```
int main(int argc, char** argv)
{
  char c = 0;
  char* commands = "ads pq"; // key commands: "left,right,rotate,confirm,pause,quit"
  int speed = 2; // sets max moves per row
  int moves_to_go = 2;
  int full = 0; // whether board is full
  init(); // initialize board an tetrominoes


  cur =
  state[
  state[

  system
  printf

  while
    {
      dr

    // process user action
    c = getchar();   // get new action
    if (c == commands[0] && !intersect(cur, state[0]-1, state[1])) state[0]--; // move left
    if (c == commands[1] && !intersect(cur, state[0]+1, state[1])) state[0]++; // move right
    if (c == commands[2] && !intersect(cur->rotated, state[0], state[1])) cur = cur->rotated;
    if (c == commands[3]) moves_to_go=0;

    // scroll down
    if (!moves_to_go--)
      {
        if (intersect(cur,state[0],state[1]+1)) // if tetromino intersected with sth
          {
            cramp_tetromino();
            remove_complete_lines();
            cur = &tetrominoes[rand() % NUM_POSES];
            state[0] = (WIDTH - cur->width)/2;
```

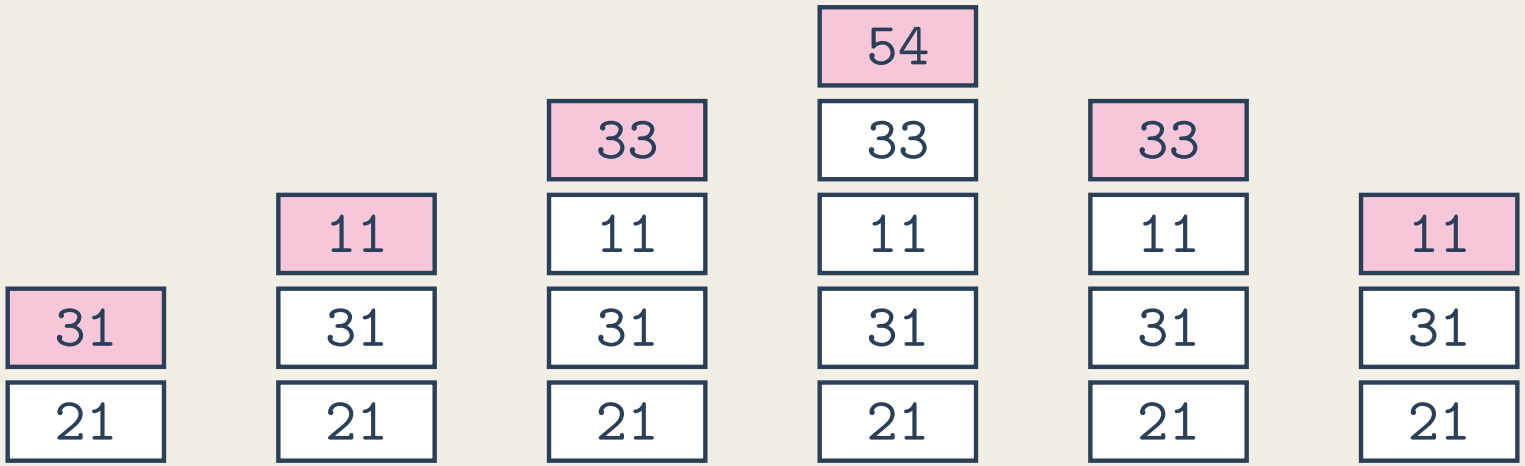
MAC122 - PRINCÍPIOS DE DESENVOLVIMENTO DE ALGORITMOS

Filas

Pilha: Definição

Tipo de dados abstrato para manipular conjuntos ordenados de objetos acessados/removidos em ordem inversa de inserção (LIFO: *last-in first-out*)

- Empilhar (*push*): Insere elemento no topo
- Desempilhar (*pop*): Remove elemento do topo
- Topo (*peek*): Devolve elemento no topo
- ...



Revisão: TAD Pilha

```c
/* pilha.h */
typedef int elem;          /* Tipo do elemento da pilha */
typedef struct pilhaTCD *pilha; /* Estrutura de dados
                                   (depende de implementação) */
pilha      CriaPilha (void);  /* Cria nova pilha vazia */
void       DestroiPilha (pilha p);    /* Destrói pilha */
void       Empilha (pilha p, elem x);     /* Empilha x */
elem       Desempilha (pilha p); /* Remove e ret. topo */
int        TamanhoPilha (pilha p); /* Tamanho da pilha */
int        PilhaVazia (pilha p); /* Pilha esta' vazia? */
int        PilhaCheia (pilha p); /* Pilha esta' cheia? */
elem TopoPilha (pilha p);          /* Retorna topo */
```

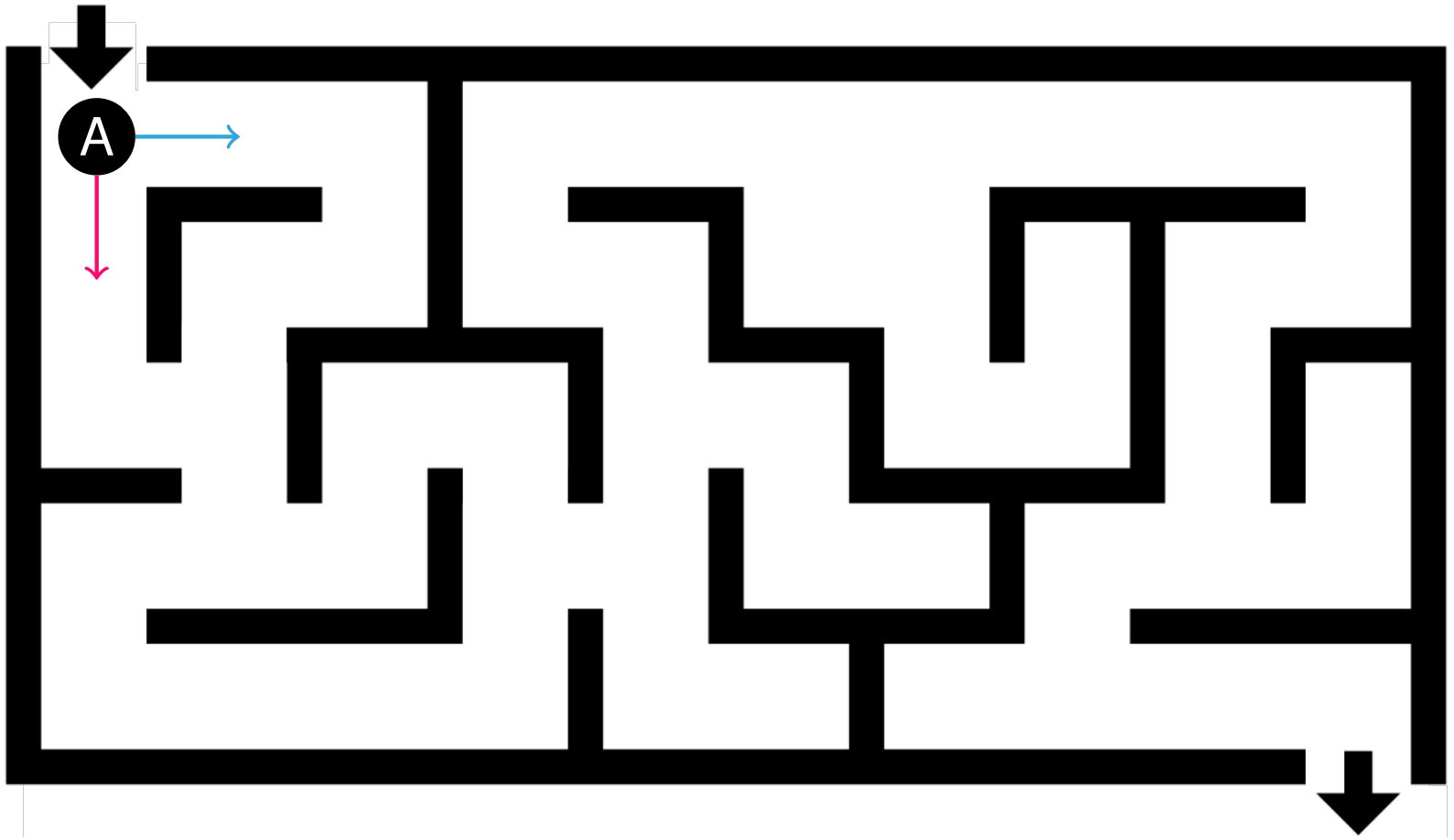
Exercício

```
1 /* Lê uma sequência de n inteiros do usuário e exibe
      números em ordem inversa
2 Exemplo: Usuário entra com 3, 4 e 5, exibir 5 4 3  */
3 void inverte(int n) {
4
5  /* Como fazer usando pilha (sem criar vetor)? */
6
7 }
```

