        distancia[cidade] = N;

    //colocar a primeira cidade na fila
    enfileira(fila_cidades, inicio);

    //calcular a distância da primeira cidade
    distancia[inicio] = 0;

    //percorrer cidades
    while(!vazia(fila_cidades)) {

        //definir a cidade a ser analisada/descoberta
        int partida = desenfileira(fila_cidades);

        //verificar cidades
        for(int cidade=0; cidade<N; cidade++)

            ///se existe estrada e ainda não foi visitada
            if( mapa[partida][cidade]==1 && distancia[cidade]>=N ) {

                //calcular a distância a partir do ponto de partida
                distancia[cidade] = distancia[partida] + 1;

                //definir que a cidade deve ser analisada
                enfileira(fila_cidades, cidade);
            }
    }
}
```

TAD Fila - FIFO (first-in first-out) - Exemplo

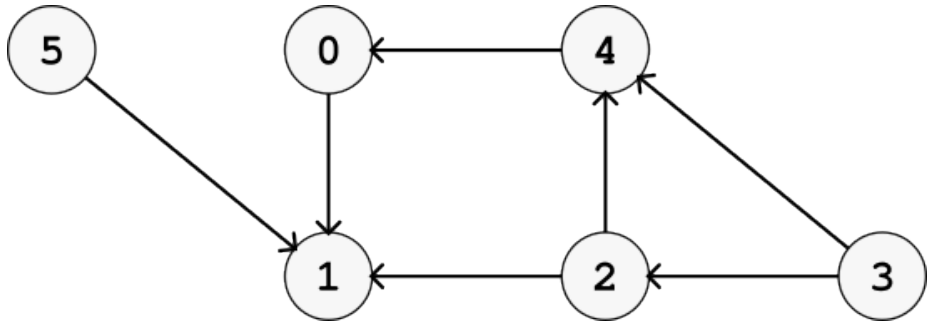
```c
#define N 6
int main(){
    cabeca *cidades = criar_lista();
    int dist[N];
    //matriz de adjacências
    int mapa[N][N] ={ {0, 1, 0, 0, 0, 0},
                      {0, 0, 0, 0, 0, 0},
                      {0, 1, 0, 0, 1, 0},
                      {0, 0, 1, 0, 1, 0},
                      {1, 0, 0, 0, 0, 0},
                      {0, 1, 0, 0, 0, 0}};

    distancias_do_inicio(mapa, cidades, 3, dist);

    printf("Distâncias:\n");
    for(int cidade=0; cidade<N; cidade++)
    {
        printf("3-%d = %d\n", cidade, dist[cidade]);
    }
    printf("\n");

    return 0;
}
```

Problema usando **TAD Pilha**

Distância das demais cidade

- Problema:
  - Dada uma cidade c
  - Encontrar a distância (menor número de estradas) de c a cada uma das demais cidades.
- Dado um mapa:
  - A[i][j] vale 1 se existe estrada de i para j e vale 0 em caso contrário
- **E se armazenar as cidades alcançáveis com uma pilha?!**

		destinos					
		0	1	2	3	4	5
origens	0	0	1	0	0	0	0
	1	0	0	0	0	0	0
	2	0	1	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



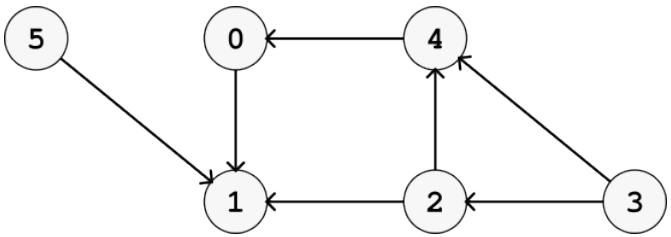
TAD Pilha - LIFO (Last-in first-out) - Exemplo

Exemplo: distâncias da cidade **3**

```
PILHA [ 3 ]
```

```
partidas   cidades alcançáveis
 3 ------- 3 (3 para 3 = 0 estrada)
```

```
mapa[origens][destinos]
      0  1  2  3  4  5
 0    0  1  0  0  0  0
 1    0  0  0  0  0  0
 2    0  1  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
```

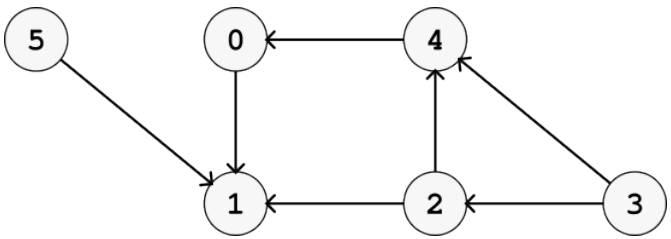


TAD Pilha - LIFO (Last-in first-out) - Exemplo

Exemplo: distâncias da cidade **3**

