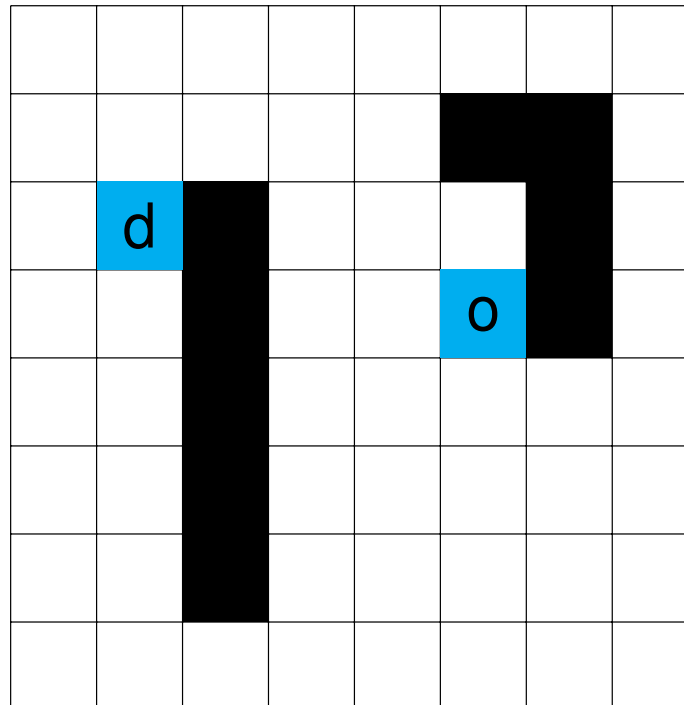


# Trabalho 1: Programação Paralela para Processador Multicore com Memória Compartilhada Usando OpenMP

- **Problema de roteamento usando algoritmo de Lee**

- Grid de  $n \times m$  células
- Célula origem e célula destino
- Obstáculos (ocupam células que não podem ser usadas no roteamento)
- Encontrar menor caminho entre células origem e destino
- Caminho percorre células vizinhas (norte, sul, leste, oeste)



# Programa Sequencial: Estruturas de Dados

- **Grid**: matriz  $n \times m$  de inteiros
  - `dist[i][j]`: distância da origem ate célula  $(i, j)$  do grid
  - Calculado na fase de expansão
- **Fila de células a serem tratadas**: lista encadeada de células  $(i, j)$ 
  - Fila FIFO
  - Usada na fase de expansão
- **Caminho mínimo**: lista encadeada de células  $(i, j)$ 
  - Construído na fase de traceback

# Programa Sequencial: Principal

```
...  
// Lê arquivo de entrada e inicializa estruturas de dados  
inicializa(...) ;  
  
// Fase de expansão: calcula distância da origem até demais células do grid  
achou = expansao() ;  
  
// Se não encontrou caminho de origem até destino  
if (! achou)  
    distancia_min = -1 ;  
else  
{  
    // Obtém distância do caminho mínimo da origem até destino  
    distancia_min = dist[destino.i][destino.j] ;  
    // Fase de traceback: obtém caminho mínimo  
    traceback() ;  
}  
  
// Finaliza e escreve arquivo de saída  
finaliza(...) ;  
...
```

# Programa Sequencial: Inicialização

- **Lê arquivo de entrada:**

- $n^o$  de linhas  $n$  e  $n^o$  de colunas  $m$  do grid
- origem  $(i, j)$
- destino  $(i, j)$
- obstáculos (retângulos)