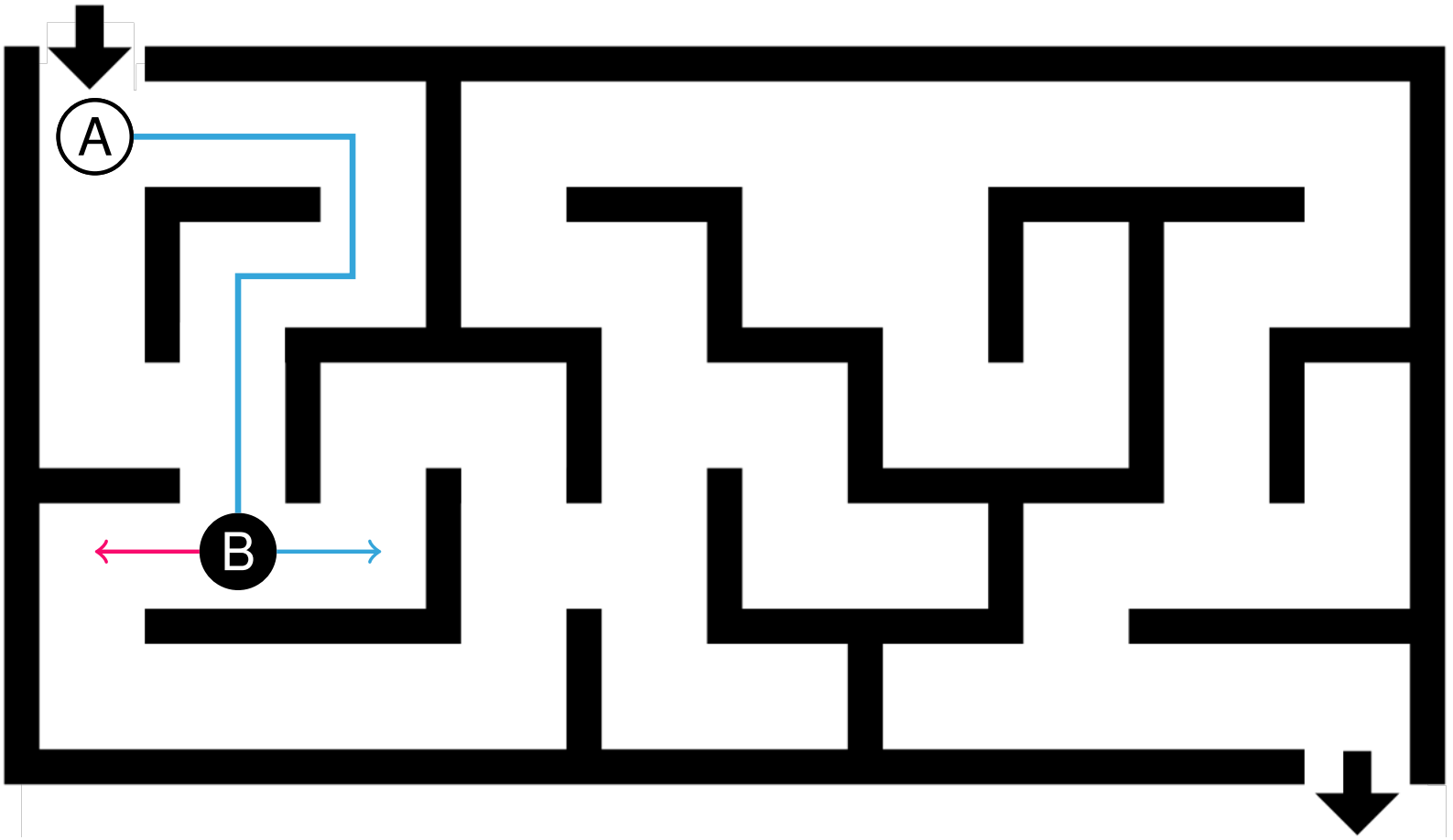
Exercício: Solução

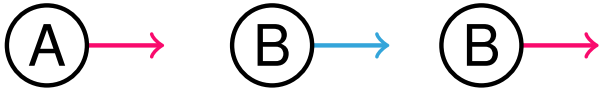
```
1  /* Lê uma sequência de n inteiros do usuário e exibe
       números em ordem inversa
2  Exemplo: Usuário entra com 3, 4 e 5, exibir 5 4 3  */
3  void inverte(int n) {
4      int num;
5      pilha = CriaPilha();
6      while (n) {
7          scanf("%d", &num);
8          Empilha(p, num);
9          n--;
10     }
11     while (!PilhaVazia(p))
12         printf("%d ", Desempilha(p));
13     DestroiPilha(p);
14 }
```

