Universidade Federal de Pernambuco
Centro de Informática
Bach. em Engenharia da Computação

**IF672ec Algoritmos e Estr. de Dados 2019.1**
Prof. Paulo Fonseca

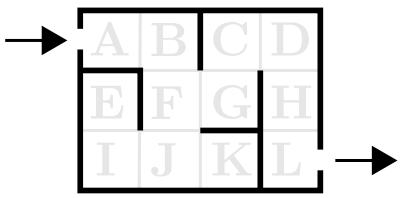
## Segunda Prova — 27 de Junho de 2019

### QUESTÃO 1 (2,0pt)

Steve precisa atravessar um labirinto na forma de uma grade retangular $G$ de $m$ linhas e $n$ colunas de células. O labirinto é separado do exterior por um muro. Só é possível entrar pela célula no canto superior esquerdo e só é possível sair dele pela célula do canto inferior direito. Algumas das $m \times n$ células são separadas entre si por muros internos. Steve gostaria de saber se é possível atravessar o labirinto da entrada para saída, e para isso, lembrou-se do seguinte algoritmo baseado na estrutura de dados ***Union-find***.

Começe com cada célula $X$ numa componente individual. Em seguida, processe as células uma linha por vez, de cima para baixo, e da esquerda para a direita. Em cada célula corrente $X = G[i, j]$, considere a célula vizinha à direita $Y = G[i, j+1]$ e abaixo $Z = G[i+1, j]$, nessa ordem. Se a célula vizinha existe é acessível a partir da célula corrente (isto é, não são separadas por um muro), então una as componentes correspondentes ($Union(X, Y)$, $Union(X, Z)$). É possível atravessar o labirinto se, e somente se, ao final as células de entrada e saída estão na mesma componente.

Ilustre a execução do algoritmo acima sobre o labirinto a seguir, exibindo a Union-find na forma de floresta ao final da execução, supondo a utilização das heurísticas de ***união ponderada*** e ***compressão de caminhos***




### QUESTÃO 2 (2,0pt)

Steve continuou a pensar sobre o problema do labirinto e percebeu que poderia descobrir o ***caminho mais curto*** da origem até cada uma das células, em particular até célula de saída, pensando o labirinto como um grafo. Nesse grafo, cada célula corresponde a um nó, e caso duas células não estejam separadas por um muro, seus nós respectivos são ligados por uma aresta. É preciso então apenas realizar um percurso sobre esse grafo a partir do nó da célula de entrada.

(a) Indique qual o percurso apropriado para resolver esse problema e (b) Considerando o grafo da questão anterior, execute esse percurso exibindo ao final

1. A ordem de visita dos nós
2. O vetor de distâncias $D$
3. O vetor de precursores $F$.

Em cada vértice, considere os seus vizinhos em ordem alfabética.

### QUESTÃO 3 (2,0pt)

Mas afinal, de quantas maneiras é possível atravessar um labirinto — perguntou-se Steve — e pôs-se a pensar sobre algoritmos para computar esse número. Para simplificar um pouco a análise, ele considerou inicialmente o caso em que ***só é permitido mover-se para a direita ou para baixo***. Ele criou o seguinte algoritmo baseado na técnica da **(a)** _____.

Para cada célula $G[i, j]$ computamos o número $D[i, j]$ correspondente à quantidade de caminhos distintos da entrada até essa célula. Os valores $D[i, j]$ devem ser armazenados numa matriz $m \times n$. A resposta final ficará armazenada na posição **(b)** _____.

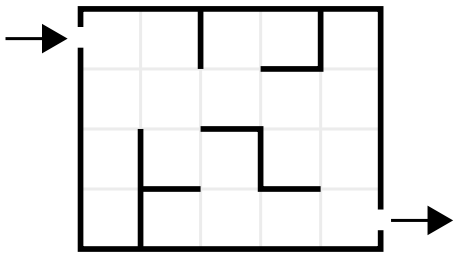
Inicialmente fazemos $D[0, 0] = 1$. Para cada um dos restantes elementos da primeira linha, fazemos **(c)** $D[0, j] =$ _____, se essa célula é acessível a partir da célula da esquerda $D[0, j-1]$. Caso contrário, fazemos **(d)** $D[0, j] =$ _____.