- **Fila:** vazia

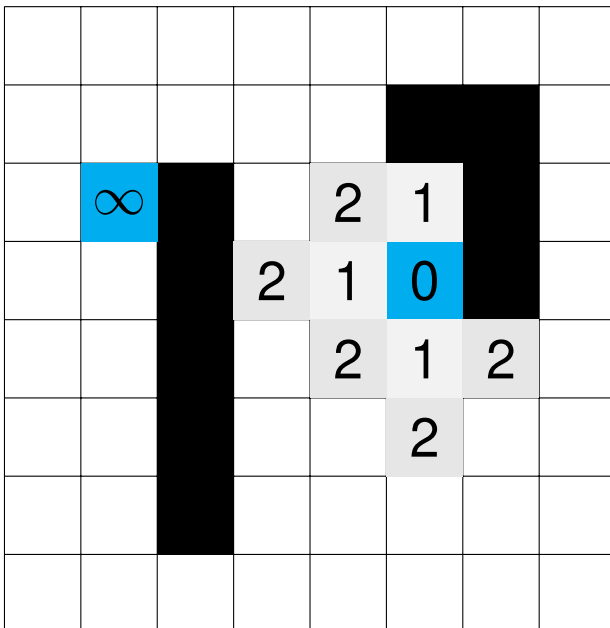
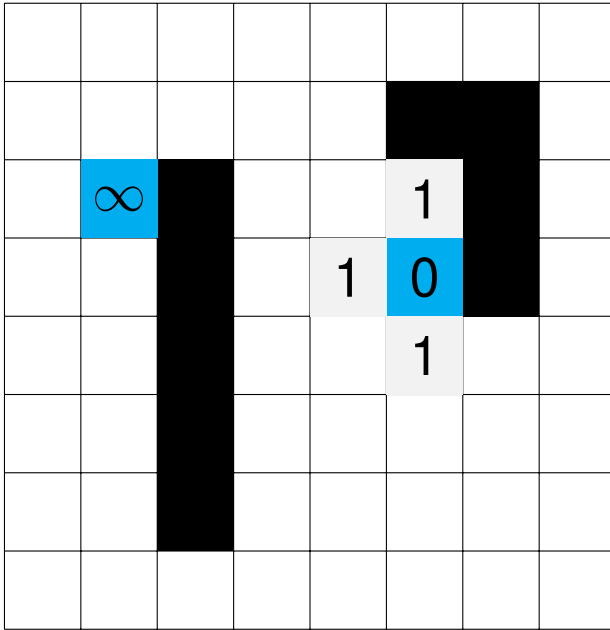
- **Caminho:** vazio

- **Grid:** matriz  $dist$

- Origem: 0
- Obstáculos: -1
- Demais células:  $\infty$   
(inclusive destino)

	0	1	2	...				$m-1$
0	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$
1	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$	-1	-1	$\infty$
2	$\infty$	$\infty$	-1	$\infty$	$\infty$	$\infty$	-1	$\infty$
:	$\infty$	$\infty$	-1	$\infty$	$\infty$	0	-1	$\infty$
	$\infty$	$\infty$	-1	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$
	$\infty$	$\infty$	-1	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$
	$\infty$	$\infty$	-1	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$
$n-1$	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$

## Programa Sequencial: Fase de Expansão



Fila FIFO garante que expansão ocorre por níveis:  
primeiro encontra células com distância 1 da origem,  
depois células com distância 2, ...

# Programa Sequencial: Fase de Expansão (cont.)

				3			
	$\infty$		3	2	1		
			2	1	0		
			3	2	1	2	3
				3	2	3	
					3		

				4			
			4	3			
	$\infty$		3	2	1		
			2	1	0		4
			3	2	1	2	3
			4	3	2	3	4
				4	3	4	
					4		

			5	4	5		
		5	4	3			
	$\infty$		3	2	1		5
			2	1	0		4
			3	2	1	2	3
			4	3	2	3	4
			5	4	3	4	5
				5	4	5	

		6	5	4	5	6	
	6	5	4	3			6
	$\infty$		3	2	1		5
			2	1	0		4
			3	2	1	2	3
			4	3	2	3	4
			5	4	3	4	5
			6	5	4	5	6

	7	6	5	4	5	6	
7	6	5	4	3			6
	7		3	2	1		5
			2	1	0		4
			3	2	1	2	3
			4	3	2	3	4
			5	4	3	4	5
			6	5	4	5	6

## Programa Sequencial: Fase de Expansão

```
achou = false // Destino foi encontrado?
insere_fila(origem) // Insere origem no fim da fila
// Enquanto fila não está vazia e não chegou na célula destino
while ((fila != vazia) && (! achou)) {
    cel = remove_fila() // Remove célula do início da fila
    if (cel.i == destino.i && cel.j == destino.j) // cel é o destino
        achou = true
    else {
        // Para cada um dos 4 possíveis vizinhos da célula (norte, sul, oeste e leste):
        // se célula vizinha existe e ainda não possui valor de distância,
        // calcula distância e insere vizinho na fila de células a serem tratadas
        viz.i = cel.i - 1 ; // Vizinho norte
        viz.j = cel.j ;

        if ((viz.i >= 0) && (dist[viz.i][viz.j] == INT_MAX)) {
            dist[viz.i][viz.j] = dist[cel.i][cel.j] + 1 ;
            insere_fila(viz) ; // Insere viz no fim da fila
        }
        // Vizinho sul, oeste e leste
        ...
    }
}
```