Sair do labirinto

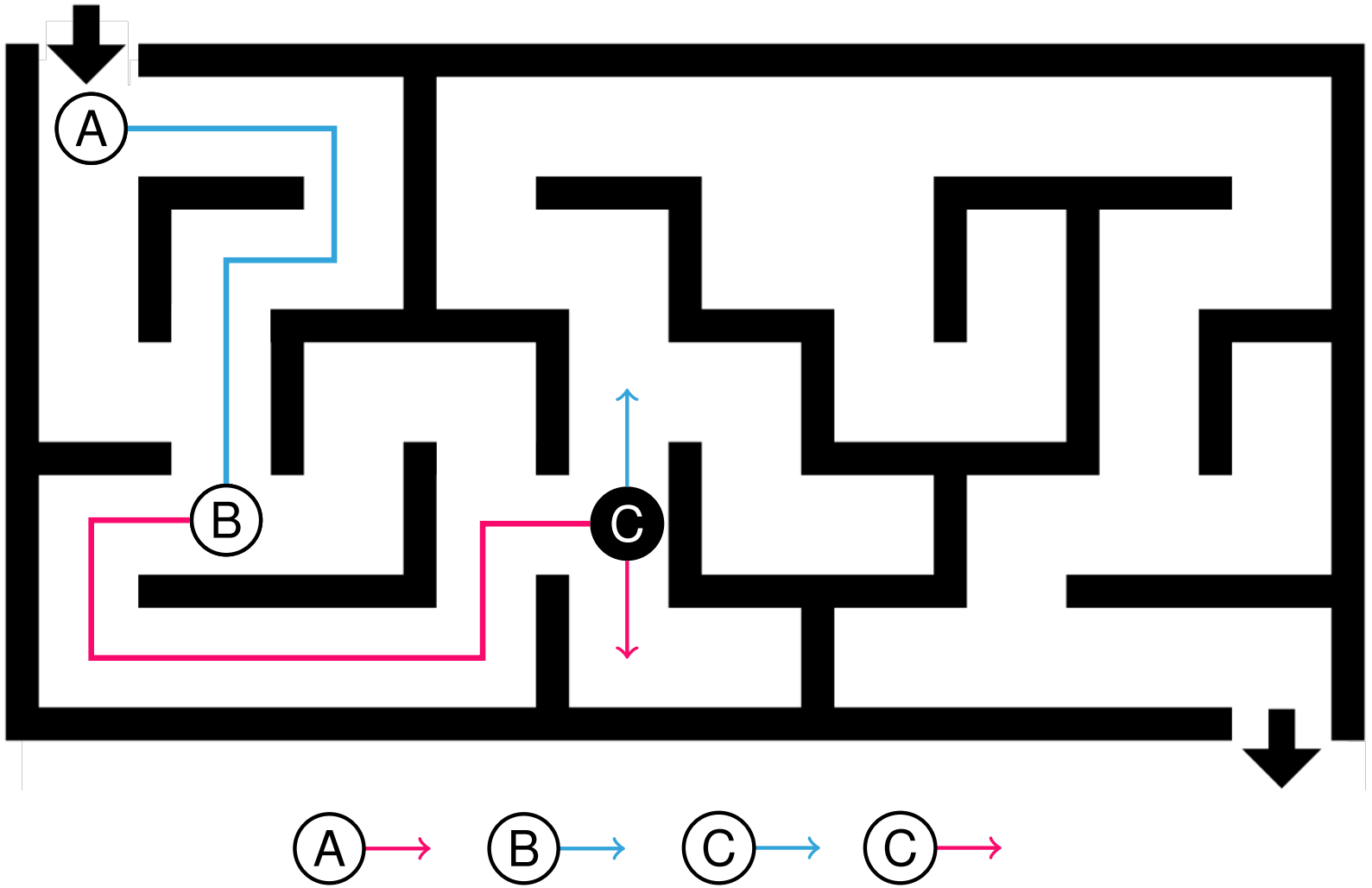


Sair do labirinto

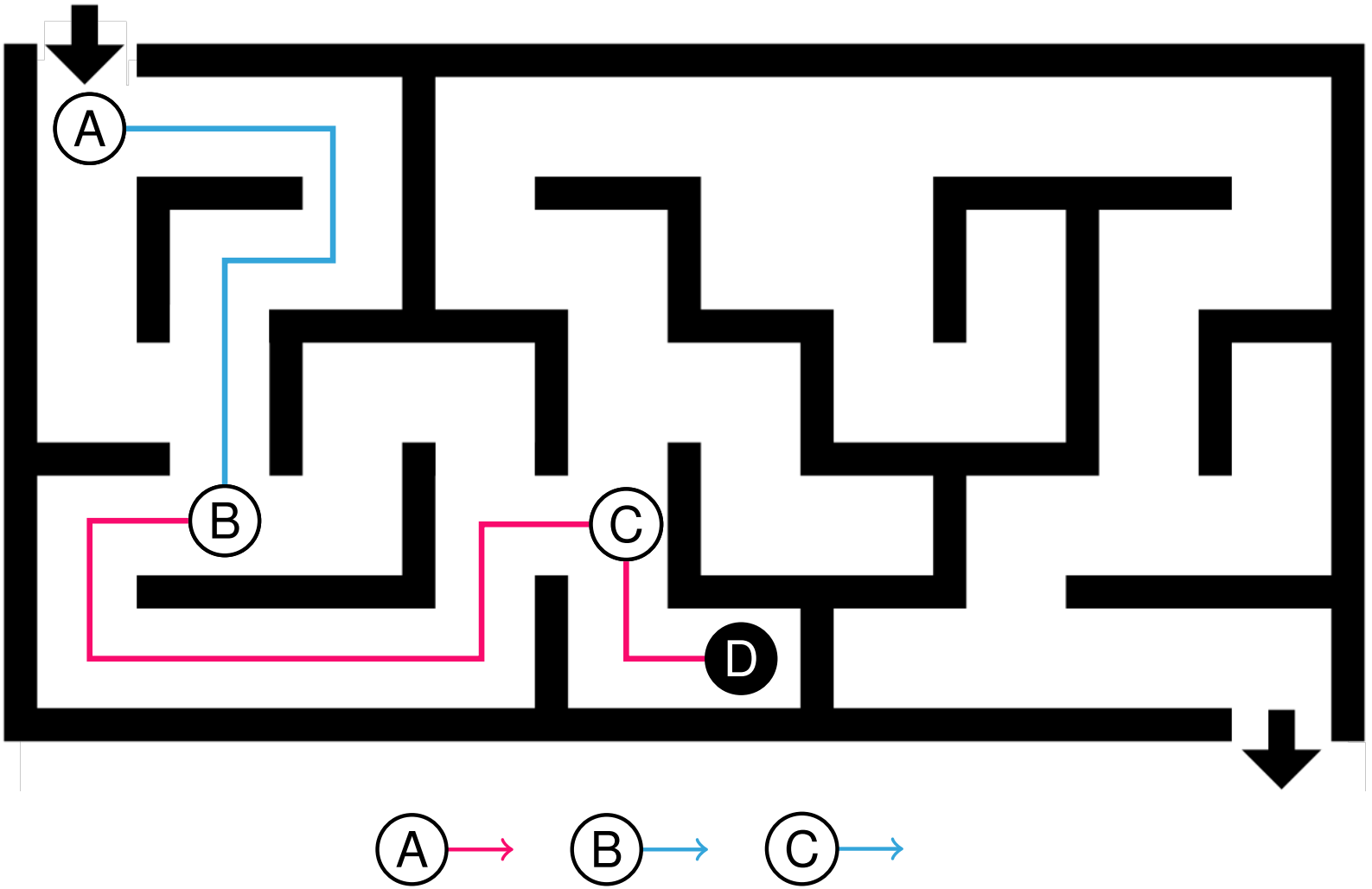




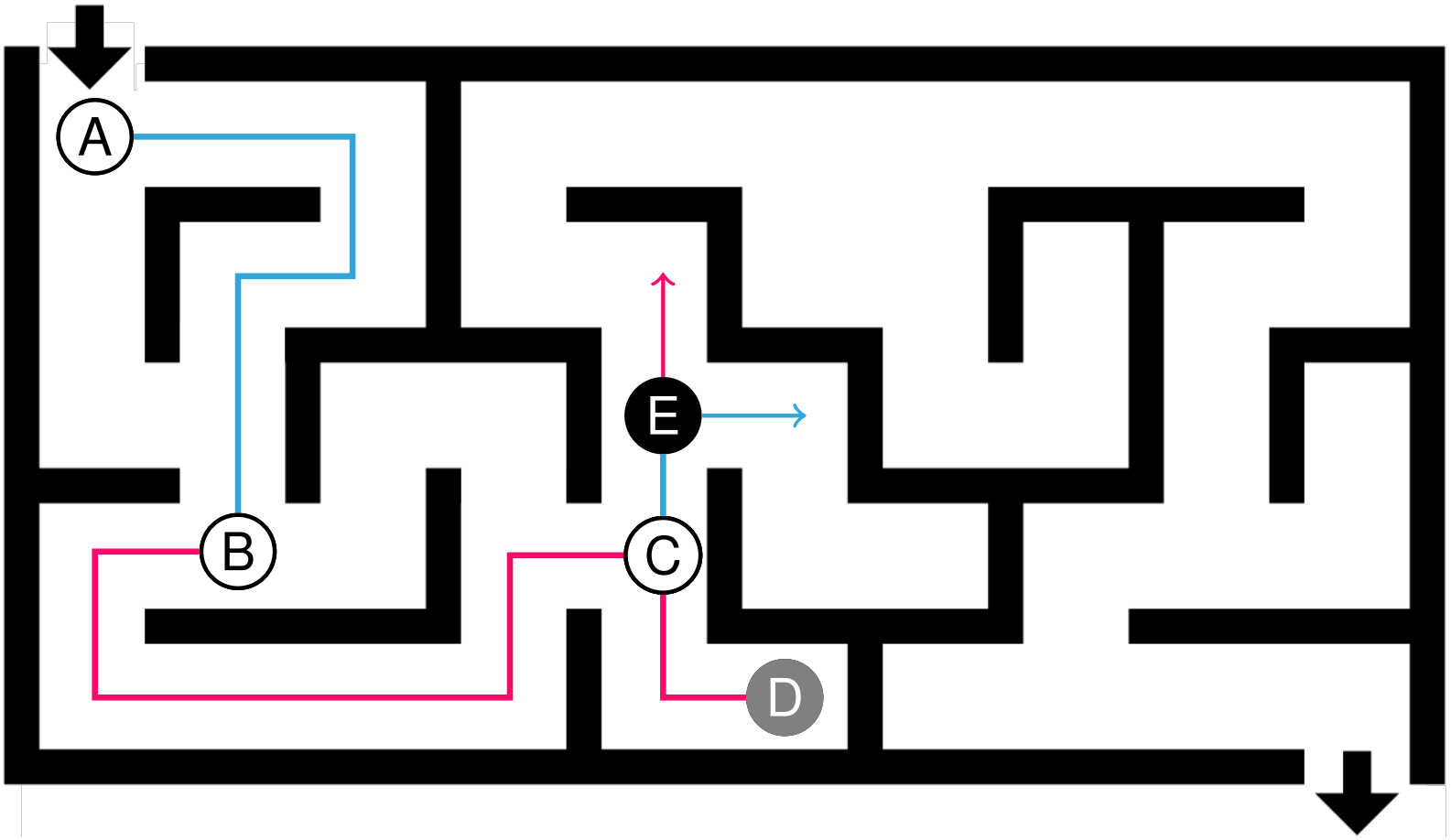
Sair do labirinto



Sair do labirinto

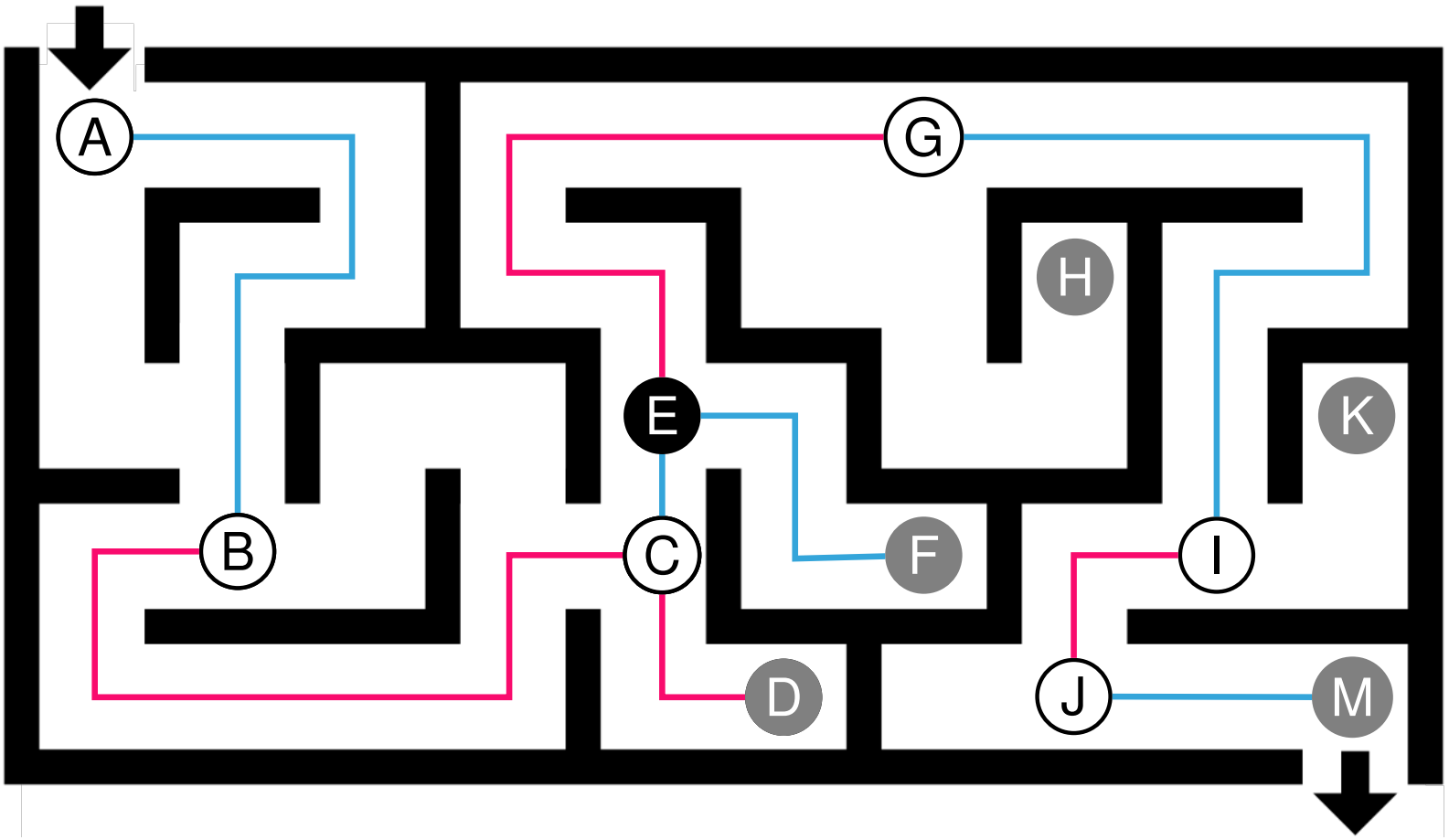


Sair do labirinto



A → B → E → E →

Sair do labirinto



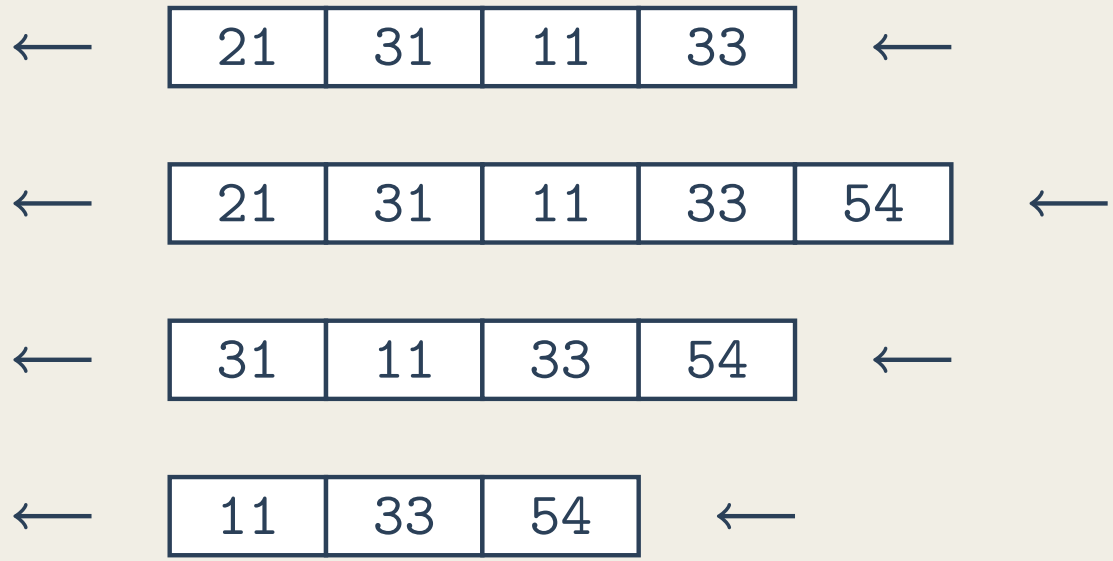
Sair do Labirinto: Solução

```
1  typedef struct {
2    char de; char para;
3  } ação;
4
5  void resolve_labirinto(labirinto L) {
6     ação a; char p;
7     pilha = CriaPilha();
8     Empilha(p, {'A', 'e'}); /* ações em ponto A */
9     Empilha(p, {'A', 'd'});
10    while (!PilhaVazia(p)) {
11      a = Desempilha(p);
12      if (a.de == 'M') printf("encontrei!");
13      p = PróximoPonto(L, a);
14      Empilha(p, {p, 'e'} );
15      Empilha(p, {p, 'd'} );
16    }
17    DestróiPilha(p);
18 }
```

Fila: Definição

Tipo de dados abstrato para manipular conjuntos ordenados de objetos removidos/acessados em ordem de inserção (FIFO: *first-in first-out*):

- Enfileirar (*queue*): Insere no fim (cauda)
- Desinfileirar (*deque*): Remove no início (cabeça)
- ...