```
PILHA [ 2 4 ]

mapa[origens][destinos]
      0   1   2   3   4   5
 0    0   1   0   0   0   0
 1    0   0   0   0   0   0
 2    0   1   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
```

```
partidas   cidades alcançáveis
 3 -------  3 (3 para 3 = 0 estrada)

 3 -------  2 4 (3 para 2 = 1 estrada)
               (3 para 4 = 1 estrada)
```

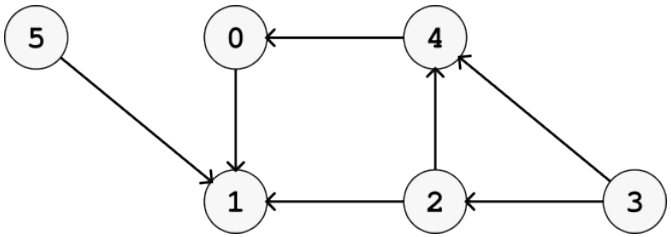


TAD Pilha - LIFO (Last-in first-out) - Exemplo

Exemplo: distâncias da cidade **3**

```
PILHA [ 2 0 ]

mapa[origens][destinos]
     0  1  2  3  4  5
 0   0  1  0  0  0  0
 1   0  0  0  0  0  0
 2   0  1  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
```

```
partidas   cidades alcançáveis
 3 ------- 3 (3 para 3 = 0 estrada)

 3 ------- 2 4 (3 para 2 = 1 estrada)
              (3 para 4 = 1 estrada)

 4 ------- 0 (3 para 4 = 1 estrada e
              4 para 0 = 1 estrada, portanto,
              3 para 0 = 2 estradas)
```

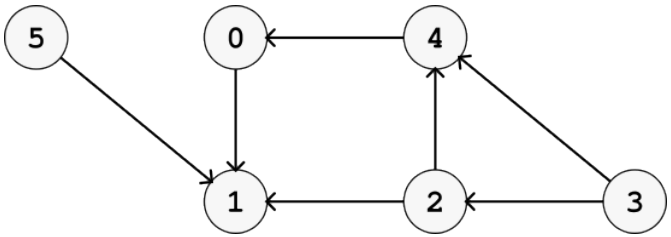


TAD Pilha - LIFO (Last-in first-out) - Exemplo

Exemplo: distâncias da cidade **3**

```
PILHA [ 2 1 ]

mapa[origens][destinos]
      0   1   2   3   4   5
[0]   0 [1]   0   0   0   0
 1    0   0   0   0   0   0
 2    0   1   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
```

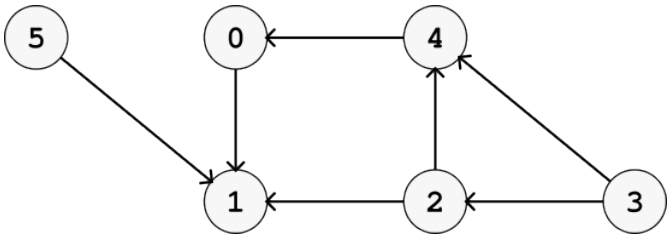
```
partidas   cidades alcançáveis
 3 ------- 3 (3 para 3 = 0 estrada)

 3 ------- 2 4 (3 para 2 = 1 estrada)
              (3 para 4 = 1 estrada)

 4 ------- 0 (3 para 4 = 1 estrada e
              4 para 0 = 1 estrada, portanto,
              3 para 0 = 2 estradas)

 0 ------- 1 (3 para 0 = 2 estrada e
              0 para 1 = 1 estrada, portanto,
              3 para 1 = 3 estradas)
```



TAD Pilha - LIFO (Last-in first-out) - Exemplo

Exemplo: distâncias da cidade **3**

```
PILHA [ 2 ]

mapa[origens][destinos]
      0   1   2   3   4   5
[0]   0 [1]  0   0   0   0
[1]   0   0   0   0   0   0
 2    0   1   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
```

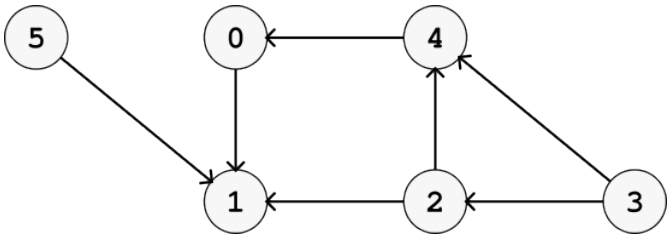