# Programa Sequencial: Fase de Backtracking

	7	6	5	4	5	6	
7	6	5	4	3			6
	7		3	2	1		5
			2	1	0		4
			3	2	1	2	3
			4	3	2	3	4
			5	4	3	4	5
			6	5	4	5	6

	7	6	5	4	5	6	
7	6	5	4	3			6
	7		3	2	1		5
			2	1	0		4
			3	2	1	2	3
			4	3	2	3	4
			5	4	3	4	5
			6	5	4	5	6

	7	6	5	4	5	6	
7	6	5	4	3			6
	7		3	2	1		5
			2	1	0		4
			3	2	1	2	3
			4	3	2	3	4
			5	4	3	4	5
			6	5	4	5	6

	7	6	5	4	5	6	
7	6	5	4	3			6
	7		3	2	1		5
			2	1	0		4
			3	2	1	2	3
			4	3	2	3	4
			5	4	3	4	5
			6	5	4	5	6

...

	7	6	5	4	5	6	
7	6	5	4	3			6
	7		3	2	1		5
			2	1	0		4
			3	2	1	2	3
			4	3	2	3	4
			5	4	3	4	5
			6	5	4	5	6



## Programa Sequencial: Fase de Backtracking

```
insere_caminho(destino) // Insere destino no início do caminho
cel = destino
// Enquanto não chegou na origem
while (cel.i != origem.i || cel.j != origem.j) {
    // Determina se célula anterior no caminho é vizinho norte, sul, oeste ou leste
    // e insere esse vizinho no início do caminho
    viz.i = cel.i - 1 ; // Norte
    viz.j = cel.j ;
    if ((viz.i >= 0) &&
        (dist[viz.i][viz.j] == dist[cel.i][cel.j] - 1))
        insere_caminho(viz) ;
    else {
        viz.i = cel.i + 1 ; // Sul
        viz.j = cel.j ;
        if ((viz.i < n_linhas) &&
            (dist[viz.i][viz.j] == dist[cel.i][cel.j] - 1))
            insere_caminho(viz) ;
        else {
            // Oeste, leste ...
        }
    }
    cel = viz ;
}
```

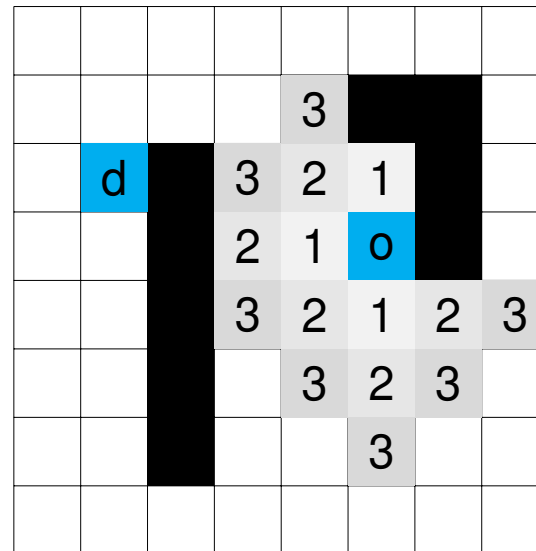
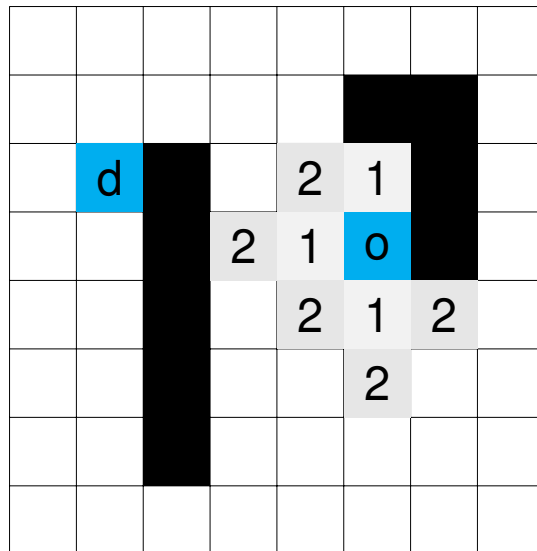
# Programa Sequencial: Finalização

- **Escreve arquivo de saída:**
  - Distância mínima de origem até destino: `dist[destino.i][destino.j]`
  - Caminho mínimo:
    - Sequência de células (i,j) da origem até o destino
- **Obs.:**
  - Pode haver mais de um caminho mínimo
  - Todos são soluções corretas

