Tipos Abstrato de Dados

Profª. Rose Yuri Shimizu

Roteiro

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

Roteiro

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?

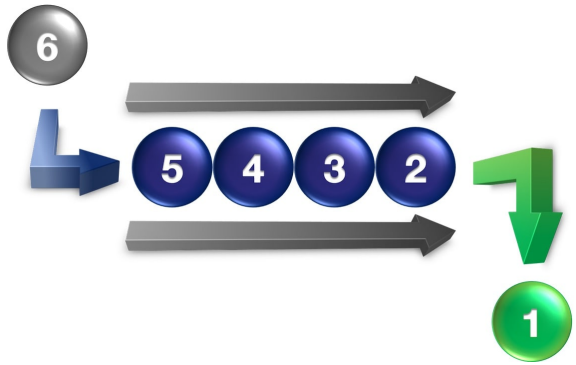
TAD Fila

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.



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
  - 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

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

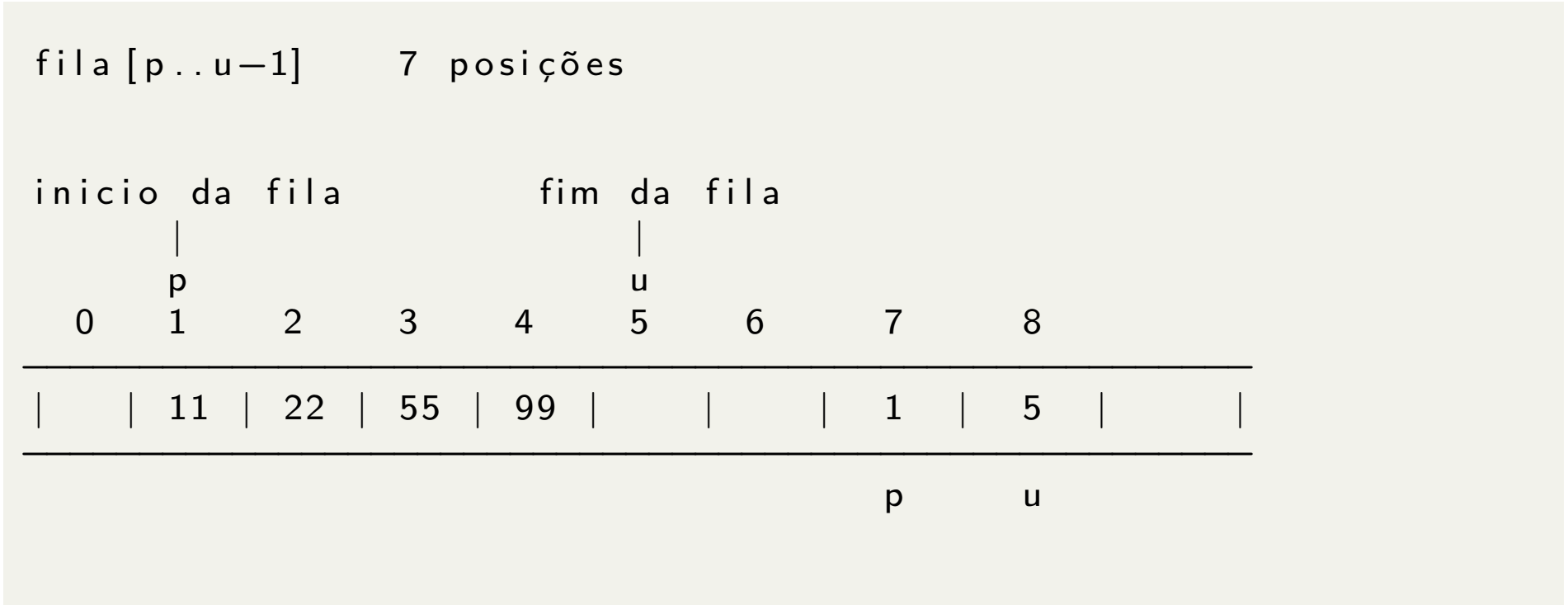
Roteiro

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



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
```

```
1
2          início e fim da fila
3                  |
4                 p=u
5      0     1     2     3     4     5     6     7     8
6  ____________________________________________________________
7  |     |     |     |     |     |     |     |     |     |     |
8  ____________________________________________________________
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 - desenfileirar
- Início da fila **p** é deslocado para mais próximo do fim
  - "removendo" logicamente 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
14
```

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

```
fila[p..u-1]:
   inserir o elemento fila[u] = 88


      início da fila       fim da fila
            |                  |
            p                  u = u+1
  0    1    2    3    4    5    6    7    8
-------------------------------------------------
|   | 11 | 22 | 55 | 99 | 88 |   |  2 | 5+1 |   |
-------------------------------------------------
                                   p    u
```

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