De maneira similar, preenchemos os res-

tantes elementos da primeira coluna. Fazemos **(e)** $D[i,0] =$ _____, se essa célula é acessível a partir da célula acima $D[i-1,0]$. Caso contrário, fazemos **(f)** $D[i,0] =$ _____.

As demais posições são preenchidas uma linha por vez, de cima para baixo, da esquerda para a direita. Iniciamos com $D[i,j] = 0$. Caso a célula seja acessível a partir da célula acima, atualizamos **(g)** $D[i,j] = D[i,j] +$ _____. Em seguida, caso a célula seja accessível a partir da célula à esquerda, atualizamos **(h)** $D[i,j] = D[i,j] +$ _____.

**(i)** Ilustre a execução do algoritmo acima sobre o labirinto a seguir, exibindo a matriz $D$ completamente preenchida.

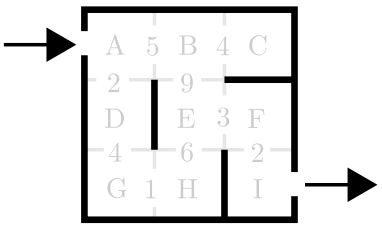


## ■ QUESTÃO 4 (2,0pt)

Steve deparou-se, por fim, com um tipo estranho de labirinto. Desta vez, algumas células são separadas por muros intransponíveis, como nos casos anteriores. Porém algumas outras células são separadas por muretas que podem ser escaladas e atravessadas. Cada mureta tem uma altura própria, e para transpô-la, Steve precisa gastar uma energia proporcional a essa altura. Dessa forma, o interesse passa a ser atravessar o labirinto gastando a menor quantidade de energia possível, ao invés de simplesmente percurso mais curto.

Benditas aulas de Algoritmos, pensou Steve! O ***Algoritmo de Dijkstra*** vai ajudar a resolver o problema. Para tal, basta imaginar o labirinto como um grafo, como na Questão 2, e usar as alturas das muretas como pesos das arestas.

Considere então o labirinto abaixo. Os números entre as células correspondem às alturas das muretas que as separam. Uma linha preta corresponde a um muro que não pode ser atravessado.

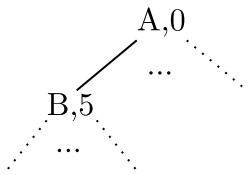



Ilustre a execução do Algoritmo sobre o labirinto acima, completando o quadro abaixo.

| Iter. # | Peso, Precursor A | B | C | ··· | H | I |
|---|---|---|---|---|---|---|
| 0 | $0,-$ | $\infty,?$ | $\infty,?$ | ··· | $\infty,?$ | $\infty,?$ |
| 1 | $(0,-)$ | $5,A$ | ··· | ··· | ··· | ··· |
| ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ |

## ■ QUESTÃO 5 (2,0pt)

Considerando ainda o problema da Questão 4, Steve ponderou que o poderia resolvê-lo com um algoritmo de ***backtracking***. Basta iniciar o caminho com o vértice/célula de entrada e ir acrescentando vértices vizinhos progressivamente ao caminho, atualizando, a cada vértice acrescentado, a soma parcial dos pesos. Assim que a célula de saída for alcançada, temos uma possível travessia do labirinto e seu respectivo custo. Como o caminho mínimo não pode ter ciclos, o backtacking ocorre sempre que não for possível prosseguir sem repetir uma célula já visitada, ou se o custo parcial do caminho exceder o custo da melhor solução já conhecida. Ilustre a execução desse algoritmo sobre o labirinto da Questão 4, exibindo a árvore de percurso do espaço de soluções. Indique claramente na árvore a solução ótima encontrada e os pontos de backtracing.




■ ■ ■