```
partidas   cidades alcançáveis
 3 ------- 3 (3 para 3 = 0 estrada)

 3 ------- 2 4 (3 para 2 = 1 estrada)
              (3 para 4 = 1 estrada)

 4 ------- 0 (3 para 4 = 1 estrada e
              4 para 0 = 1 estrada, portanto,
              3 para 0 = 2 estradas)

 0 ------- 1 (3 para 0 = 2 estrada e
              0 para 1 = 1 estrada, portanto,
              3 para 1 = 3 estradas)

 1 ------- X
```

TAD Pilha - LIFO (Last-in first-out) - Exemplo

Exemplo: distâncias da cidade **3**

```
PILHA [ ]

mapa[origens][destinos]
     0  1  2  3  4  5
[0]  0 [1] 0  0  0  0
[1]  0  0  0  0  0  0
[2]  0 [1] 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
```




```
partidas   cidades alcançáveis
 3 ------- 3 (3 para 3 = 0 estrada)

 3 ------- 2 4 (3 para 2 = 1 estrada)
              (3 para 4 = 1 estrada)

 4 ------- 0 (3 para 4 = 1 estrada e
              4 para 0 = 1 estrada, portanto,
              3 para 0 = 2 estradas)

 0 ------- 1 (3 para 0 = 2 estrada e
              0 para 1 = 1 estrada, portanto,
              3 para 1 = 3 estradas)

 1 ------- X

 2 ------- 1 4 (1 já visitada: 3 estradas
               menor caminho: 3-2-1 = 2 estradas )
              (4 já visitada - rota ignorada)
```

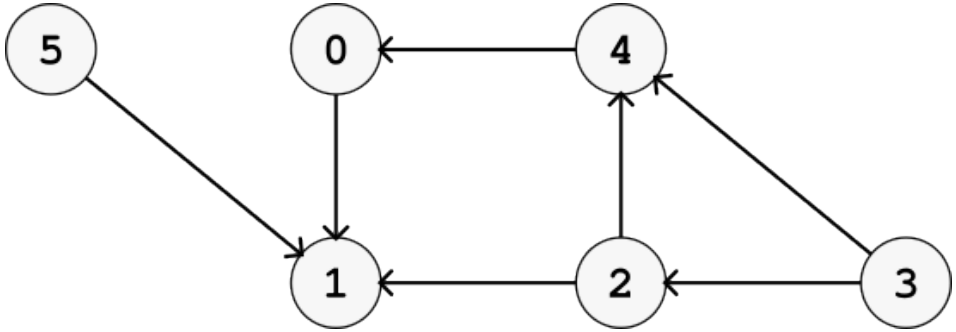
```c
//representar o mapa com qual estrutura??
cabeca ** criar_mapa()
{
    cabeca **adj = malloc(sizeof(cabeca*)*N); //???

    for(int i=0; i<N; i++) adj[i] = criar();

    int estradas[7][2] ={ {0,1},
                          {2,1}, {2,4},
                          {3,2}, {3,4},
                          {4,0},
                          {5,1} };
    for(int e=0; e<7; e++)
    {
        int i = estradas[e][0];
        int j = estradas[e][1];
        inserir_fim(adj[i], j);
    }
    return adj;
}
```

```c
int main(int argc, char *argv[]) {
    cabeca **mapa = criar_mapa();

    int inicio = 3;

    int distancia[N];
    for(int cidade=0; cidade<N; cidade++)
        distancia[cidade] = N;

    distancia[inicio] = 0;

    if(argc>1 && !strcmp(argv[1], "r"))
        percorre_rec(mapa, distancia, inicio);
    else
        percorre_ite(mapa, distancia, inicio);

    printf("\n");
    printf("Distâncias:\n");
    for(int cidade=0; cidade<N; cidade++)
        printf("3-%d = %d\n", cidade, distancia[cidade]);

    printf("\n");

    return 0;
}
```

```c
void percorre_ite(cabeca *mapa[N], int distancia[], int inicio)
{
  cabeca *pilha_cidades = criar();
  empilhar(pilha_cidades, inicio);

  while(!vazia(pilha_cidades))
  {
    inicio = desempilhar(pilha_cidades);