Buffer de entrada

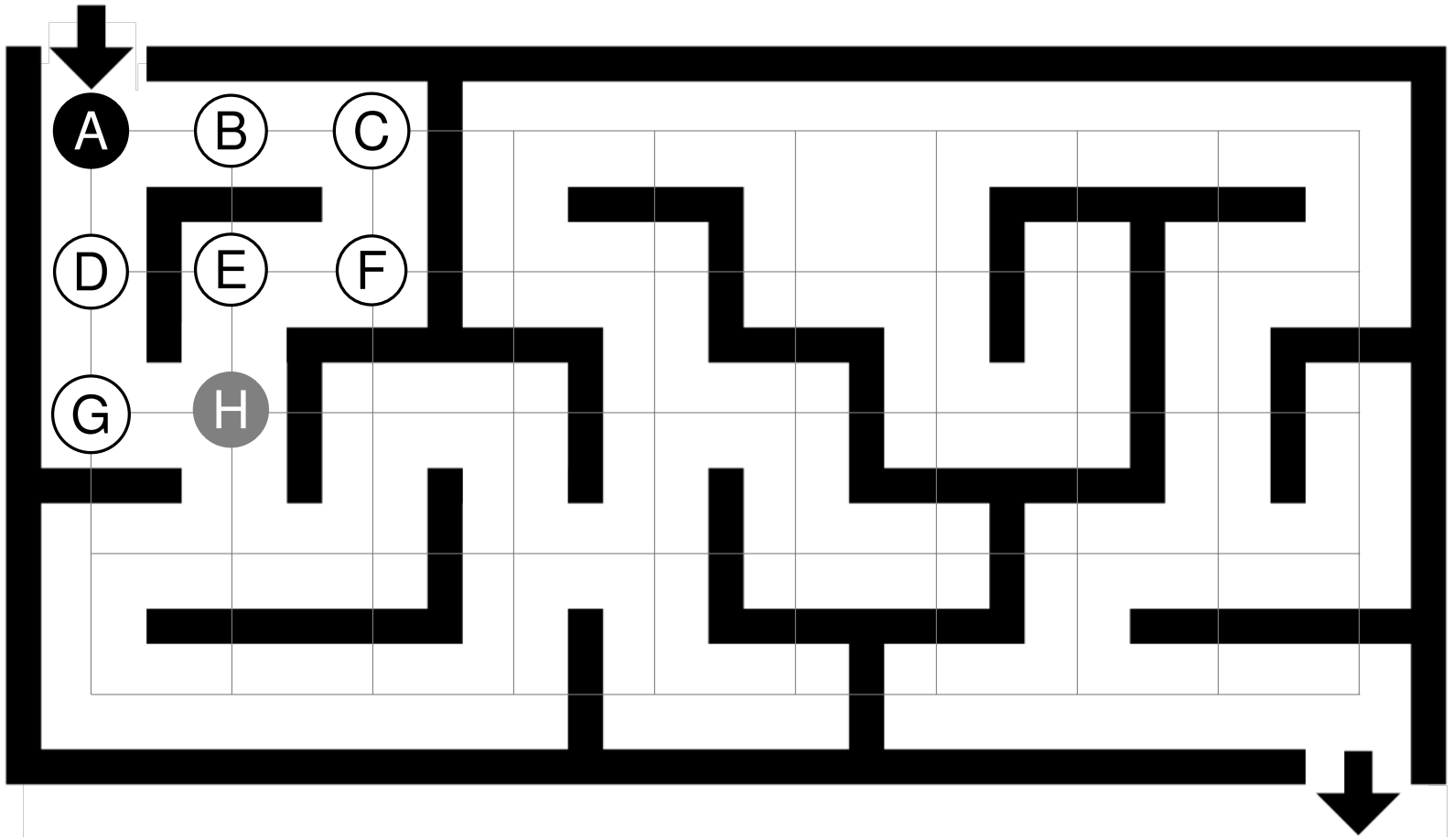
Entrada do teclado:


```c
scanf("%d", &x); /* x = 31 */
```


```c
scanf("%d", &x); /* x = 11 */
```

Sair em menos passos?

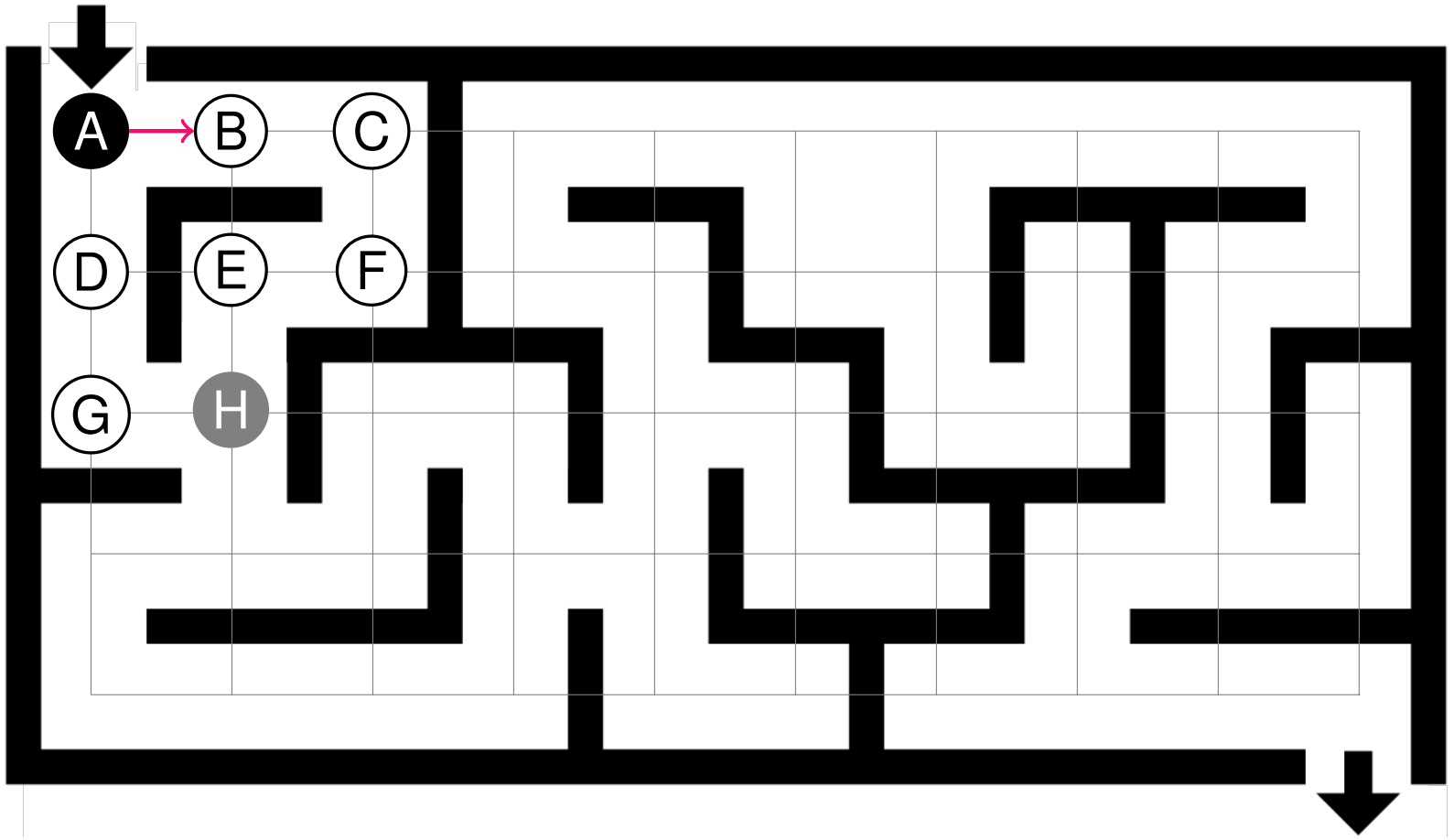
Ir de ponto A a ponto B, cada ação leva a ponto vizinho



A → B A → D

Sair em menos passos?