```
fila[p..u-1] : N = 7


     início da fila                    fim da fila - "fila cheia"
            |                                |
            p                                u
   0   1    2    3    4    5    6    7    8
 ---------------------------------------------------------
 |   |    | 22 | 55 | 99 | 88 | 33 |  2 |  7 |    |
 ---------------------------------------------------------
                                N-1   p    u
```

- E se inserir em lista cheia?

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

- Fila cheia? Como identificar?
  - u = N

```
fila[p..u-1] : N = 7


      início da fila                          fim da fila - "fila cheia"
              |                                       |
              p                                       u
   0    1     2     3     4     5     6     7      8
 ---------------------------------------------------------------------
|    |     | 22  | 55  | 99  | 88  | 33  |  2  |  7  |     |
 ---------------------------------------------------------------------
                                     N-1    p     u
```

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

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

- Inserção em fila cheia
- Transbordamento

```
fila[p .. u-1] : N = 7

fila[u] = 4
fila[7] = 4, como 7 > N-1, ocorre transbordamento


      início da fila                          fim da fila
             |                                    |
             p                                   u  =  u+1
  0     1     2     3     4     5     6          7        8
-------------------------------------------------------------------------
|     |     | 22  | 55  | 99  | 88  | 33  | 2 -> 4  | 7+1 |         |
-------------------------------------------------------------------------
                                    N-1     p          u
                                            |
                                   invadiu a área de p
```

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 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)
7
8              fim     início
9               |        |
10              u   = u+1=p
11     0        1       2      3      4     5      6      7      8
12    ------------------------------------------------------------------
13    | 44    |       | 22  | 55  | 99  | 88  | 33  | 2  |  1   |       |
14    ------------------------------------------------------------------
15                                              N-1    p      u
16
```

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
```

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

```
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
```

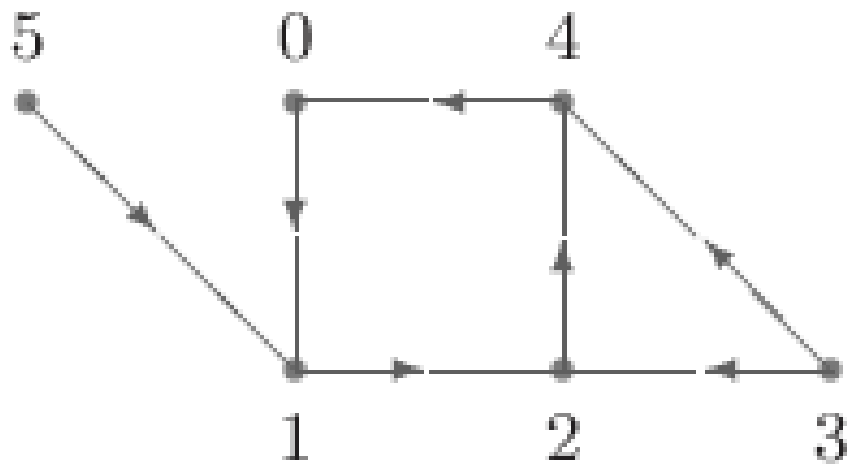
Roteiro

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

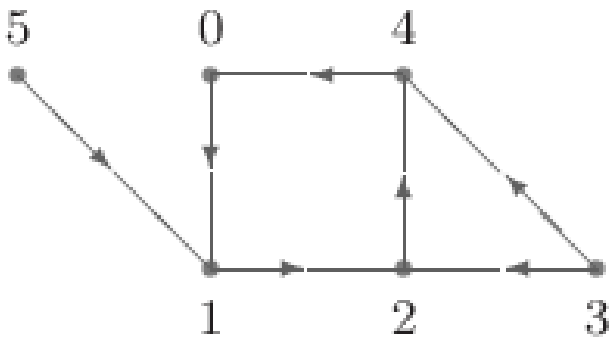


TAD Fila - FIFO (first-in first-out) - Exemplo

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)
```



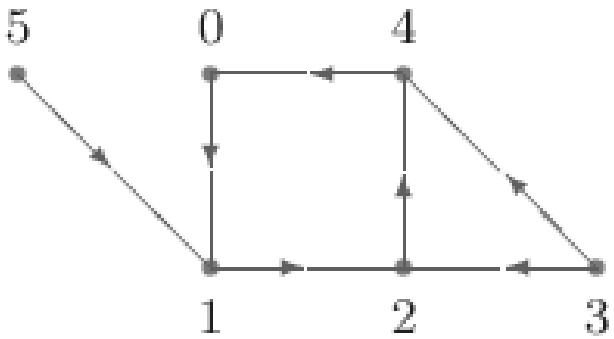
TAD Fila - FIFO (first-in first-out) - Exemplo

