

Tipos Abstrato de Dados

Prof^a. Rose Yuri Shimizu

1 Tipos Abstratos de Dados

- Fila
 - Implementação com lista estática
 - Implementação com lista encadeada
 - Exemplo programa cliente
- Pilha
 - Implementação com listas estáticas
 - Implementação com lista encadeada
 - Exemplo programa cliente
- Árvores Binárias

Tipos Abstratos de Dados - TAD

- TAD: generalização de tipos primitivos de dados (int, float, char)
 - ▶ Assim como funções são generalizações de operações primitivas (adição, subtração e multiplicação)
- É um **tipo de estrutura**:
 - ▶ Armazena um conjunto de dados **encapsulados** como um objeto
 - ▶ **Características** e **operações/ações** particulares
- 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

1 Tipos Abstratos de Dados

- Fila

- Implementação com lista estática
- Implementação com lista encadeada
- Exemplo programa cliente

- Pilha

- Implementação com listas estáticas
- Implementação com lista encadeada
- Exemplo programa cliente

- Árvores Binárias

TAD Fila

O que é uma FILA?

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.

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?

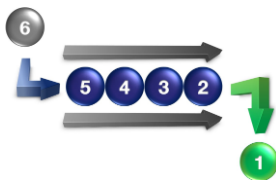
TAD Fila

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.



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
 - ▶ Transmissão de mensagens/pacotes em redes de computadores
 - ▶ Aplicações cliente x servidor (fila de requisições)
 - ▶ Fila de processos no sistema operacional
 - ▶ Gravação de mídias (ordem dos dados importa)
 - ▶ Busca: varredura pelos mais próximos primeiro

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

1 Tipos Abstratos de Dados

- Fila

- Implementação com lista estática
- Implementação com lista encadeada
- Exemplo programa cliente

- Pilha

- Implementação com listas estáticas
- Implementação com lista encadeada
- Exemplo programa cliente

- Árvores Binárias

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

fila [p..u-1] 7 posições

início da fila

fim da fila

1

□

0

1

2

3

4

5

7

8

1

1

Ro:

Rose (RYSH)

TAD

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

- CRIAÇÃO DA FILA

```
1 #define N 7
2
3 int fila[N];
4 int p, u;
5 void criar_fila ()
6 {
7     p = u = 0;
8 }
9
```

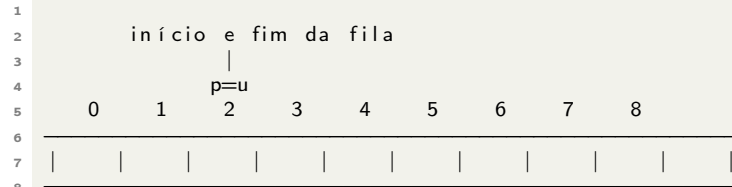
TAD Fila - FIFO (first-in first-out) - lista estática

- FILA VAZIA

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

- FILA VAZIA

```
1 #define N 7
2
3 int fila[N];
4 int p, u;
5
6 int vazia ()
7 {
8     return p == u;
9 }
10
```

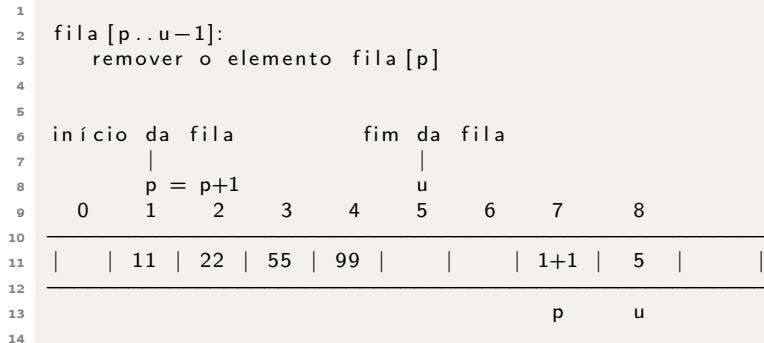


TAD Fila - FIFO (first-in first-out) - lista estática

- REMOÇÃO NO INÍCIO DA FILA - desenfileirar

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

- REMOÇÃO NO INÍCIO DA FILA - desenfilear
- Início da fila **p** é deslocado para mais próximo do fim
 - ▶ “removendo” logicamente o elemento da fila



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” logicamente 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++];
9  }
10
```

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

- INSERÇÃO NO FIM DA FILA - enfileirar

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	<hr/>									
11		11	22	55	99	88		2	5+1	
12	<hr/>									
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;
9 }
10
```

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

- Fila cheia? Como identificar?

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

- Fila cheia? Como identificar?

► $u = N$

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

- E se inserir em lista cheia?

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

- Fila cheia? Como identificar?

- ▶ $u = N$



- 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
```

- Vamos testar.

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

- Problema: e se fila cheia, $u == N$, com espaços livres?

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 }
14
```

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

lista estática circular

- Decisão: posição anterior a **p** fica vazio
- Diferenciar fila cheia e vazia
 - ▶ Fila cheia:
 - ★ $u+1 == p$ ou $(u+1 == N \text{ e } p == 0)$
 - ★ ou seja, se $(u+1) \% N == p$
 - ▶ Fila vazia: $u == p$

```
1
2  fila [p..u-1] : N = 7
3
4  fila [u] = 44 -> fila [0] = 44 -> u = u+1 -> u=1
5  fila [u] = 77 -> fila [1] = 77 (??)
6                      não pois u+1=p (fila cheia)
```

	fim		início								
		u	=	u+1=p							
	0	1	2	3	4	5	6	7	8		
12	<hr/>										
13	44		22	55	99	88	33	2	1		
14	<hr/>										
15							N-1	p	u		

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

- Vamos testar.

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

Implementação com lista estática - possibilidade de ter várias filas

```
1  typedef struct {
2      Item *item;
3      int primeiro;
4      int ultimo;
5  } Fila;
6
7  Fila *criar( int maxN ){
8      Fila *p = malloc( sizeof *p );
9      p->item = malloc( maxN * sizeof Item );
10     p->primeiro = 0;
11     p->ultimo = 0;
12
13     return p;
14 }
15
16 int vazia( Fila *f ){
17     return f->primeiro == f->ultimo;
18 }
```

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

Implementação com lista estática - possibilidade de ter várias filas

```
1  int desenfileira (Fila *f)
2  {
3      return f->item[f->primeiro++];
4  }
5
6  void enfileira (int y)
7  {
8      f->item[p->ultimo++] = y;
9  }
10
11  ...
12
13  Fila *fila1 = criar(100);
14  Fila *fila2 = criar(400);
15
```

1 Tipos Abstratos de Dados

- Fila

- Implementação com lista estática
- **Implementação com lista encadeada**
- Exemplo programa cliente

- Pilha

- Implementação com listas estáticas
- Implementação com lista encadeada
- Exemplo programa cliente

- Árvores Binárias

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

```
1  int vazia(cabeca *lista);  
2  /*  Complexidade — constante  
3      lista->prox == NULL  
4  */  
5  
6  no *primeiro(cabeca *lista);  
7  /*  Devolve o primeiro elemento da lista  
8      Elemento mais velho da fila  
9      Complexidade — constante  
10     lista->prox  
11  */  
12  
13  no *ultimo(cabeca *lista)  
14  /*  Devolve o último elemento da lista  
15      Elemento mais novo da fila  
16      Complexidade — constante  
17     lista->ultimo  
18  */  
19
```

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

```
1 void enfileira(cabeca *lista, no *novo)
2 /* Insere no fim da lista
3     Complexidade – busca pelo último → constante
4     lista → ultimo → prox = novo
5     lista continua encadeada
6 */
7
8 Item desenfileira(cabeca *lista)
9 /* Remove o elemento mais velho
10     Remove do início da fila
11     Complexidade – constante
12     lista → prox = removido → prox
13     lista continua encadeada
14 */
15
```

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

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

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

- Alternativas para quando não temos o último elemento na cabeça:
 - ▶ Lista duplamente encadeada circular: ultimo = lista->prox->ant
 - ▶ Lista simplesmente encadeada circular modificada:
 - ★ último elemento apontar para a cabeça
 - ★ utilizar a cabeça para inserir o novo conteúdo, transformando-o em um elemento da “normal” da lista
 - ★ criar uma nova cabeça
 - ▶ Lista simplesmente encadeada com cauda:
 - ★ Podemos utilizar um apontador direto para a cauda

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

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

1 Tipos Abstratos de Dados

- Fila

- Implementação com lista estática
- Implementação com lista encadeada
- Exemplo programa cliente

- Pilha

- Implementação com listas estáticas
- Implementação com lista encadeada
- Exemplo programa cliente

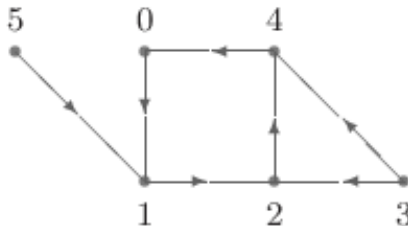
- Árvores Binárias

TAD Fila - Exemplo programa cliente

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	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



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

Exemplo: distâncias da cidade 3

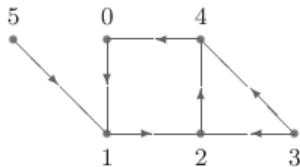
mapa[origens][destinos]

	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

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

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

2 ----- 4 (já visitada - rota ignorada)



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

Exemplo: distâncias da cidade 3

mapa[origens][destinos]

	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

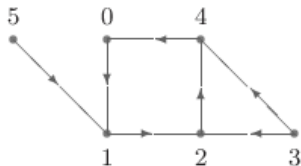
partidas cidades alcançáveis

3 ----- 3
(3 para 3 = 0 estrada)

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

2 ----- 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

mapa[origens][destinos]

	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

partidas cidades alcançáveis

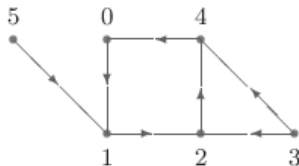
3 ----- 3
(3 para 3 = 0 estrada)

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

2 ----- 4 (já visitada - rota ignorada)

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 estradas e
0 para 1 = 1 estrada, portanto,
3 para 1 = 3 estradas)



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

Decisões:

- 1 Como armazenar as cidades alcançáveis?

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

Decisões:

- ❶ Como armazenar as cidades alcançáveis?
 - ▶ **Fila das cidades**
 - ★ Que possuem ligações com um ponto de partida
 - ★ Analisando as mais próximas primeiro
 - ▶ Utilize a interface:
 - ❶ **criar**: uma fila vazia
 - ❷ **vazia**: verificar se está vazia
 - ❸ **enfileirar**: inserir um novo item (fim da fila)
 - ❹ **desenfileirar**: remover o item mais velho (início da fila)

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

Decisões:

- ❶ Como armazenar as cidades alcançáveis?
 - ▶ **Fila das cidades**
 - ★ Que possuem ligações com um ponto de partida
 - ★ Analisando as mais próximas primeiro
 - ▶ Utilize a interface:
 - ❶ **criar**: uma fila vazia
 - ❷ **vazia**: verificar se está vazia
 - ❸ **enfileirar**: inserir um novo item (fim da fila)
 - ❹ **desenfileirar**: remover o item mais velho (início da fila)
- ❷ Como saber quantas estradas foram necessárias para chegar em uma cidade?

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

Decisões:

- ❶ Como armazenar as cidades alcançáveis?
 - ▶ **Fila das cidades**
 - ★ Que possuem ligações com um ponto de partida
 - ★ Analisando as mais próximas primeiro
 - ▶ Utilize a interface:
 - ❶ **criar**: uma fila vazia
 - ❷ **vazia**: verificar se está vazia
 - ❸ **enfileirar**: inserir um novo item (fim da fila)
 - ❹ **desenfileirar**: remover o item mais velho (início da fila)
- ❷ Como saber quantas estradas foram necessárias para chegar em uma cidade?
 - ▶ Contador para cada cidade
 - ▶ **Vetor**: cada índice é uma cidade
 - ▶ **Cada cidade na fila**:
 - ★ É um ponto novo de partida
 - ★ Que conecta mais cidades à cidade inicial

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

Decisões:

- ❶ Como armazenar as cidades alcançáveis?
 - ▶ **Fila das cidades**
 - ★ Que possuem ligações com um ponto de partida
 - ★ Analisando as mais próximas primeiro
 - ▶ Utilize a interface:
 - ❶ **criar**: uma fila vazia
 - ❷ **vazia**: verificar se está vazia
 - ❸ **enfileirar**: inserir um novo item (fim da fila)
 - ❹ **desenfileirar**: remover o item mais velho (início da fila)
- ❷ Como saber quantas estradas foram necessárias para chegar em uma cidade?
 - ▶ Contador para cada cidade
 - ▶ **Vetor**: cada índice é uma cidade
 - ▶ **Cada cidade na fila**:
 - ★ É um ponto novo de partida
 - ★ Que conecta mais cidades à cidade inicial
- ❸ Como saber se um cidade já foi visitada?

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

Decisões:

- ❶ Como armazenar as cidades alcançáveis?
 - ▶ **Fila das cidades**
 - ★ Que possuem ligações com um ponto de partida
 - ★ Analisando as mais próximas primeiro
 - ▶ Utilize a interface:
 - ❶ **criar**: uma fila vazia
 - ❷ **vazia**: verificar se está vazia
 - ❸ **enfileirar**: inserir um novo item (fim da fila)
 - ❹ **desenfileirar**: remover o item mais velho (início da fila)
- ❷ Como saber quantas estradas foram necessárias para chegar em uma cidade?
 - ▶ Contador para cada cidade
 - ▶ **Vetor**: cada índice é uma cidade
 - ▶ **Cada cidade na fila**:
 - ★ É um ponto novo de partida
 - ★ Que conecta mais cidades à cidade inicial
- ❸ Como saber se um cidade já foi visitada?
 - ▶ Um valor para cidade desconhecida ou inalcançável
 - ▶ “Infinito”: N (rota máxima - linha reta)
 - ▶ Diferente de infinito = cidade já visitada

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

```
1 void distancias_do_inicio(int mapa[][N], head *fila_cidades, int
   inicio, int *distancia)
2 {
3     for(int cidade=0; cidade<N; cidade++)
4         distancia[cidade] = N;
5
6     enfileira(fila_cidades, inicio);
7     distancia[inicio] = 0;
8     while(!vazia(fila_cidades)){
9
10         inicio = desenfileira(fila_cidades);
11
12         for(int cidade=0; cidade<N; cidade++)
13         {
14             if(mapa[inicio][cidade]==1 && distancia[cidade]>=N)
15             {
16                 distancia[cidade] = distancia[inicio] + 1;
17                 enfileira(fila_cidades, cidade);
18             }
19         }
20     }
21 }
```

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

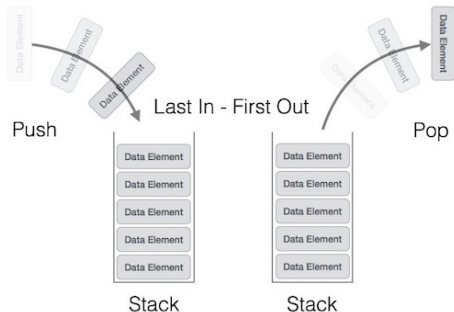
```
1 #define N 6
2 int main(){
3     head *cidades = criar_lista();
4     int dist[N];
5     int mapa[N][N] = { {0, 1, 0, 0, 0, 0},
6                        {0, 0, 1, 0, 0, 0},
7                        {0, 0, 0, 0, 1, 0},
8                        {0, 0, 1, 0, 1, 0},
9                        {1, 0, 0, 0, 0, 0},
10                       {0, 1, 0, 0, 0, 0}};
11
12     distancias_do_inicio(mapa, cidades, 3, dist);
13
14     printf("Distâncias:\n");
15     for(int cidade=0; cidade<N; cidade++){
16     {
17         printf("3-%d = %d\n", cidade, dist[cidade]);
18     }
19     printf("\n");
20
21     return 0;
22 }
23
```


1 Tipos Abstratos de Dados

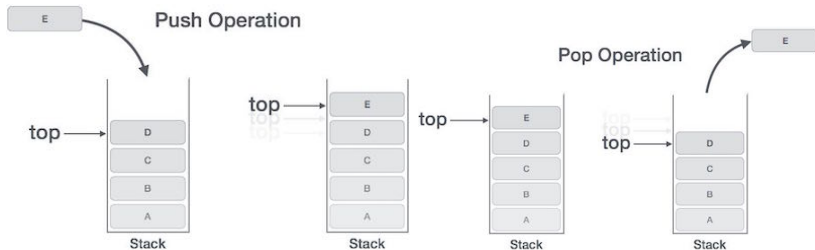
- Fila
 - Implementação com lista estática
 - Implementação com lista encadeada
 - Exemplo programa cliente
- Pilha
 - Implementação com listas estáticas
 - Implementação com lista encadeada
 - Exemplo programa cliente
- Árvores Binárias

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:
 - criar**: uma pilha vazia;
 - vazia**: verificar se está vazia;
 - empilhar**: inserir um item no topo;
 - desempilhar** remover o item mais recente;
 - 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 de sintaxe (compiladores)
- ▶ **Conversão de expressões:** infixo para prefixo, posfixo para infixo, etc.

1 Tipos Abstratos de Dados

- Fila
 - Implementação com lista estática
 - Implementação com lista encadeada
 - Exemplo programa cliente
- Pilha
 - **Implementação com listas estáticas**
 - Implementação com lista encadeada
 - Exemplo programa cliente
- Árvores Binárias

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

1	pilha [0..t-1] \rightarrow tamanho 7									
2										
3	topo da pilha									
4										
5	t									
6	0	1	2	3	4	5	6	7	8	
7	<hr/>									
8	10	11	22	55	99			5		
9	<hr/>									
10	t									
11										

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 *pilha);  
7      t=0;  
8  }  
9
```


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” logicamente 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     atualização topo t=t-1
5
6
7     topo da pilha
8         |
9         t-1 <- t
10
11     0      1      2      3      4      5      6      7      8
12  +-----+-----+-----+-----+-----+-----+-----+-----+
13  | 10 | 11 | 22 | 55 | 99 |   |   | 5-1 |   |   |
14  +-----+-----+-----+-----+-----+-----+-----+
15                                     t
```

TAD Pilha - LIFO (Last In, First Out)

listas estáticas

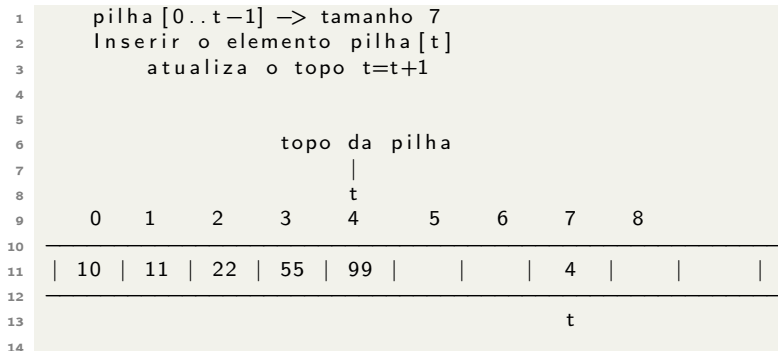
- REMOÇÃO NO TOPO DA FILA - desempilhar
- Topo da pilha **t** é deslocado para mais próximo do início
 - ▶ “removendo” logicamente o elemento da pilha
 - ▶ Item indicado pela nova posição do topo é ignorado

```
1  int *pilha ;
2  int t ;
3
4  Item desempilha ()
5  {
6      return pilha[--t];
7  }
8
```

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



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  }  
8
```

TAD Pilha - LIFO (Last In, First Out)

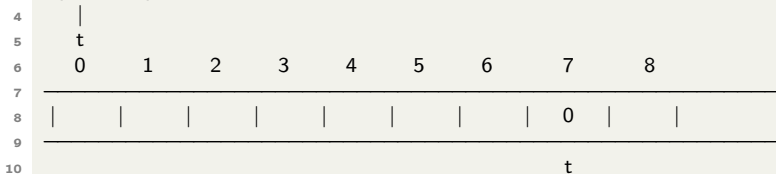
listas estáticas

● 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

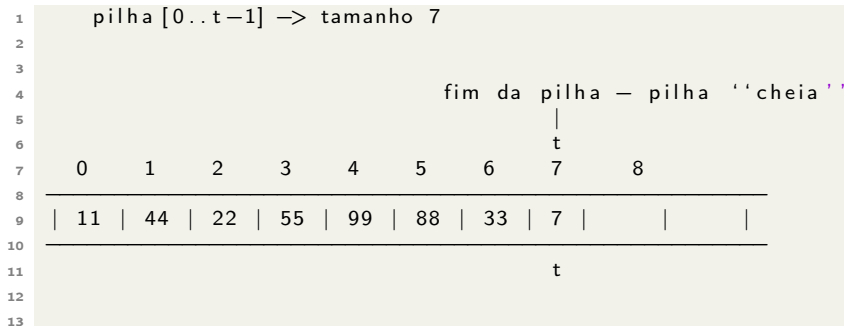
2
3 topo da pilha



TAD Pilha - LIFO (Last In, First Out)

listas estáticas

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



TAD Pilha - LIFO (Last In, First Out)

listas estáticas

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  typedef char Item;
5
6  /*****
7  /* Implementacao com array */
8  /* Varias pilhas          */
9  *****/
10 typedef struct pilha_t Pilha;
11 struct pilha_t {
12     Item *item;
13     int topo;
14 };
15
16 Pilha *criar( int maxN ){
17     Pilha *p = malloc(sizeof *p);
18     p->item = malloc(maxN*sizeof Item);
19     p->topo = 0;
20     return p;
21 }
```

TAD Pilha - LIFO (Last In, First Out)

listas estáticas

```
1  int vazia( Pilha *p )
2  {
3      return p->topo == 0;
4  }
5
6  void empilhar( Pilha *p, Item item )
7  {
8      p->item[p->topo++] = item;
9  }
10
11 Item desempilhar( Pilha *p )
12 {
13     return p->item[--p->topo];
14 }
15
16 Item espiar( Pilha *p )
17 {
18     return p->item[p->topo - 1];
19 }
20
```


1 Tipos Abstratos de Dados

- Fila
 - Implementação com lista estática
 - Implementação com lista encadeada
 - Exemplo programa cliente
- Pilha
 - Implementação com listas estáticas
 - **Implementação com lista encadeada**
 - Exemplo programa cliente
- Árvores Binárias

```
1  /*****  
2  /* Implementacao com lista encadeada */  
3  *****/  
4  typedef int Item;  
5  
6  typedef struct registro node;  
7  struct registro {  
8      Item info;  
9      node *prox;  
10 };  
11  
12 typedef struct cabeca head;  
13 struct cabeca {  
14     int num_itens;  
15     node *prox;  
16     node *ultimo;  
17 };
```

```

1  head * criar_pilha()
2  {
3      head *le = malloc(sizeof(head));
4      le->num_itens = 0;
5      le->prox = NULL;
6      le->ultimo = NULL;
7      return le;
8  }
9
10
11 node *criar_no(Item x)
12 {
13     node *no = malloc(sizeof(node));
14     no->prox = NULL;
15     no->info = x;
16     return no;
17 }
18
19 int vazia(head *p)
20 {
21     return (p->prox==NULL);
22 }
23
24 Item espia(head *p)
25 {
26     return (p->prox->info);
27 }

```

```
1 //EMPILHA NO TOPO – TOPO???
```

```
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         novo->prox = lista->prox;
```

```
8         lista->prox = novo;
```

```
9         lista->num_itens++;
```

```
10     }
```

```
11 }
```

```
12 //DESEMPILHA DO TOPO – TOPO???
```

```
13 Item desempilha(head *lista)
```

```
14 {
```

```
15     node *topo = lista->prox;
```

```
16     lista->prox = topo->prox;
```

```
17     if(topo == lista->ultimo) lista->ultimo = NULL;
```

```
18     lista->num_itens--;
```

```
19     Item x = topo->info;
```

```
20     free(topo);
```

```
21     return x;
```

```
22 }
```

TAD Pilha - 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 + *"

```
1 int main (int argc, char *argv[]) {
2     char *a = argv[1];
3
4     head *pilha = criar_lista();
5
6     for(int i=0; a[i]!='\0'; i++) {
7
8         //operacao do operador sobre os ultimos operandos lidos
9         if(a[i] == '+')
10             empilha(pilha, desempilha(pilha)+desempilha(pilha));
11         if(a[i] == '*')
12             empilha(pilha, desempilha(pilha)*desempilha(pilha));
13
14         //colocar zero a esquerda
15         if((a[i] >= '0') && (a[i] <= '9')) empilha(pilha, 0);
16
17         //calcular o equivalente numerico de uma
18         // sequencia de caracteres
19         while((a[i] >= '0') && (a[i] <= '9'))
20             //calcula o decimal, centena ... + valor numerico
21             empilha(pilha, 10*desempilha(pilha) + (a[i++] - '0'));
22     }
23     printf("%d \n", desempilha(pilha));
24 }
```

1 Tipos Abstratos de Dados

- Fila
 - Implementação com lista estática
 - Implementação com lista encadeada
 - Exemplo programa cliente
- Pilha
 - Implementação com listas estáticas
 - Implementação com lista encadeada
 - Exemplo programa cliente
- Árvores Binárias

TAD Pilha - Exemplo programa cliente

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

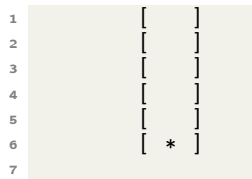
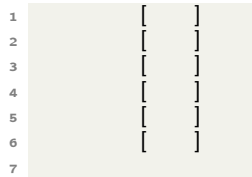
Mas não sei implementar Pilha! Mas tem a interface

- 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.

Entrada: 6 *b/i/*

b/i/

```
1  int  n;  
2  char c;  
3  scanf("%d", &n); //tamanho da expressão  
4  
5  pilha *p = criar(); //criamos a pilha  
6  
7  while(n>0 && scanf(" %c",&c)==1) {  
8      //achou o simbolo  
9      if(c=='*' || c=='/' || c=='_')  
10         {  
11             if(!vazia(p) && espiar(p) == c)  
12                 desempilhar();  
13             else  
14                 empilhar(c); //pilha vazia -> empilha *  
15         }  
16         n--;  
17     }  
18  
19     if(vazia())  
20         printf("C\n");  
21     else  
22         printf("E\n");  
23
```



Entrada: 6 *b/i/*

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

b/i/

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

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

Entrada: 6 *b/i/*

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

b/i/

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

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

Entrada: 6 *b/i/*

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

b/i/

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

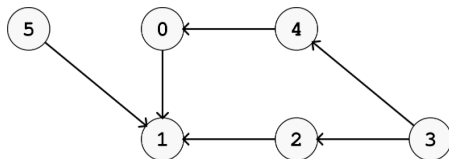
1 []
2 []
3 []
4 []
5 []
6 []
7

Exemplo: comportamento fifo x lifo

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 ao invés de uma fila?!

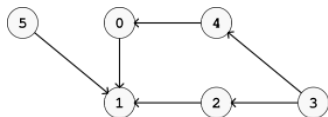
		destinos					
		0	1	2	3	4	5
origens	0	0	1	0	0	0	0
	1	0	0	0	0	0	1
	2	0	1	0	0	0	0
	3	0	0	1	0	1	0
	4	1	0	0	0	0	0
	5	0	0	0	0	0	0



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

Exemplo: distâncias da cidade 3

mapa[origens][destinos]							partidas	idades alcançáveis
0	0	1	0	0	0	0	3 -----	3
1	0	0	0	0	0	0		(3 para 3 = 0 estrada)
2	0	1	0	0	0	0		
[3]	0	0	1	0	1	0		
4	1	0	0	0	0	0		
5	0	1	0	0	0	0		

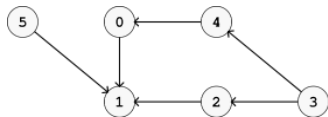


TAD Pilha - LIFO (Last-in first-out) - Exemplo

Exemplo: distâncias da cidade 3

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	0	0
[3]	0	0	1	0	1	0
4	1	0	0	0	0	0
5	0	0	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)

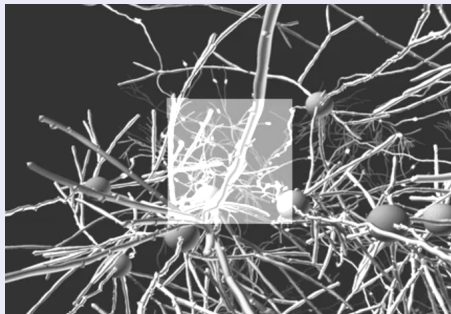
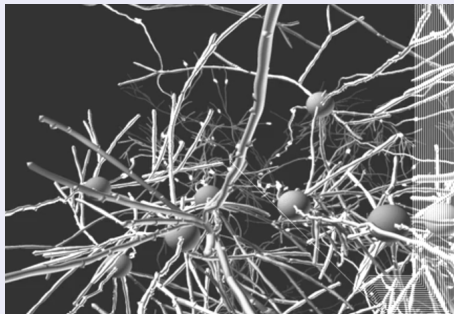
4 ----- 2 0
(3 para 4 = 1 estrada e
4 para 0 = 1 estrada, portanto,
3 para 0 = 2 estradas)

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

1 ----- X

Exemplo: comportamento fifo x lifo

Clareamento de imagem PGM



1 Tipos Abstratos de Dados

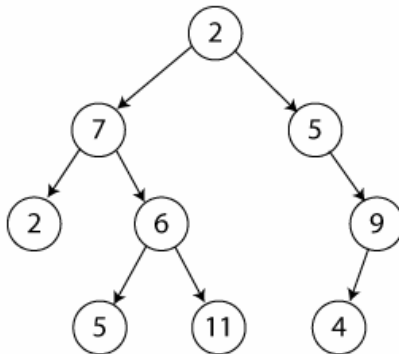
- Fila
 - Implementação com lista estática
 - Implementação com lista encadeada
 - Exemplo programa cliente
- Pilha
 - Implementação com listas estáticas
 - Implementação com lista encadeada
 - Exemplo programa cliente
- Árvores Binárias

Referências

- <https://www.ime.usp.br/~pf/algoritmos/aulas/bint.html>
- <https://www.ime.usp.br/~song/mac5710/slides/05tree>
- https://pt.wikipedia.org/wiki/%C3%81rvore_bin%C3%A1ria
- https://ww2.inf.ufg.br/~hebert/disc/aed1/AED1_10_Arvores.pdf
- <https://www.ic.unicamp.br/~rafael/cursos/2s2018/mc202/slides/unidade17-arvores-binarias.pdf>

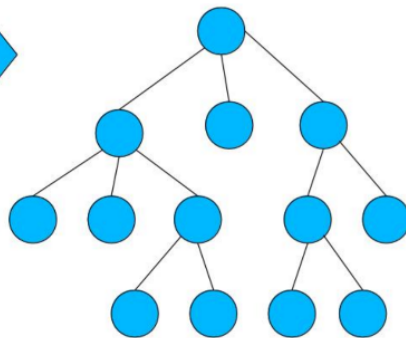
Árvore

- É uma estrutura de dados



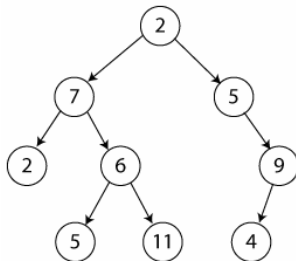
Árvores

- É uma estrutura de dados
- Armazena um conjunto de dados com relações hierárquicas



Árvores binárias

- É um grafo:
 - ▶ Estuda a relação entre objetos para a obtenção de informações
 - ★ Objetos são chamados de **vértices/nós** e as relações de **arestas**



- ▶ Acíclico, conexo, dirigido e que cada nó não tem grau maior que 2
- ▶ Só existe um caminho entre dois nós distintos

Árvores binárias - exemplo

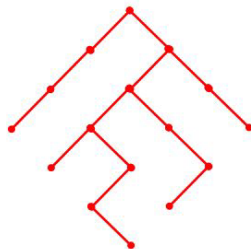
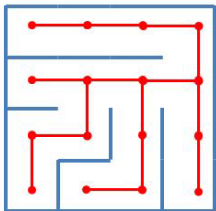


Figura: <https://openhome.cc/eGossip/OpenSCAD/SimpleGeneratedMaze.html>

Árvores binárias

- <https://www.ic.unicamp.br/~rafael/cursos/2s2018/mc202/slides/unidade17-arvores-binarias.pdf>