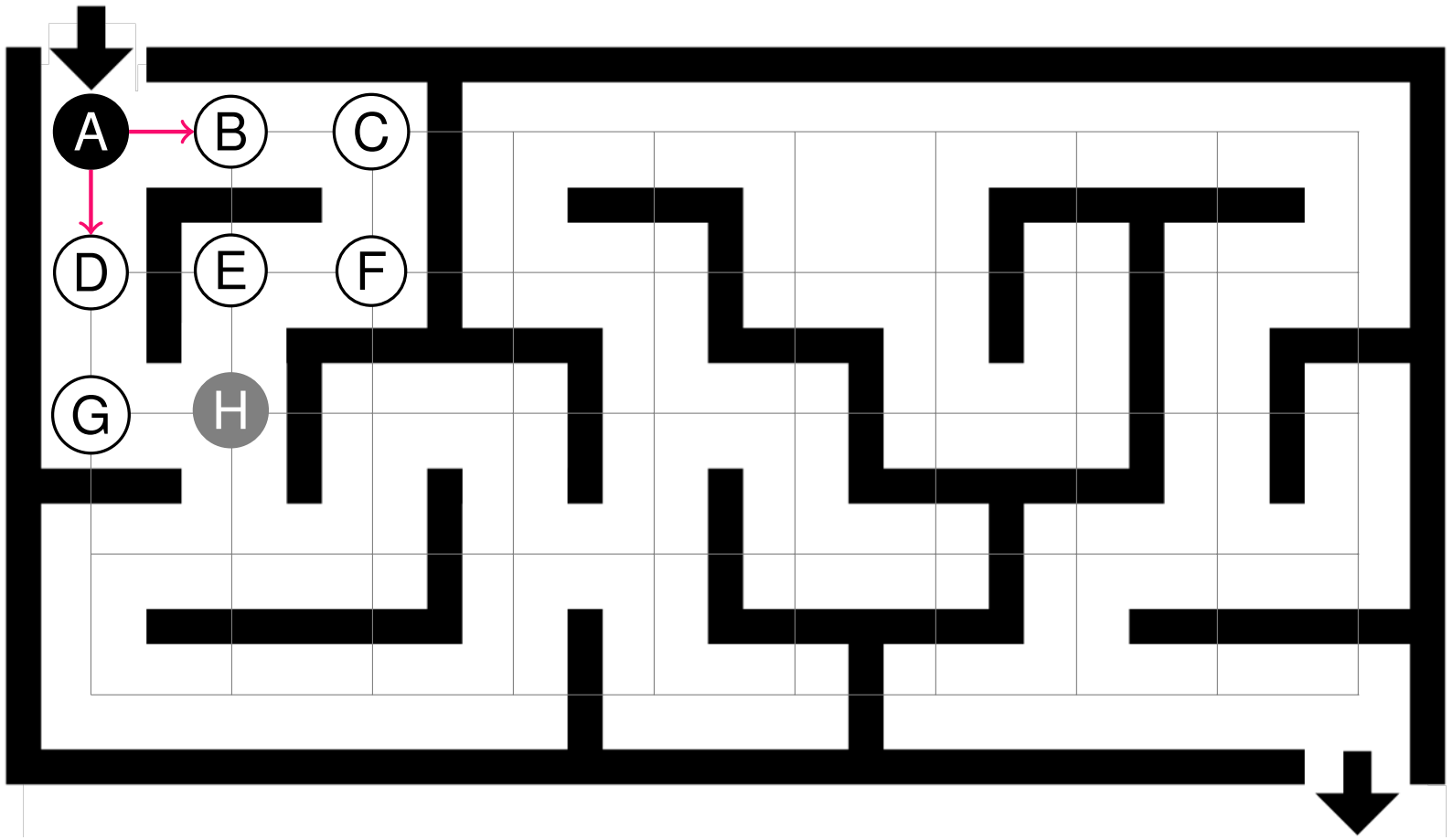
Ir de ponto A a ponto B, cada ação leva a ponto vizinho



A → D → B → C

Sair em menos passos?

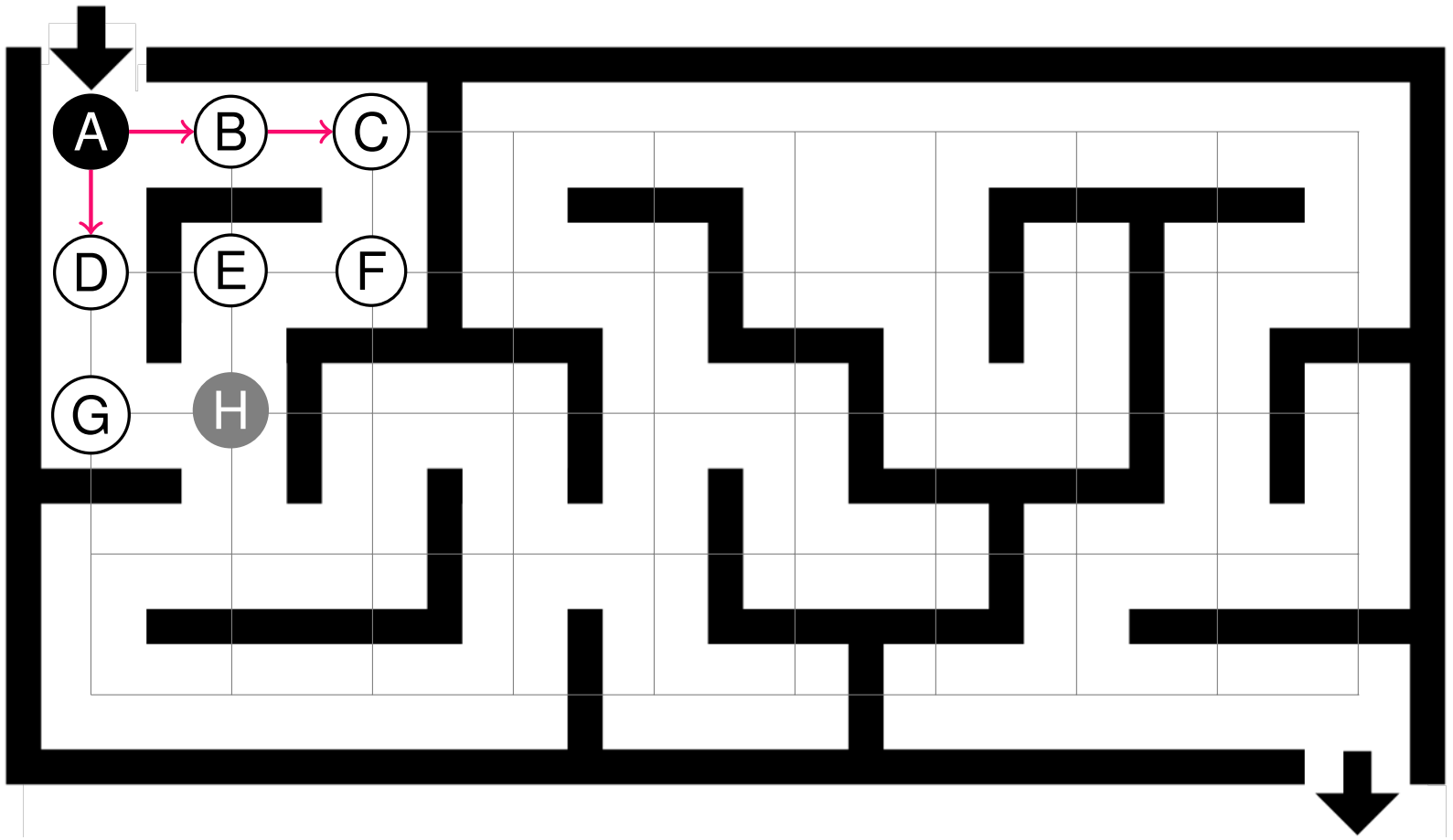
Ir de ponto A a ponto B, cada ação leva a ponto vizinho



B → C D → G

Sair em menos passos?

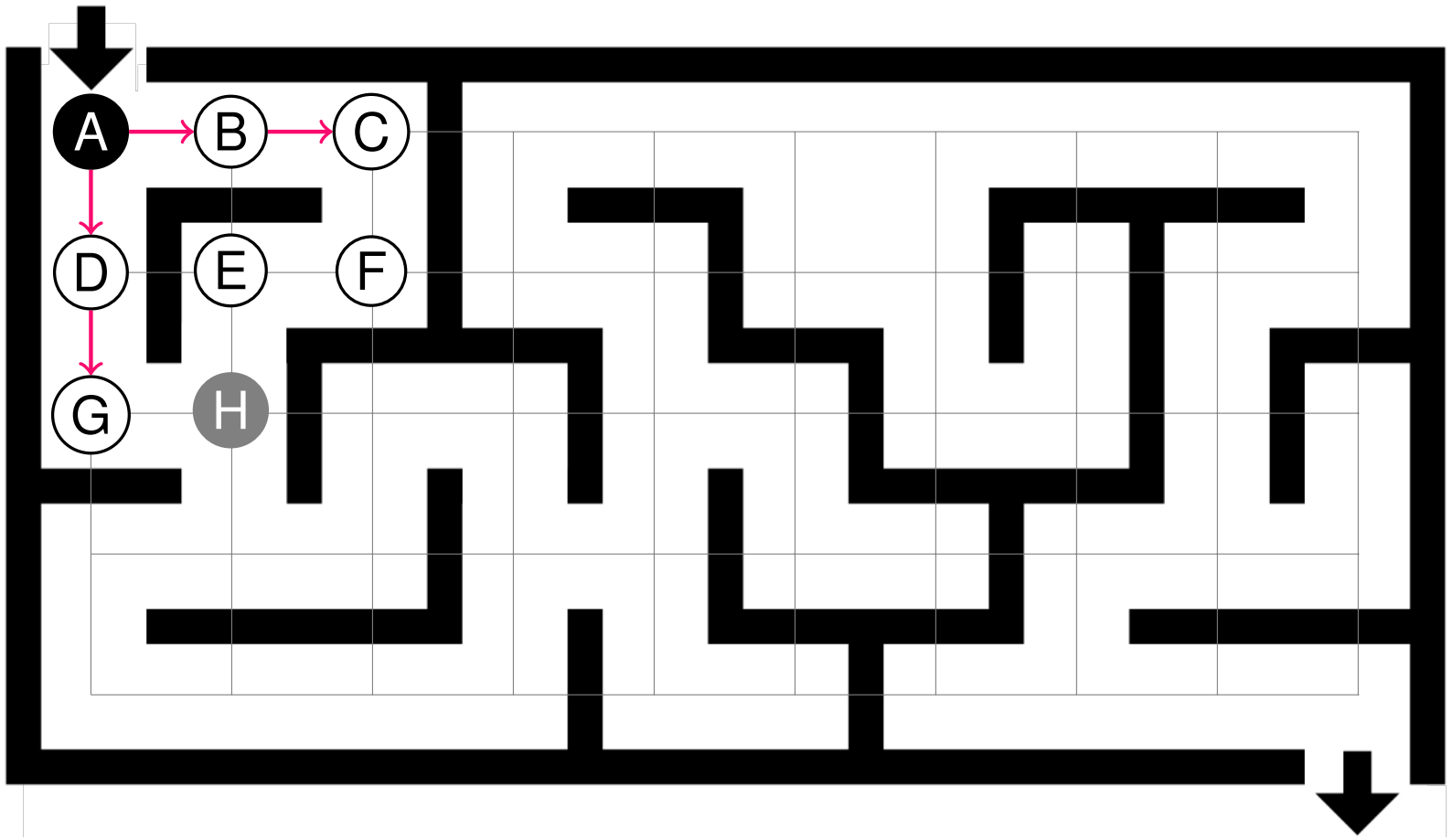
Ir de ponto A a ponto B, cada ação leva a ponto vizinho



D → G → C → F

Sair em menos passos?

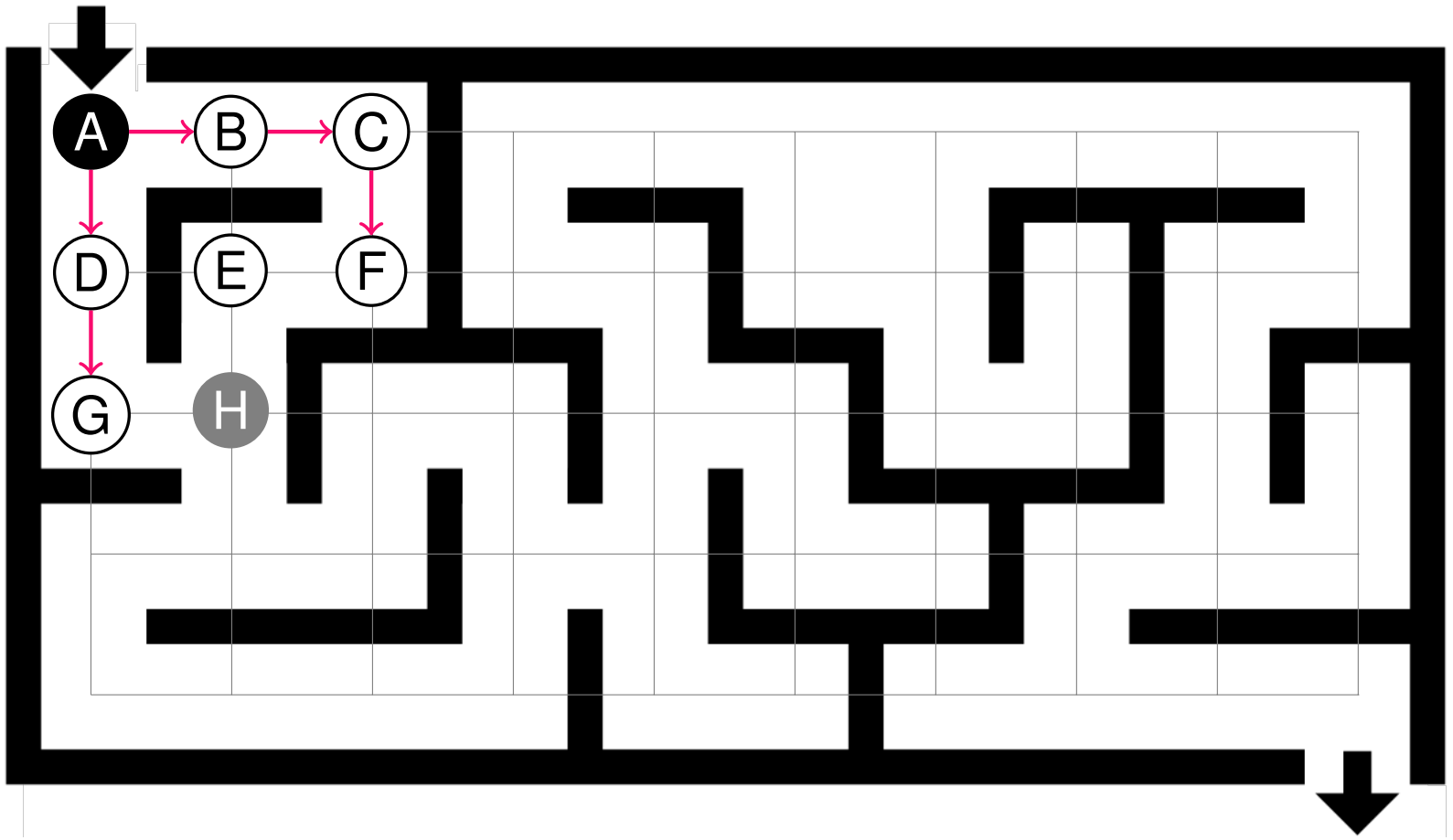
Ir de ponto A a ponto B, cada ação leva a ponto vizinho



C → F → G → H

Sair em menos passos?

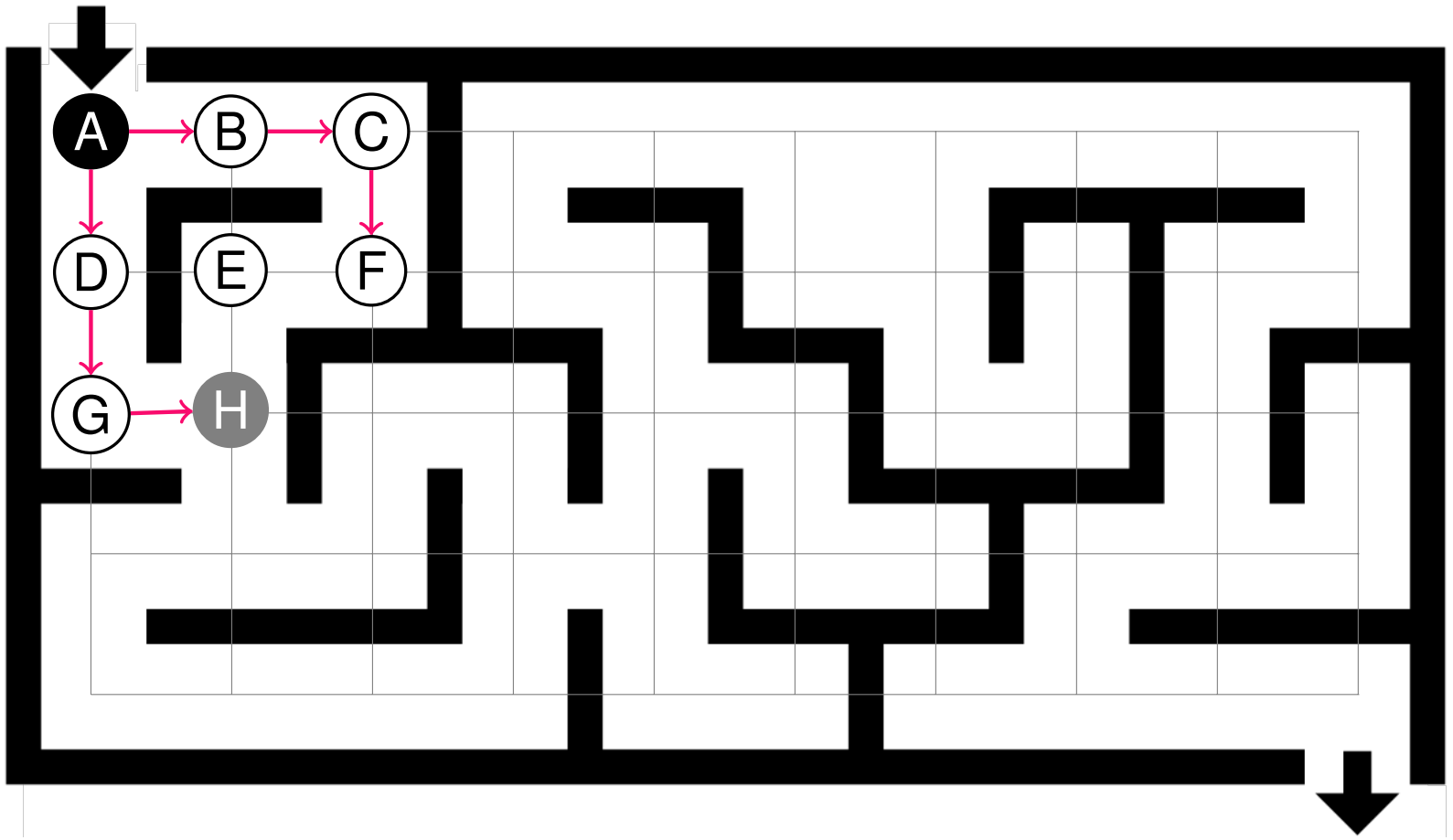
Ir de ponto A a ponto B, cada ação leva a ponto vizinho



G → H → F → E

Sair em menos passos?

Ir de ponto A a ponto B, cada ação leva a ponto vizinho



F → E

TAD Fila: Interface

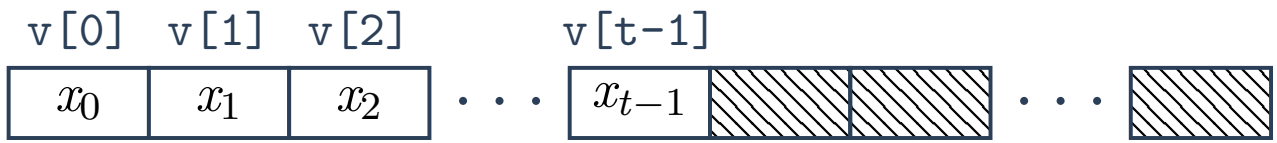
```
1  /* fila.h */
2  typedef int elem;          /* Tipo do elemento da fila */
3  typedef struct filaTCD *fila; /* Estrutura de dados
4                                   (depende de implementação) */
5  fila      CriaFila (void);  /* Cria nova fila vazia */
6  void      DestroiFila (fila f);    /* Destrói fila */
7  void      Enfileira (fila f, elem x);  /* Enfileira x */
8  elem      Desenfileira (fila f); /* Remove e ret. elem */
9  int       TamanhoFila (fila f); /* Tamanho da pilha */
10 int       FilaVazia (fila f); /* Fila esta' vazia? */
11 elem      CabeçaFila (fila f);    /* Retorna elem */
```

Fila: Implementação em vetor (de tamanho fixo)

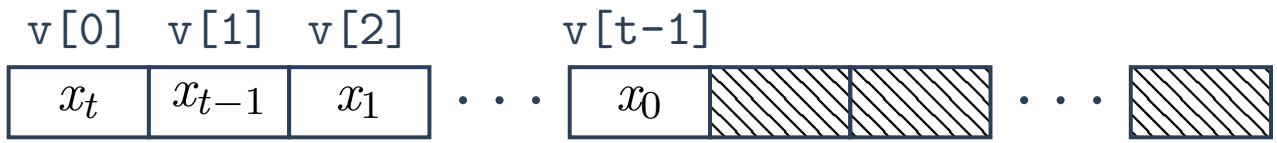
Como implementar fila para que inserção/remoção/consulta sejam feitas em tempo constante (independentemente do tamanho do vetor)?

A) Não pode ser feito

B)



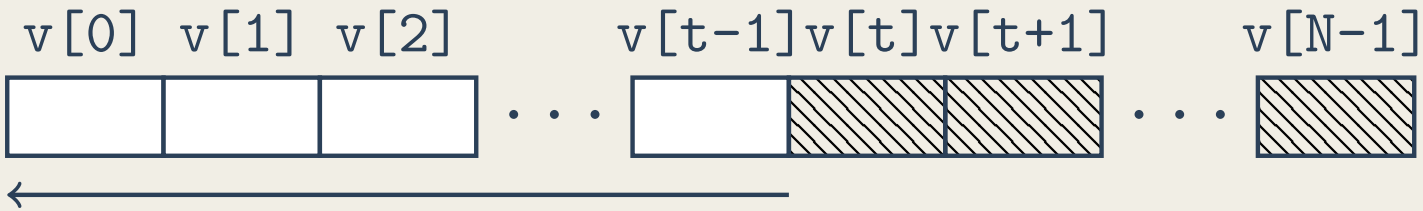
C)



Filas: Implementação em vetor B

`vetor[0..t-1]`

Insere em `vetor[t]`, remove de `vetor[0]`

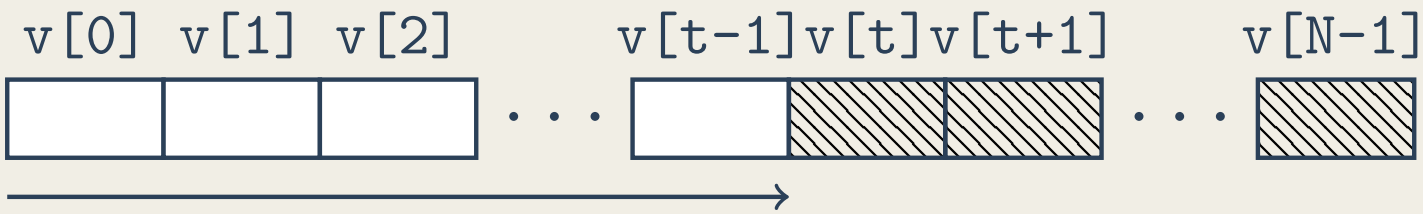


- Inserção no fim: `0` deslocamento
- Remoção no começo: `t` deslocamentos

Filas: Implementação em vetor C

`v[0..t-1]`

Insere em `vetor[0]`, remove de `vetor[t-1]`

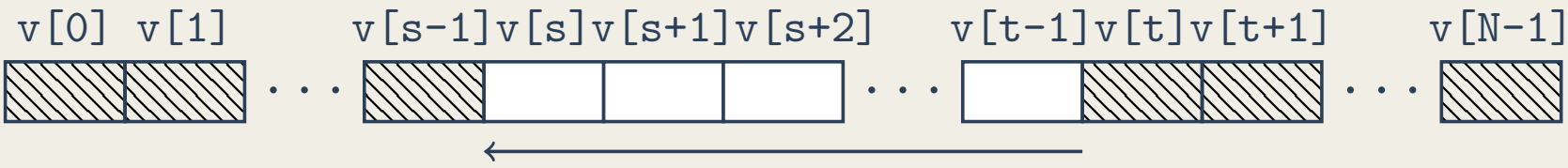


- Inserção no começo: `t` deslocamentos
- Remoção no fim: `0` deslocamento

Filas: Implementação em vetor

`vetor[s..t-1]`

Insere em `vetor[t]`, remove de `vetor[s]`

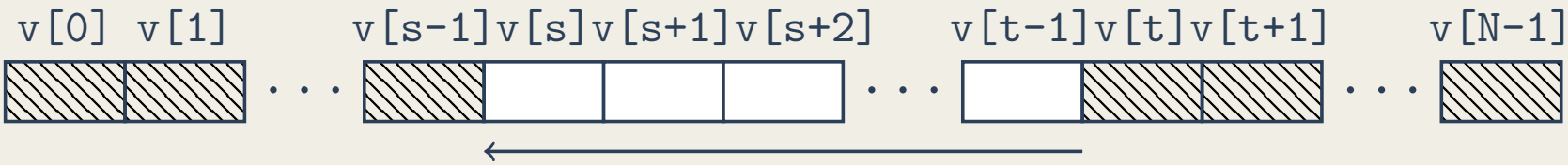



```c
/* Inserção */
 vetor[t++] = x; /* 0 deslocamento */
```

Filas: Implementação em vetor

`vetor[s..t-1]`

Insere em `vetor[t]`, remove de `vetor[s]`

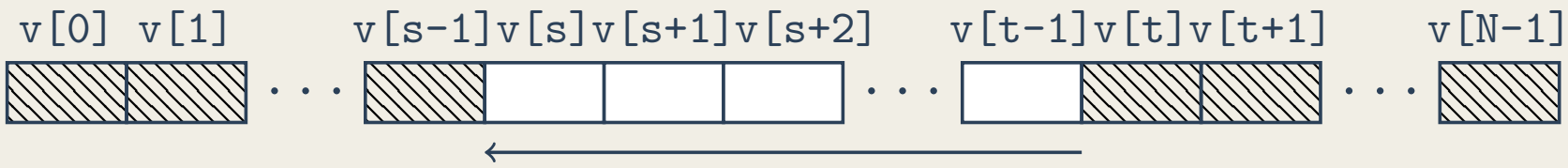



```c
/* Inserção */
 vetor[t++] = x; /* 0 deslocamento */
```

```c
/* Remoção */
x = vetor[s++]; /* 0 deslocamento */
```

Filas: Implementação em vetor

`v[s..t-1]`



Cheia

`t == N`

Vazia

`s == t`

Filas: Implementação em vetor

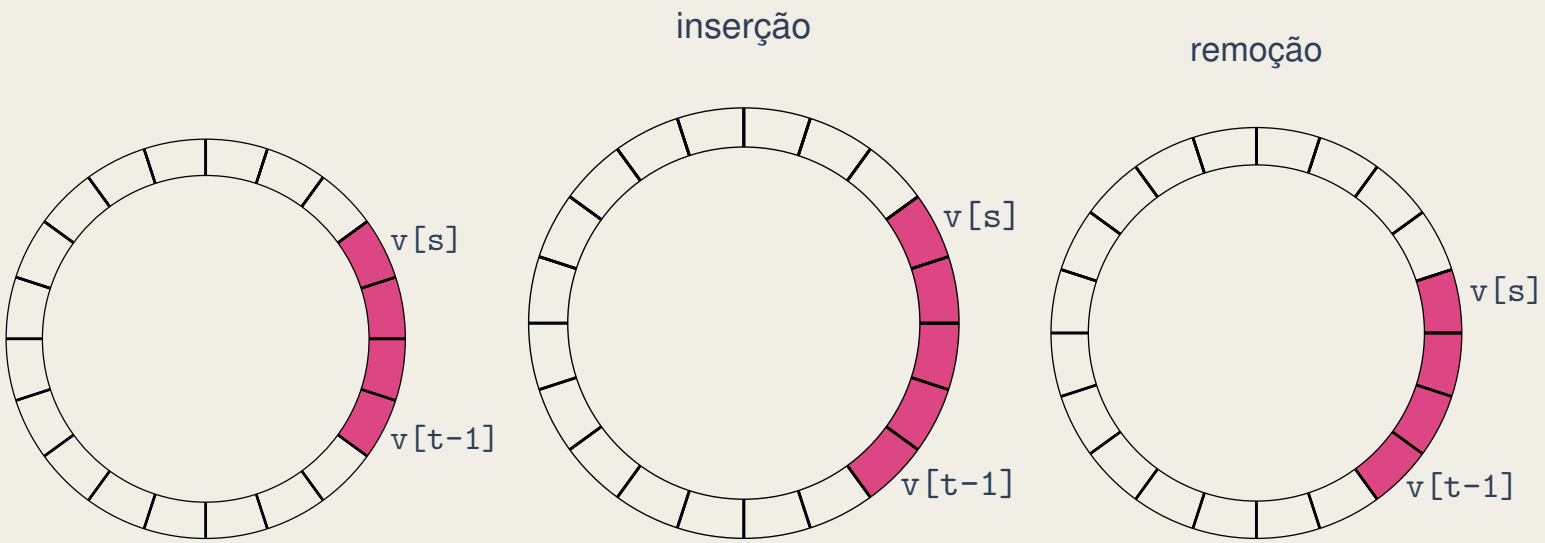
Vantagens

Inserção e remoção em tempo constante

Desvantagem

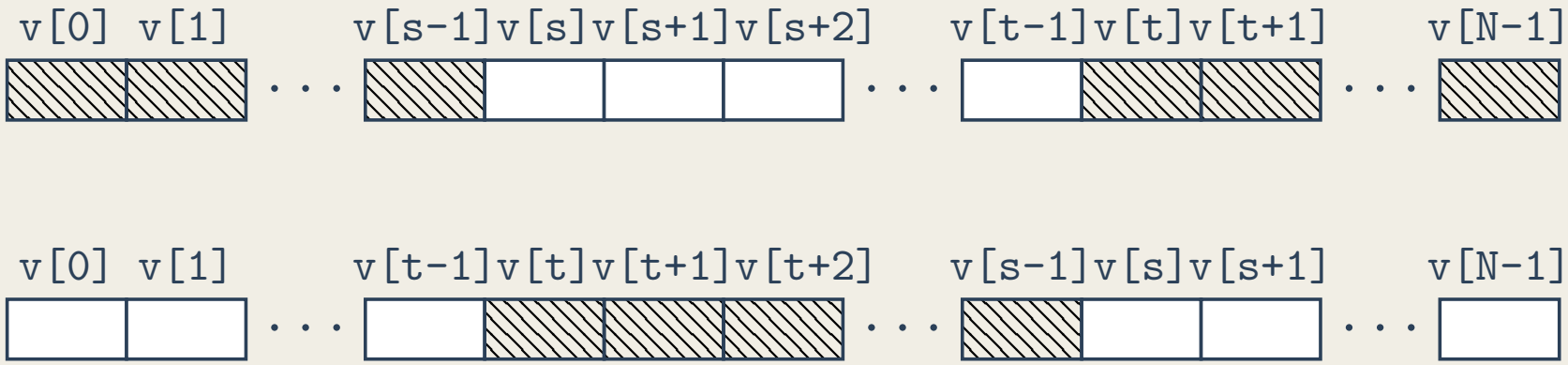
Requer alocação de memória proporcional ao número de elementos que “passaram” pela fila, independentemente do tamanho máximo da fila em qualquer momento (compare com implementação em vetor de pilhas)