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)
```



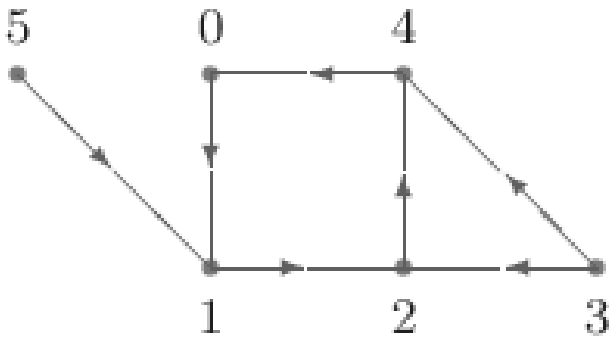
TAD Fila - FIFO (first-in first-out) - Exemplo

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)
```

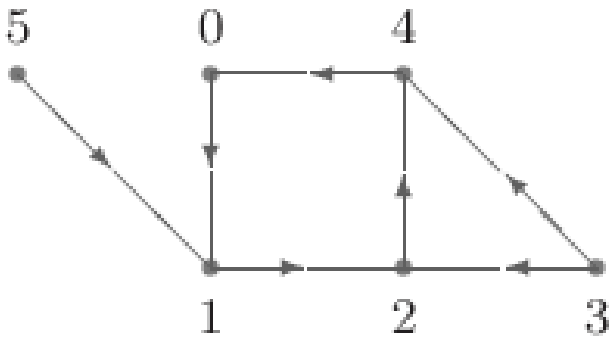


TAD Fila - FIFO (first-in first-out) - Exemplo

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



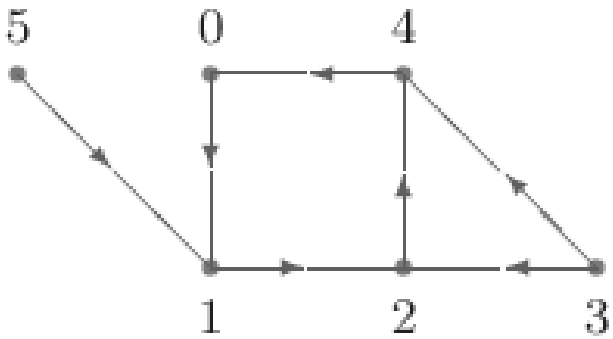

```
partidas    cidades alcançáveis
 3 -------  3
            (3 para 3 = 0 estrada)

 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:

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

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

Decisões:

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

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

Decisões:

1. Como armazenar as cidades alcançáveis?
   - **Fila das cidades**
     - ★ Que possuem ligações com um ponto de partida
     - ★ Analisando as mais próximas primeiro
   - Utilize a interface:
     1. **criar**: uma fila vazia
     2. **vazia**: verificar se está vazia
     3. **enfileirar**: inserir um novo item (fim da fila)
     4. **desenfileirar**: remover o item mais velho (início da fila)
2. 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:

1. Como armazenar as cidades alcançáveis?
   - **Fila das cidades**
     - ★ Que possuem ligações com um ponto de partida
     - ★ Analisando as mais próximas primeiro
   - Utilize a interface:
     1. **criar**: uma fila vazia
     2. **vazia**: verificar se está vazia
     3. **enfileirar**: inserir um novo item (fim da fila)
     4. **desenfileirar**: remover o item mais velho (início da fila)
2. 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
3. Como saber se um cidade já foi visitada?

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

Decisões:

1. Como armazenar as cidades alcançáveis?
   - **Fila das cidades**
     - ★ Que possuem ligações com um ponto de partida
     - ★ Analisando as mais próximas primeiro
   - Utilize a interface:
     1. **criar**: uma fila vazia
     2. **vazia**: verificar se está vazia
     3. **enfileirar**: inserir um novo item (fim da fila)
     4. **desenfileirar**: remover o item mais velho (início da fila)
2. 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
3. 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 }
22
```

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

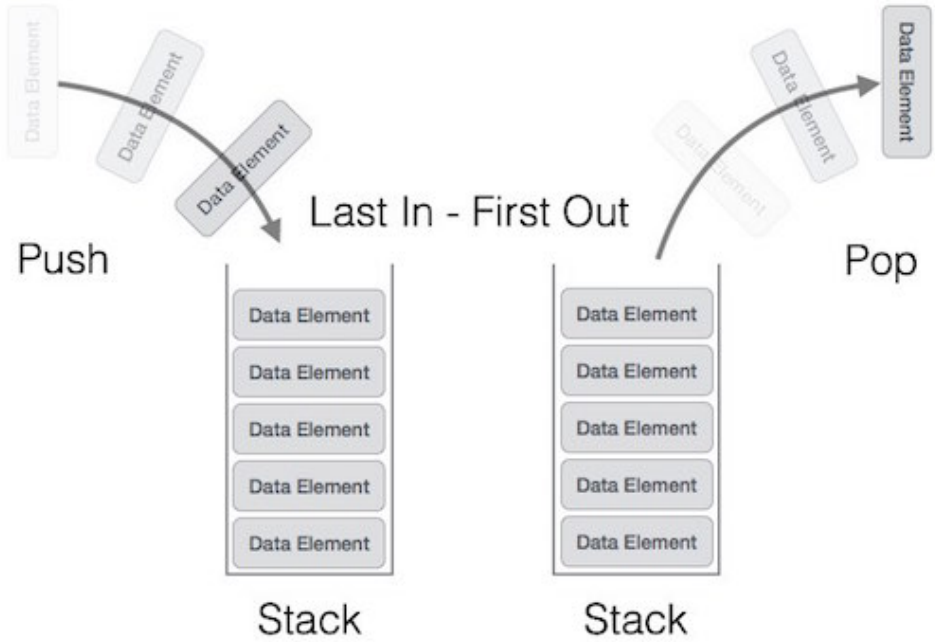
```c
#define N 6
int main(){
    head *cidades = criar_lista();
    int dist[N];
    int mapa[N][N] ={ {0, 1, 0, 0, 0, 0},
                      {0, 0, 1, 0, 0, 0},
                      {0, 0, 0, 0, 1, 0},
                      {0, 0, 1, 0, 1, 0},
                      {1, 0, 0, 0, 0, 0},
         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

```

Roteiro

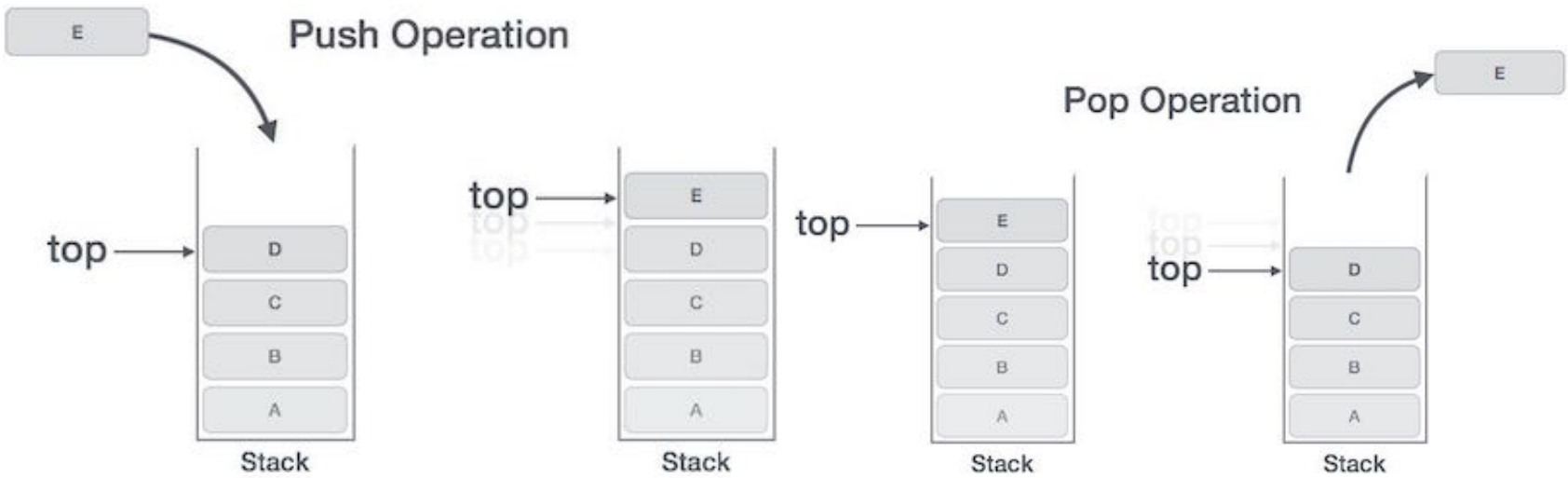
TAD Pilha - LIFO (Last In, First Out)

- 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 de sintaxe (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

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

                          topo da pilha
                                |
                                t
    0    1     2     3     4     5     6     7     8
 ---------------------------------------------------------
| 10 | 11 | 22 | 55 | 99 |     |     | 5   |     |     |
 ---------------------------------------------------------
                                      t
```

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 do 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" 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

```
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
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  }
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
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)

listas estáticas

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

typedef char Item;

/*****************************/
/* Implementacao com array */
/* Varias pilhas             */
/*****************************/
typedef struct pilha_t Pilha;
struct pilha_t {
    Item *item;
    int topo;
};

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

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
```

Roteiro

```c
/*************************************/
/* Implementacao com lista encadeada */
/*************************************/
typedef int Item;

typedef struct registro node;
struct registro {
    Item info;
    node *prox;
};

typedef struct cabeca head;
struct cabeca {
    int num_itens;
    node *prox;
    node *ultimo;
};
```

```
1
2   head * criar_pilha()
3   {
4       head *le = malloc(sizeof(head));
5       le->num_itens = 0;
6       le->prox = NULL;
7       le->ultimo = NULL;
8       return le;
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
2   //EMPILHA NO TOPO - TOPO???
3   void empilha(head *lista, Item x)
4   {
5       node *novo = criar_no(x);
6       if(novo){
7           if(vazia(lista)) lista->ultimo = novo;
8
9           novo->prox = lista->prox;
10          lista->prox = novo;
11
12          lista->num_itens++;
13      }
14  }
15  //DESEMPILHA DO TOPO - TOPO???
16  Item desempilha(head *lista)
17  {
18      node *topo = lista->prox;
19      lista->prox = topo->prox;
20
21      if(topo == lista->ultimo) lista->ultimo = NULL;
22      lista->num_itens--;
23
24      Item x = topo->info;
25      free(topo);
26      return x;
27  }
```

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 + *"

```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));
}
```

Roteiro

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/*

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

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

while(n>0 && scanf(" %c",&c)==1) {
  //achou o simbolo
  if(c=='*'||c=='/'||c=='_')
  {
    if(!vazia(p) && espiar(p) == c)
      desempilhar();
    else
      empilhar(c); //pilha vazia -> empilha *
  }
  n--;
}

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

```

b/i/

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

```
[   ]
[   ]
[   ]
[   ]
[   ]
[ * ]
```

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

b/i/

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

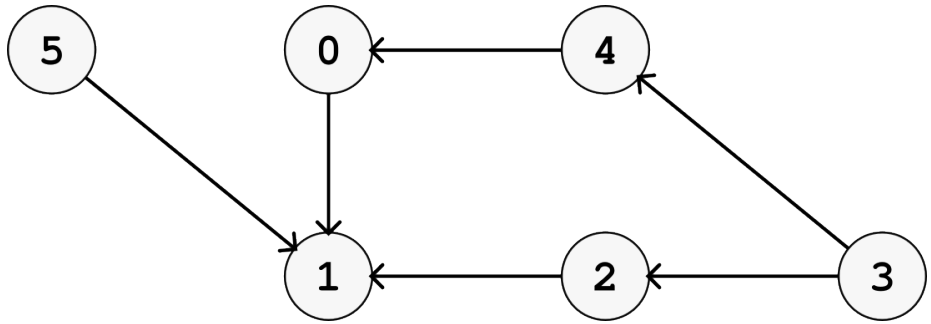
```
1   [     ]
2   [     ]
3   [     ]
4   [     ]
5   [     ]
6   [     ]
7
```

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

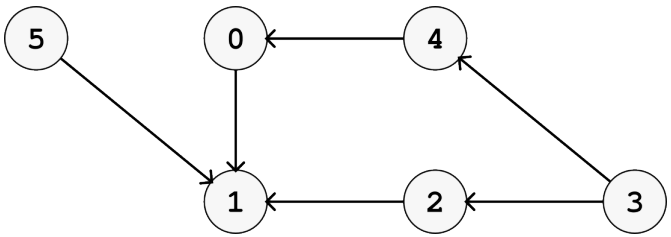


TAD Pilha - LIFO (Last-in first-out) - Exemplo

Exemplo: distâncias da cidade **3**

```
mapa[origens][destinos]
 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  1  0  0  0  0
```

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

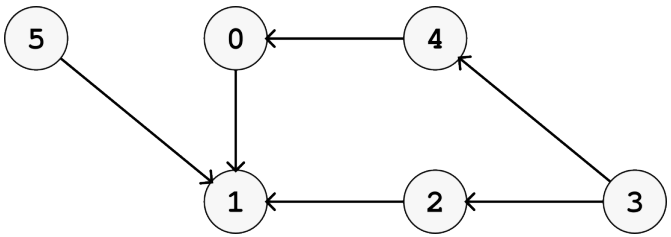


TAD Pilha - LIFO (Last-in first-out) - Exemplo

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   1   0   0   0   0
```

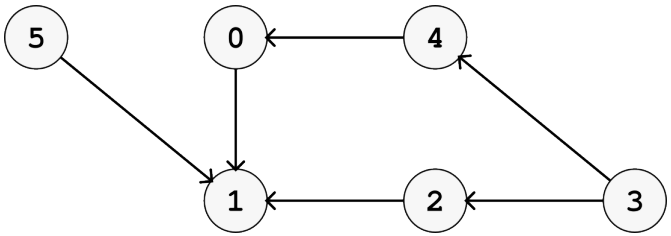
```
partidas   cidades alcançáveis
 3 -------  3
            (3 para 3 = 0 estrada)

 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**

```
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   1   0   0   0   0
```



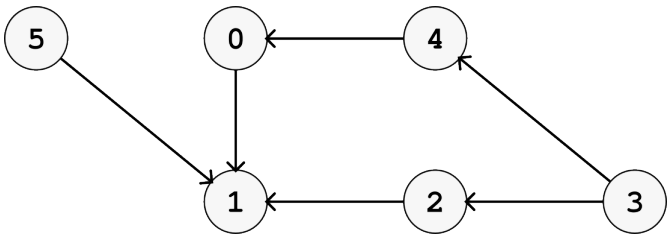

```
partidas   cidades alcançáveis
 3 -------- 3
            (3 para 3 = 0 estrada)

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

 4 -------- 2 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**

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	1	0	0	0	0

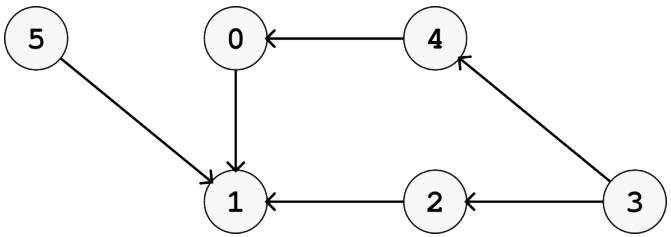



```
partidas   cidades alcançáveis
 3 ------- 3
           (3 para 3 = 0 estrada)

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

 4 ------- 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)
```

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
```



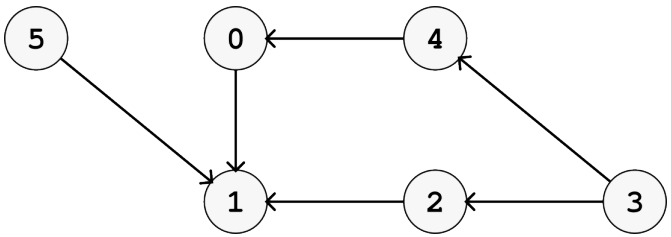

```
partidas   cidades alcançáveis
 3 ------- 3
           (3 para 3 = 0 estrada)

 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
```

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



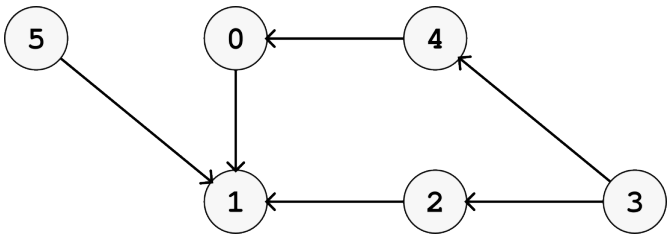

```
partidas   cidades alcançáveis
 3 ------- 3
           (3 para 3 = 0 estrada)

 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

 2 ------- 1 (já visitada: 3 para 1 = 3 estradas
              o menor caminho: 2 estradas (3, 2 e 1)
```

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

Exemplo: distâncias da cidade **3**

mapa[origens][destinos]

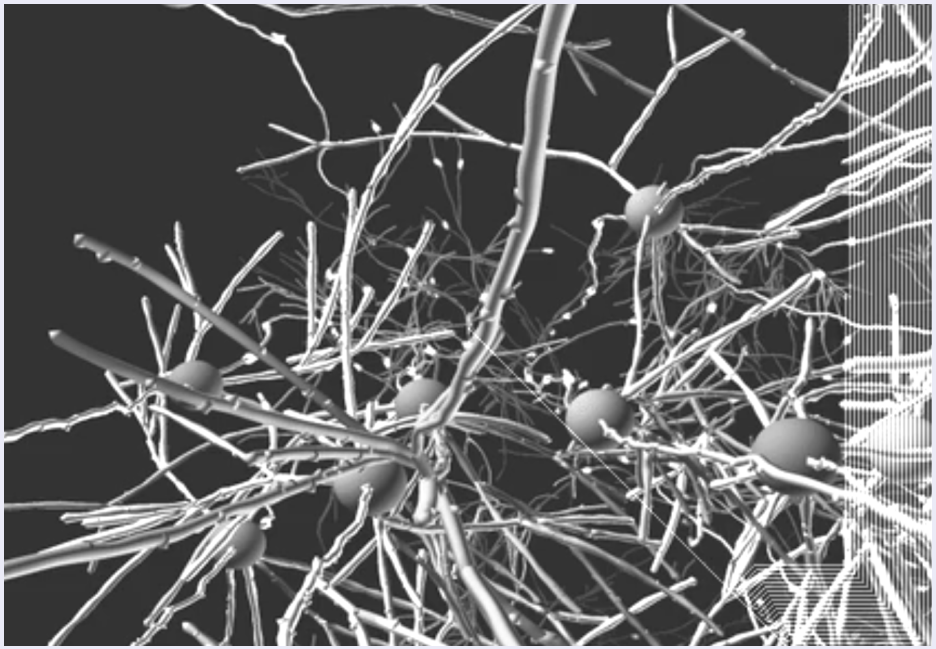
	0	1	2	3	4	5
0	0	1	0	0	0	0
1	0	0	0	0	0	1
[2]	0	[1]	0	0	0	0
[3]	0	0	[1]	0	1	0
4	1	0	0	0	0	0
5	0	0	0	0	0	0