    for (no *a = mapa[inicio]->prox; a != NULL; a = a->prox)
    {
      if(distancia[a->info]>=N)
      {
        distancia[a->info] = distancia[inicio] + 1;
        empilhar(pilha_cidades, a->info);
      }
    }
  }
}
```

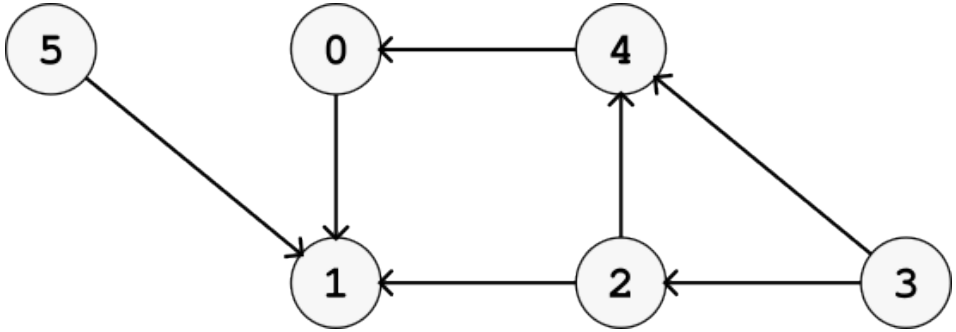
```c
void percorre_rec(cabeca *mapa[N], int distancia[], int inicio)
{
  for (no *a = mapa[inicio]->prox; a != NULL; a = a->prox)
  {
    if(distancia[a->info]>=N)
    {
      distancia[a->info] = distancia[inicio] + 1;
      percorre_rec(mapa, distancia, a->info);
    }
  }
}
```

Rastreando (trace) as chamadas da função recursiva



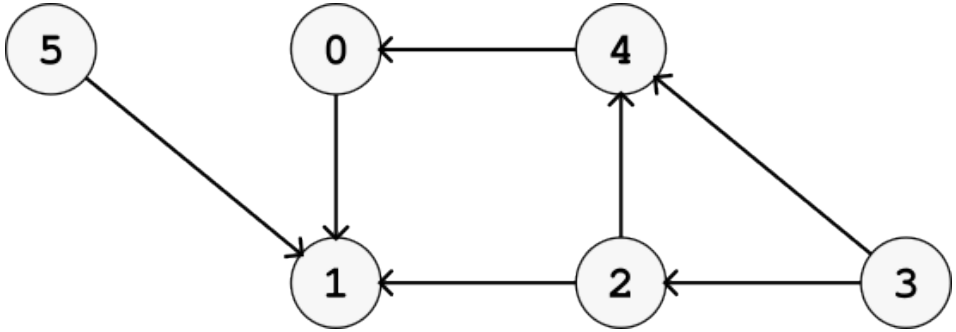

```
percorre_recursivo(3)
```

Rastreando (trace) as chamadas da função recursiva



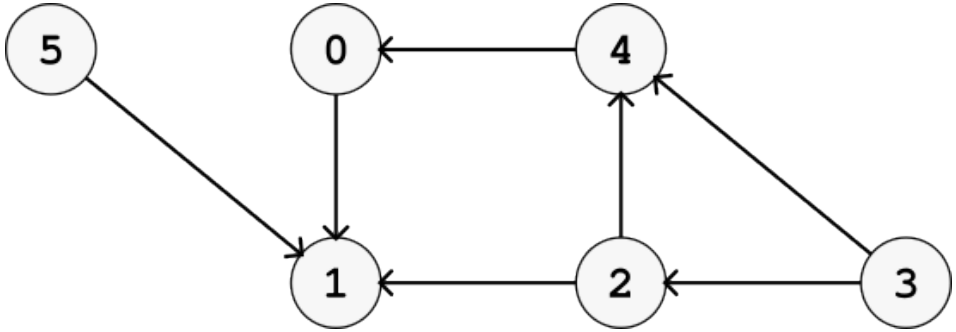

```
percorre_recursivo(3)
|______percorre_recursivo(2)
```

Rastreando (trace) as chamadas da função recursiva



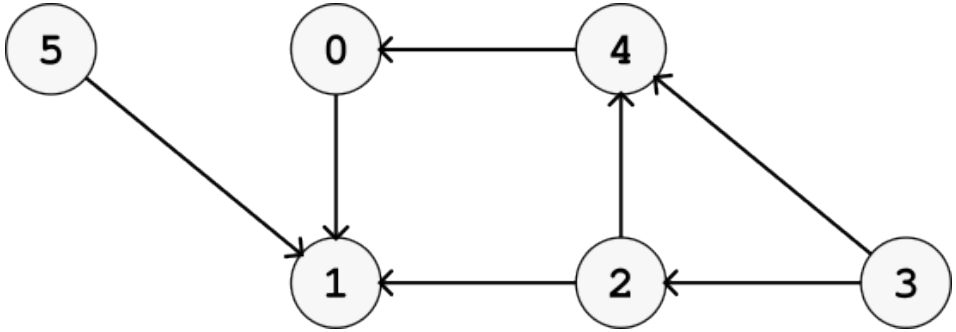

```
percorre_recursivo(3)
|______percorre_recursivo(2)
|       |______percorre_recursivo(1)
```

Rastreando (trace) as chamadas da função recursiva



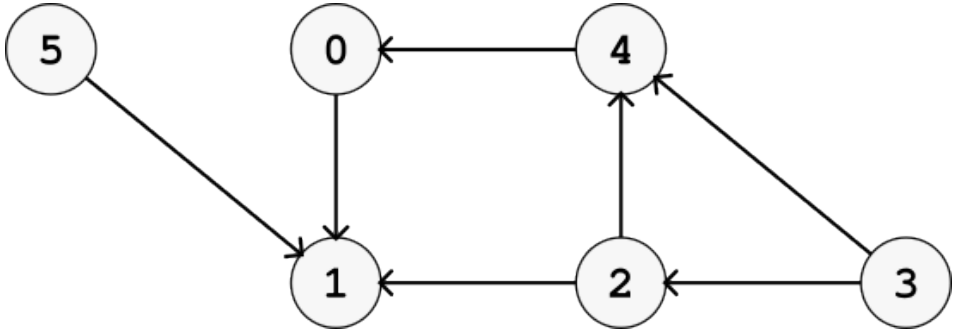

```
percorre_recursivo(3)
|______percorre_recursivo(2)
|      |______percorre_recursivo(1)
|      |      |sem mais rotas
```

Rastreando (trace) as chamadas da função recursiva



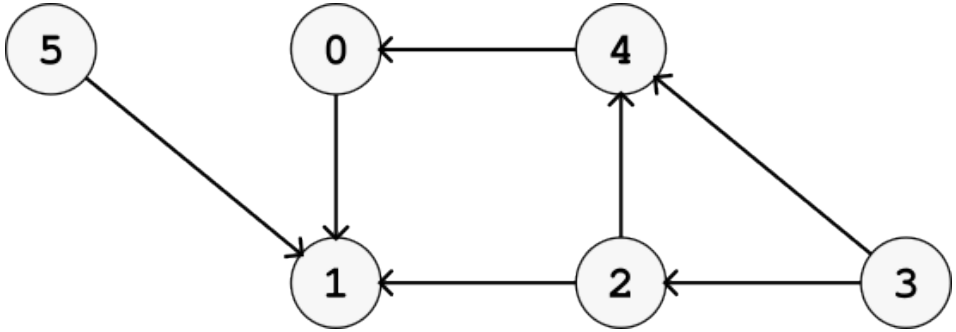

```
percorre_recursivo(3)
|______percorre_recursivo(2)
|      |______percorre_recursivo(1)
|      |      |sem mais rotas
|      |______percorre_recursivo(4)
```

Rastreando (trace) as chamadas da função recursiva



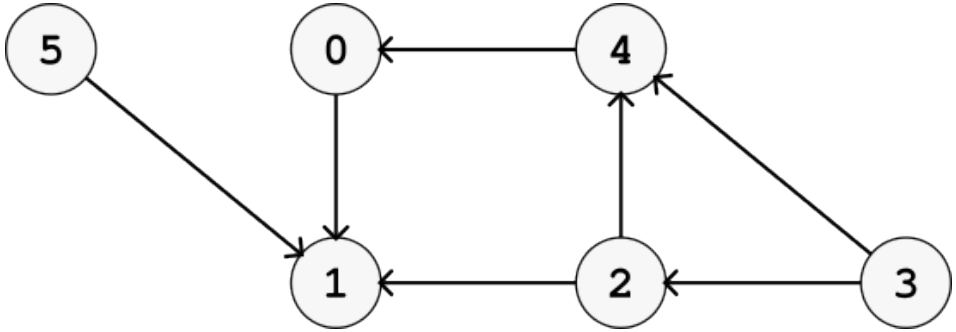

```
percorre_recursivo(3)
|______percorre_recursivo(2)
|      |______percorre_recursivo(1)
|      |      |sem mais rotas
|      |______percorre_recursivo(4)
|      |      |______percorre_recursivo(0)
```

Rastreando (trace) as chamadas da função recursiva



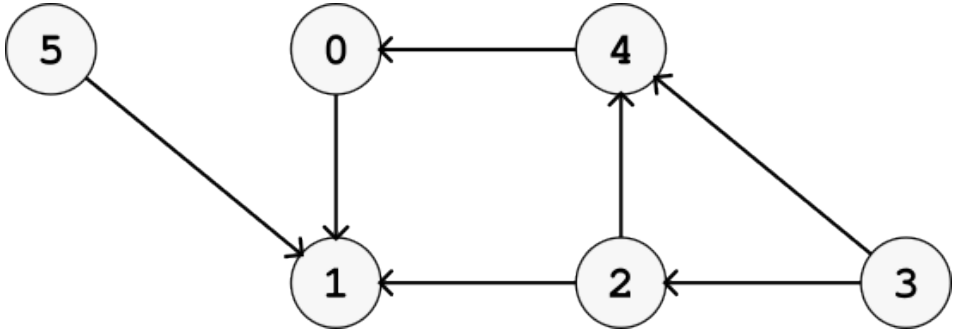

```
percorre_recursivo(3)
|______percorre_recursivo(2)
|      |______percorre_recursivo(1)
|      |      |sem mais rotas
|      |______percorre_recursivo(4)
|      |      |______percorre_recursivo(0)
|      |      |      |1 já visitado
```

Rastreando (trace) as chamadas da função recursiva



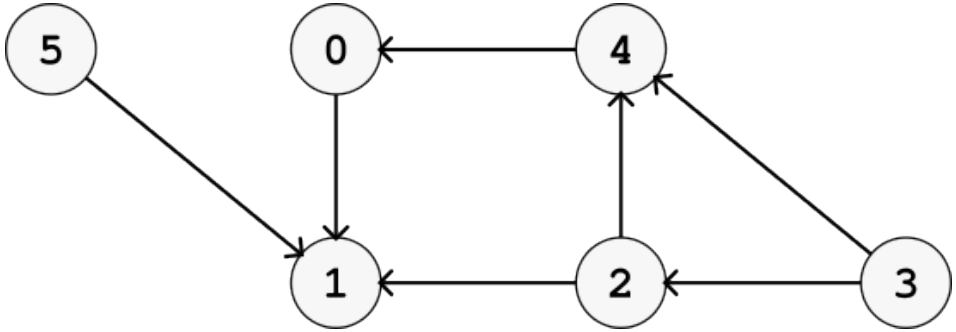

```
percorre_recursivo(3)
|______percorre_recursivo(2)
|      |______percorre_recursivo(1)
|      |      |sem mais rotas
|      |______percorre_recursivo(4)
|      |      |______percorre_recursivo(0)
|      |      |      |1 já visitado
|      |      |sem mais rotas
```

Rastreando (trace) as chamadas da função recursiva




```
percorre_recursivo(3)
|______percorre_recursivo(2)
|       |______percorre_recursivo(1)
|       |       |sem mais rotas
|       |______percorre_recursivo(4)
|       |       |______percorre_recursivo(0)
|       |       |       |1 já visitado
|       |       |sem mais rotas
|       |sem mais rotas
```

Rastreando (trace) as chamadas da função recursiva




```
percorre_recursivo(3)
|______percorre_recursivo(2)
|      |______percorre_recursivo(1)
|      |      |sem mais rotas
|      |______percorre_recursivo(4)
|      |      |______percorre_recursivo(0)
|      |      |      |1 já visitado
|      |      |sem mais rotas
|      |sem mais rotas
|4 já visitado
|sem mais rotas
```

Algoritmo Iterativo x Recursivo

```
Partida 3 -> 2-> 4
Partida 4 -> 0
Partida 0 -> 1
Partida 1
Partida 2

Distâncias:
3-0 = 2
3-1 = 3
3-2 = 1
3-3 = 0
3-4 = 1
3-5 = 6
```

```
Partida 3 -> 2
Partida 2 -> 1
Partida 1 -> 4
Partida 4 -> 0
Partida 0

Distâncias:
3-0 = 3
3-1 = 2
3-2 = 1
3-3 = 0
3-4 = 2
3-5 = 6
```

TAD Fila x Pilha

- Matrizes de adjacências: uma possível representação de grafos
- Grafos: relação(arestas) entre objetos(vértices) de um conjunto

TAD Fila x Pilha

- Matrizes de adjacências: uma possível representação de grafos
- Grafos: relação(arestas) entre objetos(vértices) de um conjunto
- Percorrer grafos com Filas
  - **Algoritmo BFS - Breadth First Search**
  - Primeiro explora vértices próximos ao início
    - O próximo vértice que será processado é o que já estava na fila
  - Aplicado em problemas como de encontrar o menor caminho

TAD Fila x Pilha

- Matrizes de adjacências: uma possível representação de grafos
- Grafos: relação(arestas) entre objetos(vértices) de um conjunto
- Percorrer grafos com Filas
  - **Algoritmo BFS - Breadth First Search**
  - Primeiro explora vértices próximos ao início
    - O próximo vértice que será processado é o que já estava na fila
  - Aplicado em problemas como de encontrar o menor caminho
- Percorrer grafos com Pilhas
  - **Algoritmo DFS - Depth First Search**
  - Primeiro explora completamente um vértice próximo ao início
    - Completamente: todos os vértices ligados
    - O próximo vértice que será processado é definido pelo processamento vigente
  - Aplicado em problemas com uma única solução (ex. método de exploração de labirintos)
  - Recursivo x Iterativo:
    - Diferem na ordem em que os vértices serão descobertos
    - Iterativo: atrasa a análise de um vértice até sua retirada da pilha
    - Recursivo: percorre o vértice assim que ele é alcançado

TAD Fila x Pilha : exemplo

(baseado: `https://www.spoj.com/problems/ELEVTRBL/`)

1. Problema dos portais:
   - Existem 't' (identificadores 1-t) tempos em uma linha temporal
   - Cada um dos t's tempos possui um portal que pula 'x' tempos para o futuro ou para o passado
   - Objetivo: se possível, chegar do tempo 's' até o 'o' passando pelo menor número de portais
   - Entradas: 't', 's', 'o' (tempos, atual, objetivo), seguidos de t's inteiros 'x'
   - Saída: quantidade mínima de portais para sair de 's' e chegar a 'o'
2. Resolução: ligações entre os tempos, percorrer
   - Largura?
   - Profundidade?
   - Recursão?

TAD Fila x Pilha : exemplo

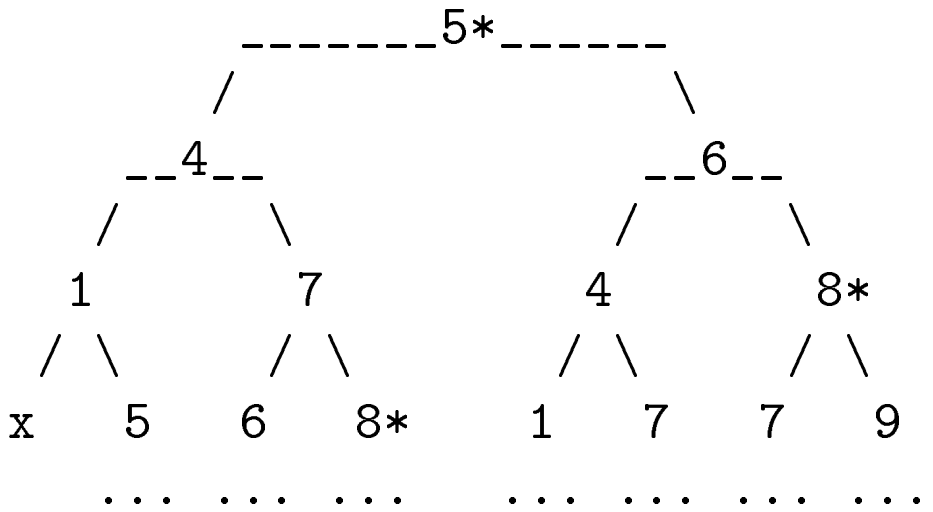
```
Entrada: 10 5 8  4 2 1 3 1 2 1 1 1 2
Saída: 2