Filas: Implementação em vetor circular



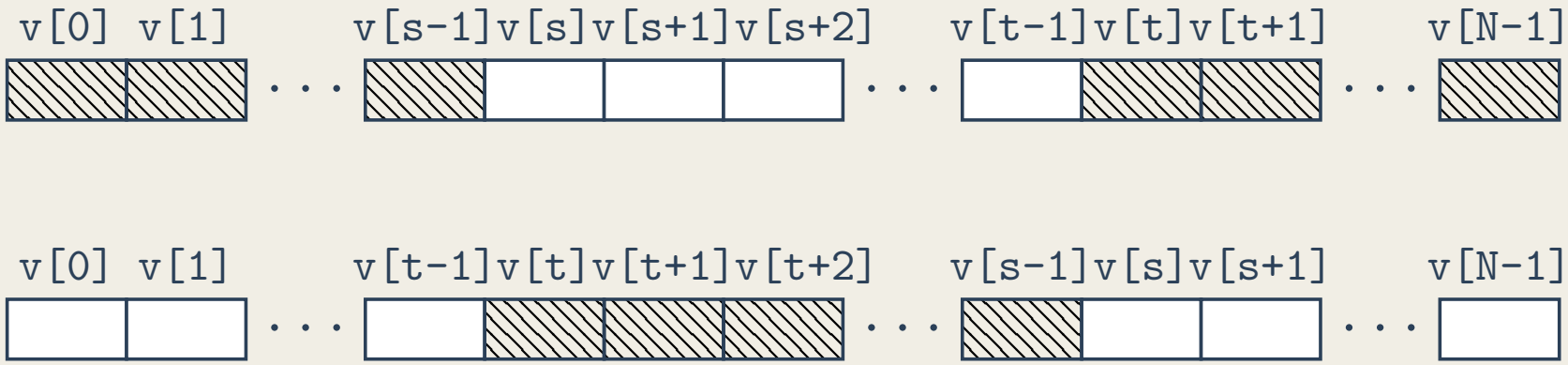
Filas: Implementação em vetor circular

`v[s..t-1]` ou `concat(v[s..N-1],v[0..t-1])`



Filas: Implementação em vetor circular

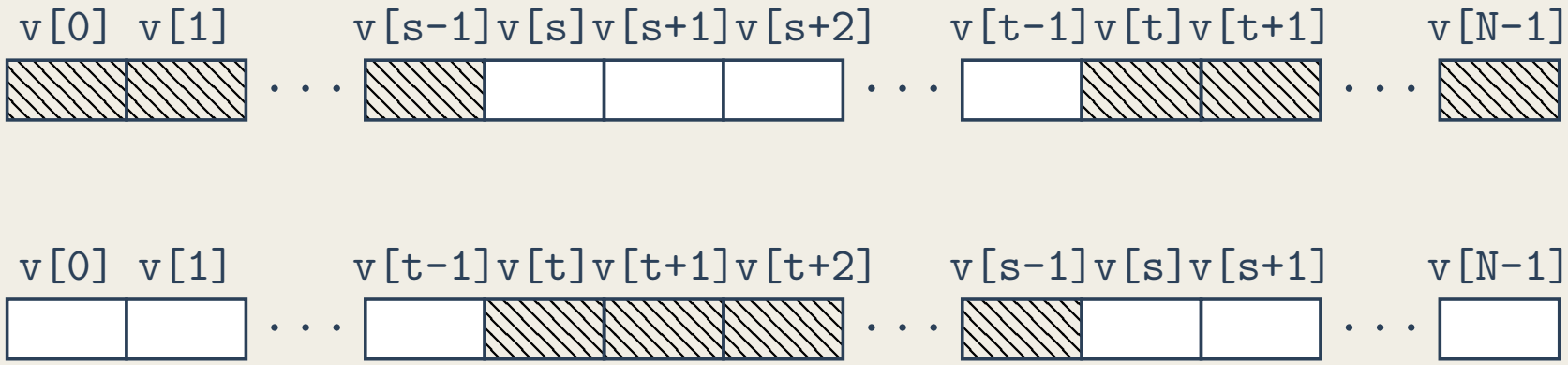
`v[s..t-1]` ou `concat(v[s..N-1],v[0..t-1])`




```
1 /* Inserção */
2  v[t++] = x; if (t == N) t = 0;
```

Filas: Implementação em vetor circular

`v[s..t-1]` ou `concat(v[s..N-1],v[0..t-1])`



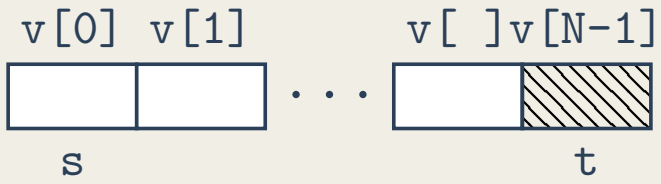

```c
/* Inserção */
v[t++] = x; if (t == N) t = 0;
```

```c
/* Remoção */
x = v[s++]; if (s == N) s = 0;
```

Filas: Implementação em vetor circular

Problema: Inserção em vetor cheio

```
1  v[t++] = x; if (t == N) t = 0;
```

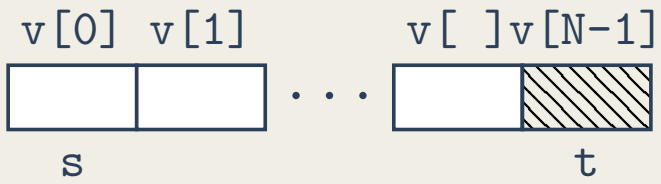


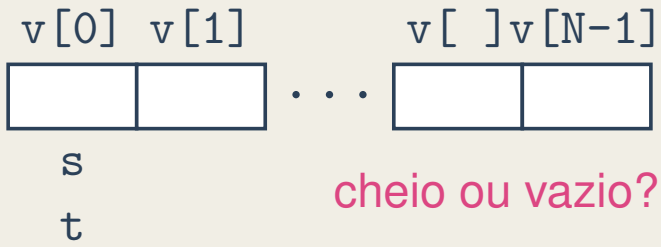
Solução: Inutilizar 1 elemento

Filas: Implementação em vetor circular

Problema: Inserção em vetor cheio

```
1  v[t++] = x; if (t == N) t = 0;
```

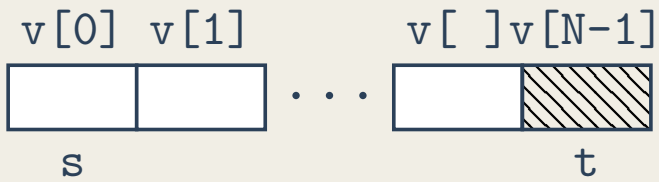


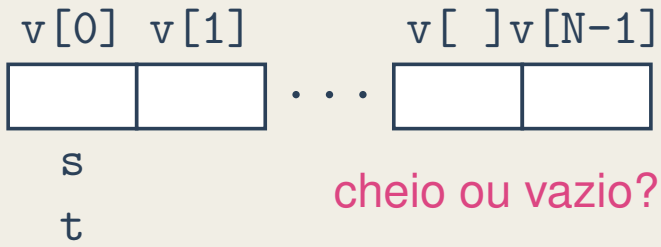


Filas: Implementação em vetor circular

Problema: Inserção em vetor cheio

```
1  v[t++] = x; if (t == N) t = 0;
```

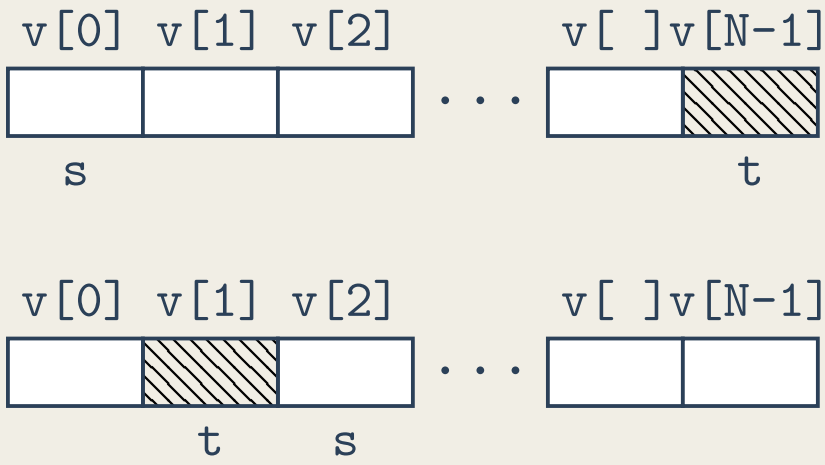




Solução: Inutilizar 1 elemento

Filas: Implementação em vetor circular

`v[s..t-1]` ou `concat(v[s..N-1],v[0..t-1])`



Cheia

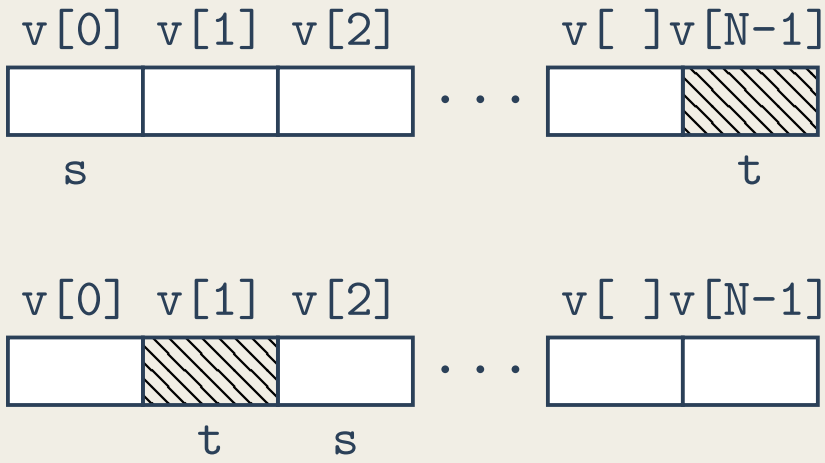
`t+1 == s` ou `s == 0` e `t+1 == N`

Vazia

`s == t`

Filas: Implementação em vetor circular

`v[s..t-1]` ou `concat(v[s..N-1],v[0..t-1])`



Cheia

```
s == (t+1) % N
```

Vazia

```
s == t
```

Filas: Implementação em lista encadeada

Como implementar fila para que inserção/remoção/consulta sejam feitas em tempo constante (independentemente do tamanho da lista)?

A) Não pode ser feito

B) 

C) 

Fila: Implementação em lista encadeada

- Manter ponteiros para células da `cabeça` (primeira) e `cauda` (última)
- Enfileirar na `cauda`
- Desenfileirar da `cabeça`

Para casa

Exercícios: 9A-9C