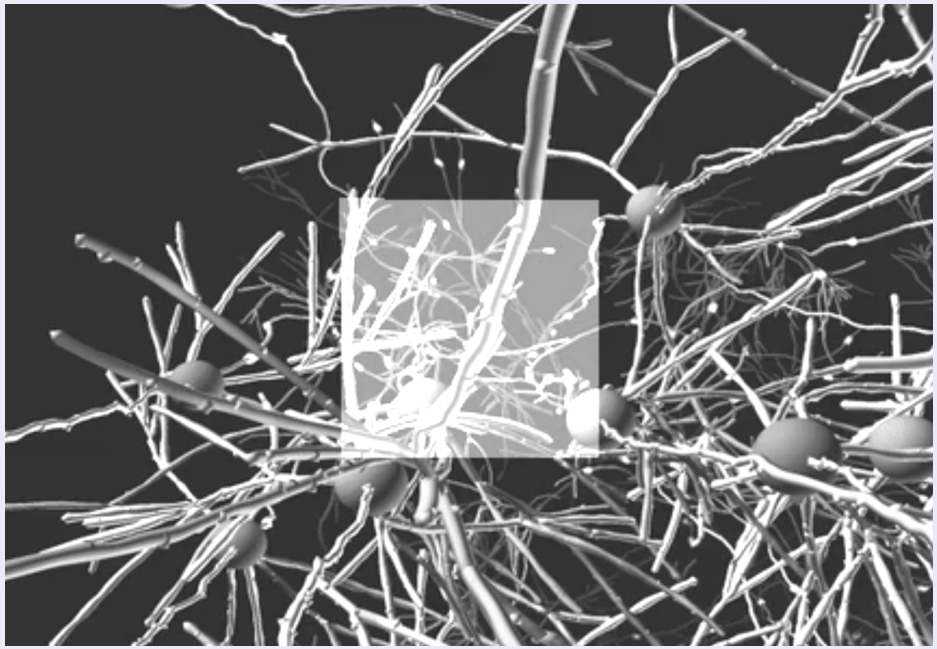



```
partidas   cidades alcançáveis
 3 ------- 3
           (3 para 3 = 0 estrada)

 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

 2 ------- 1 (já visitada: 3 para 1 = 3 estradas
              o menor caminho: 2 estradas (3, 2 e 1)
```

Exemplo: comportamento fifo x lifo

Clareamento de imagem PGM





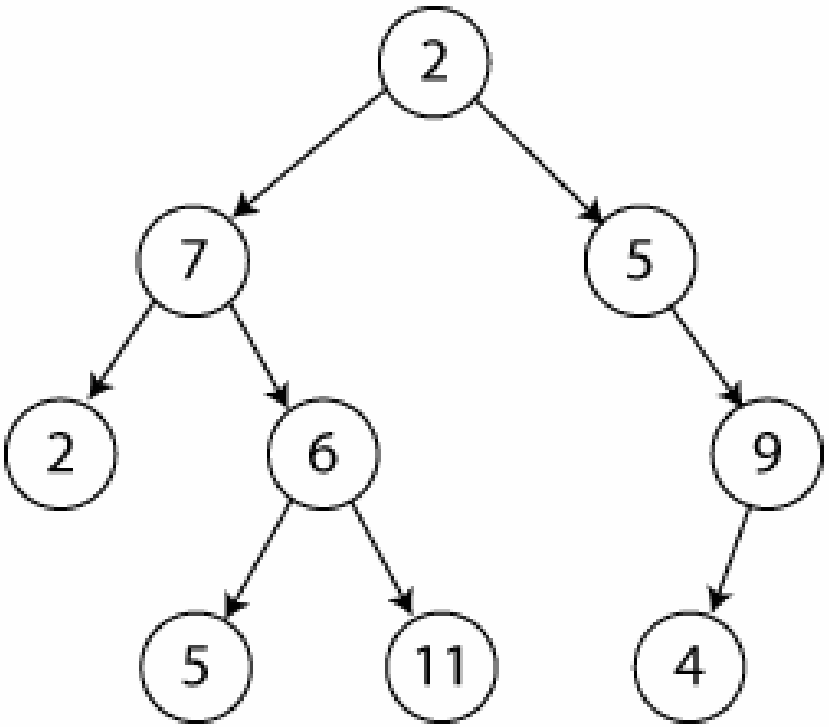
Roteiro

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

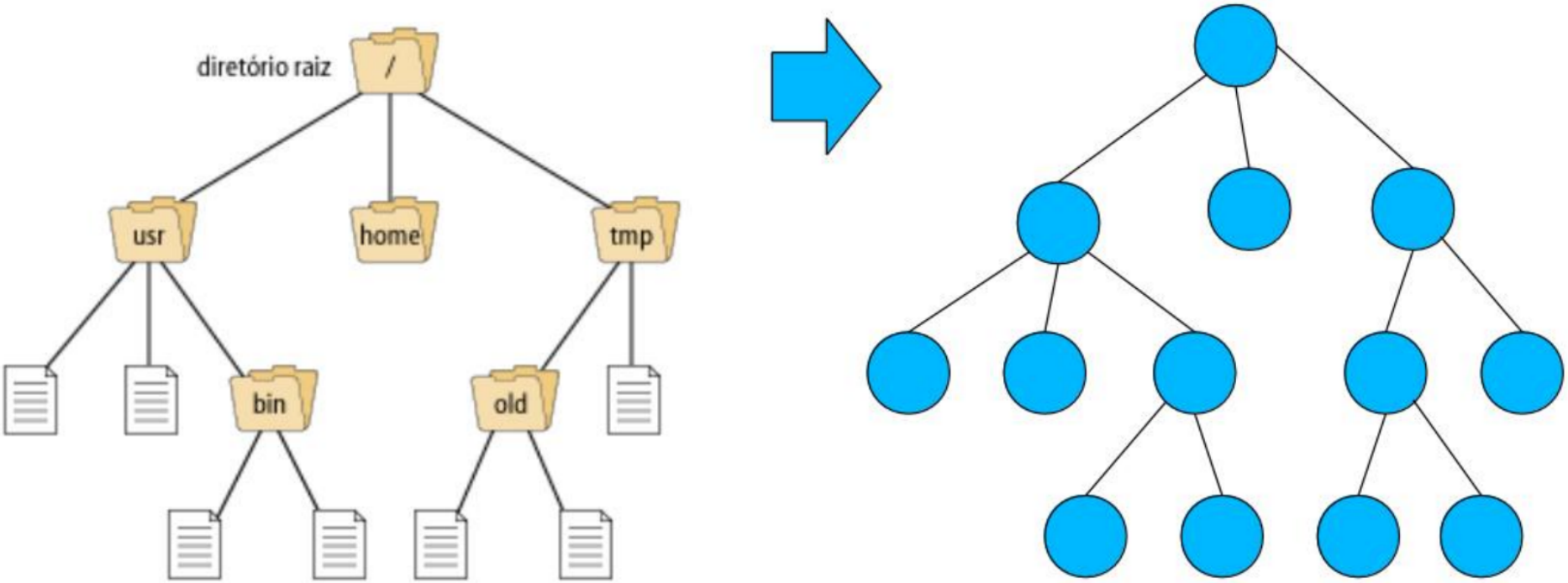
Árvore

- É uma estrutura de dados



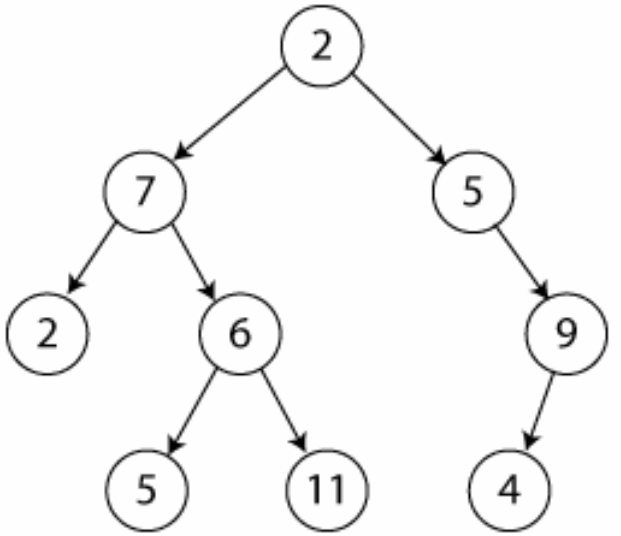
Árvores

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



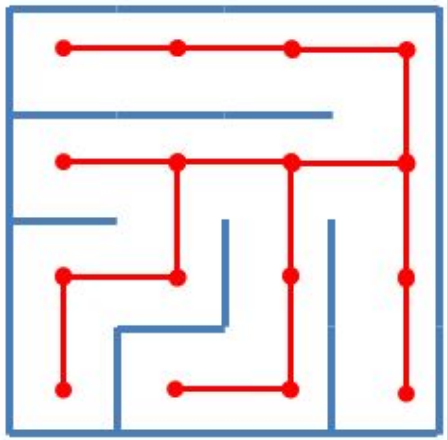
Árvores binárias

- 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



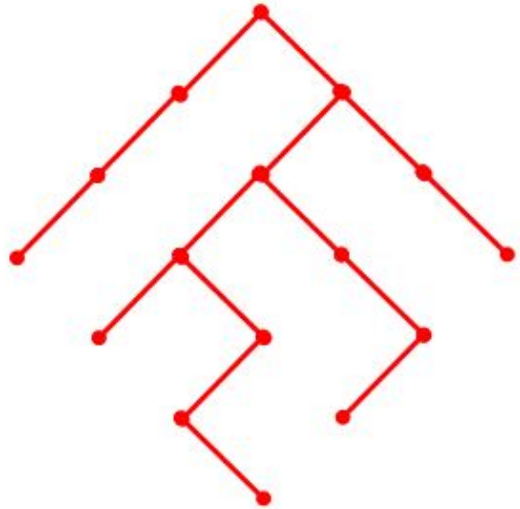


Figura: `https://openhome.cc/eGossip/OpenSCAD/SimpleGeneratedMaze.html`

Árvores binárias

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