          1  2  3  4  5  6  7  8  9 10
x[] = {?, 4, 2, 1, 3, 1, 2, 1, 1, 1, 2}

inicio 5, objetivo 8
```

	1	2	3	4	5	6	7	8	9	10
1	0	0	0	0	1	0	0	0	0	0
2	0	0	0	1	0	0	0	0	0	0
3	0	1	0	1	0	0	0	0	0	0
4	1	0	0	0	0	0	1	0	0	0
5	0	0	0	1	0	1	0	0	0	0
6	0	0	0	1	0	0	0	1	0	0
7	0	0	0	0	0	1	0	1	0	0
8	0	0	0	0	0	0	1	0	1	0
9	0	0	0	0	0	0	0	1	0	1
10	0	0	0	0	0	0	0	1	0	0

TAD Fila x Pilha : exemplo

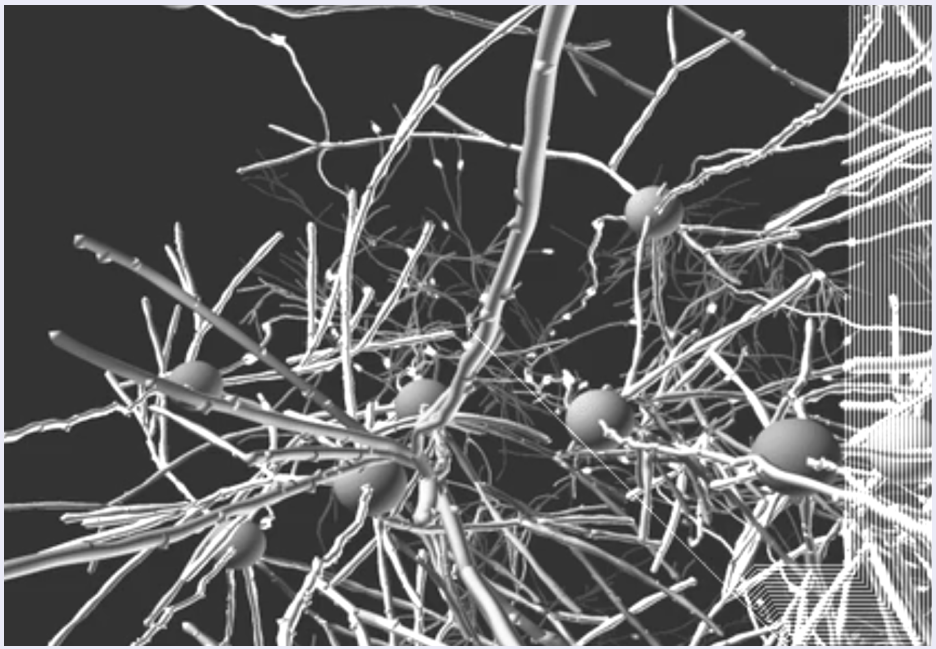
```
Entrada: 10 5 8  4 2 1 3 1 2 1 1 1 2
Saída: 2
```

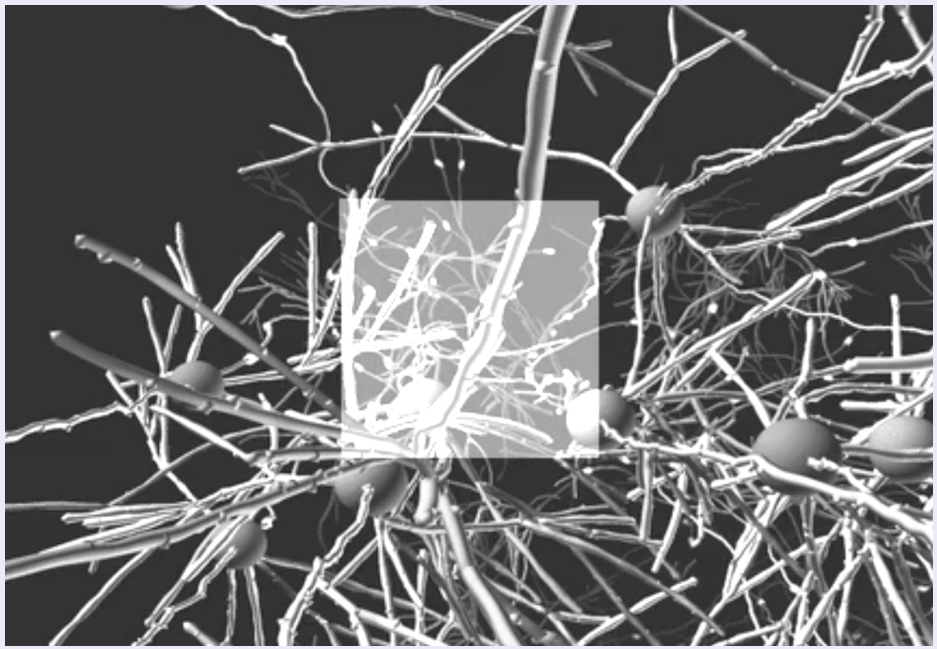
```
           1  2  3  4  5  6  7  8  9 10
x[] = {?, 4, 2, 1, 3, 1, 2, 1, 1, 1, 2}
```



Exemplo: comportamento fifo x lifo

Clareamento de imagem PGM





- Exercício proposto: `https://fga.rysh.com.br/eda1/atividade.txt`