# Ideias para Paralelização

- **Na fase de expansão:**

- Fila FIFO impõe ordem sequencial de tratamento das células
- Essa restrição é mais “forte” do que o necessário para o correto funcionamento do algoritmo
- Células em um mesmo nível de expansão podem ser tratadas em qualquer ordem (ou ao mesmo tempo)
- Células em diferentes níveis de expansão devem ser tratadas na ordem sequencial



# Ideias para Paralelização

- **Atenção para sincronização:**
  - Identificar estruturas de dados compartilhadas entre threads:
    - Matriz `dist` (grid)? Fila?
  - Usar sincronização apenas se necessário
  - É possível tornar uma estrutura compartilhada em privada, para evitar/reduzir sincronização?
- **Atenção para funções chamadas por threads:**
  - Devem ser **thread-safe**:
    - Função pode ser executada simultaneamente por várias threads, sem possibilidade de levar a inconsistências de dados
    - Pode precisar usar mecanismos de sincronização entre threads
- **Medição de tempo:**
  - Medir tempo de execução dos programas sequencial e paralelo
  - Usar função `omp_get_wtime()`
  - Não incluir na medição leitura e escrita de arquivos de entrada e saída

# Entradas e Saídas do Programa

- **Entradas:**

- Em um único arquivo texto:
  - N<sup>o</sup> de linhas do grid
  - N<sup>o</sup> de colunas do grid
  - Índices i e j da célula origem
  - Índices i e j da célula destino
  - N<sup>o</sup> de obstáculos
  - Para cada obstáculo: (obstáculos são sempre retangulares)
    - Índices i e j da célula inicial do obstáculo (superior esquerda)
    - N<sup>o</sup> de linhas e colunas do obstáculo

- **Saídas:**

- Em um único arquivo texto:
  - Distância mínima da origem até o destino
  - Caminho mínimo: sequência de células (índices i e j) da origem até o destino
- **Na tela: tempo de execução**

# Arquivos de Entrada e Saída Fornecidos

- **Arquivos de entrada fornecidos:**
  - **Entradas 1, 2 e 3:** muito pequenas  
(apenas para depuração, não servem para avaliação de desempenho)
  - **Entrada 9:** pode não ser possível executar, dependendo da configuração  
(quantidade de memória, etc) do computador utilizado

Arquivo	Grid: n <sup>o</sup> de linhas × n <sup>o</sup> de colunas
entrada1.txt	8 × 8
entrada2.txt	20 × 40
entrada3.txt	40 × 80
entrada4.txt	1 mil × 2 mil
entrada5.txt	5 mil × 10 mil
entrada6.txt	10 mil × 20 mil
entrada7.txt	20 mil × 20 mil
entrada8.txt	20 mil × 40 mil
entrada9.txt	40 mil × 80 mil

- **Arquivos de saída correspondentes:** fornecidos  
(uma solução correta, pode haver outras)

## Exemplo: Arquivos entrada1.txt e saida1.txt

### entrada1.txt

```
8 8
3 5
2 1
3
2 2 5 1
1 5 1 2
2 6 2 1
```

$n^o$  de linhas e colunas do grid

índices i e j da origem

índices i e j do destino

$n^o$  de obstáculos

cada obstáculo: índices i e j do início e

...  $n^o$  de linhas e colunas

...

	0	1	2	3	4	5	6	7
0								
1								
2		d						
3						o		
4								
5								
6								
7								

### saida1.txt

```
7
3 5
3 4
3 3
2 3
1 3
1 2
1 1
2 1
```

distância mínima de origem para destino

índices i e j da origem

índices i e j das células seguintes no caminho

...

índices i e j do destino

	7	6	5	4	5	6	
7	6	5	4	3			6
	7		3	2	1		5
			2	1	0		4
			3	2	1	2	3
			4	3	2	3	4
			5	4	3	4	5
			6	5	4	5	6

# Programa a ser Desenvolvido

- **Programa sequencial**: fornecido pronto
- **Desenvolver programa paralelo, usando OpenMP**:
  - **Explorar paralelismo na fase de expansão**
  - **Programa deve ser em C ou C++**
- **Interface de execução dos programas**:
  - Por linha de comando com argumentos:  
`rotseq entrada.txt saida.txt`  
`rotpar entrada.txt saida.txt`
- **Submissão**: um único arquivo .zip com programa fonte paralelo
  - **Programa deve ter no cabeçalho**:
    - Nome dos alunos do grupo: **máximo de 2 alunos**
    - Comando de compilação por linha de comando
  - NÃO submeter executável, programa sequencial, arquivos de entrada/saída, ...
- **Prazo de entrega**: