Percursos e Conectividade em Grafos
Breadth-First Search

Fernando Lobo

Algoritmos e Estrutura de Dados

Percursos e conectividade em Grafos

Vamos ver algoritmos que:

- visitam os nós de um grafo de uma forma sistemática.
- determinam se existe um caminho entre dois nós.

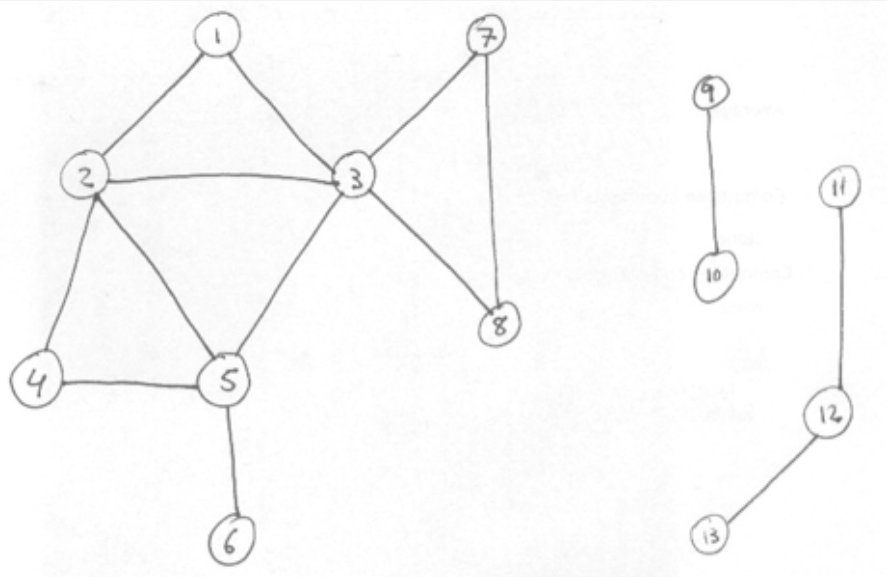
Percursos e conectividade em Grafos

Dois algoritmos essenciais:

- BFS $\rightarrow$ Breadth-First Search (pesquisa em largura).
- DFS $\rightarrow$ Depth-First Search (pesquisa em profundidade).

Nesta aula vamos aprender o BFS. Na próxima aula veremos o DFS.

BFS – pesquisa em largura

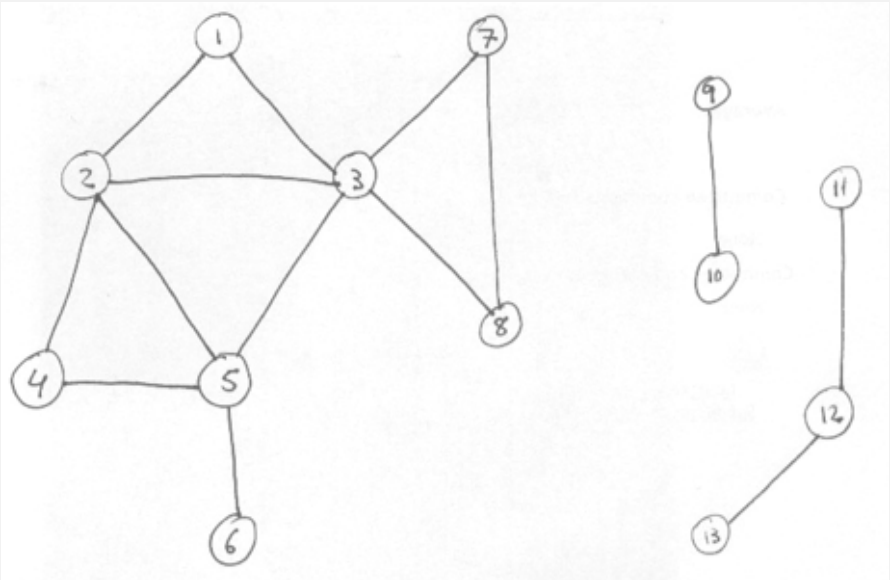


Dado um nó inicial s, BFS visita os outros nós por camadas: L_1, L_2, L_3, ..., como se fosse uma onda que avança a partir do nó inicial.

BFS

- L_1: nós que estão a uma distância 1 de s.
- L_2: nós que estão a uma distância 2 de s.
- L_3: nós que estão a uma distância 3 de s.
- ...

BFS



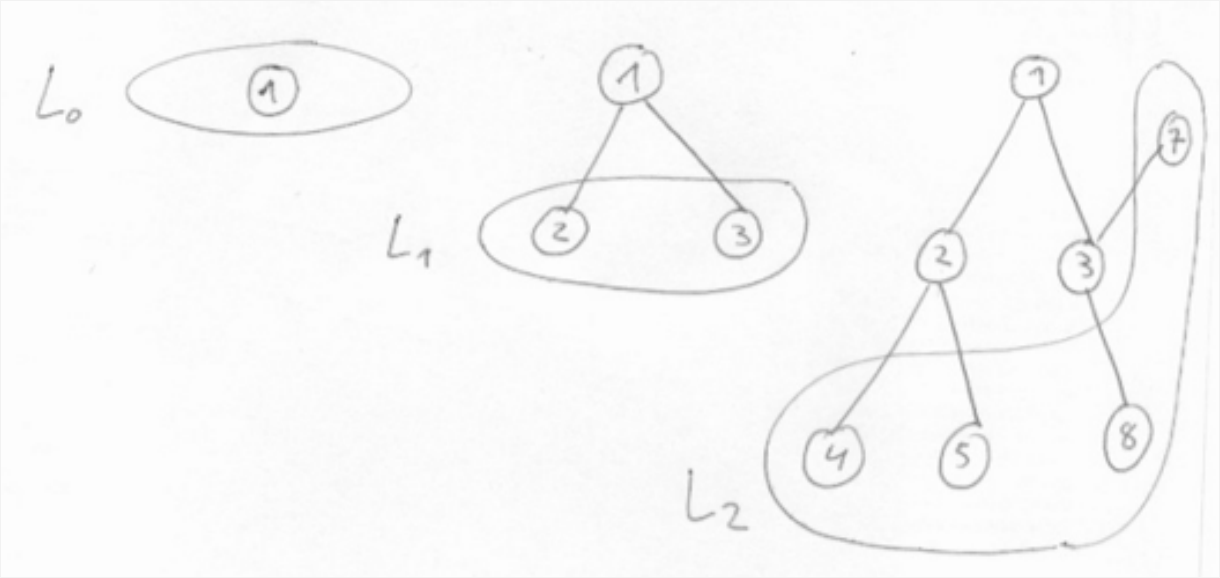
Se s for o nó 1, então:

- L_0: $\{1\}$
- L_1: $\{2, 3\}$
- L_2: $\{4, 5, 7, 8\}$
- L_3: $\{6\}$

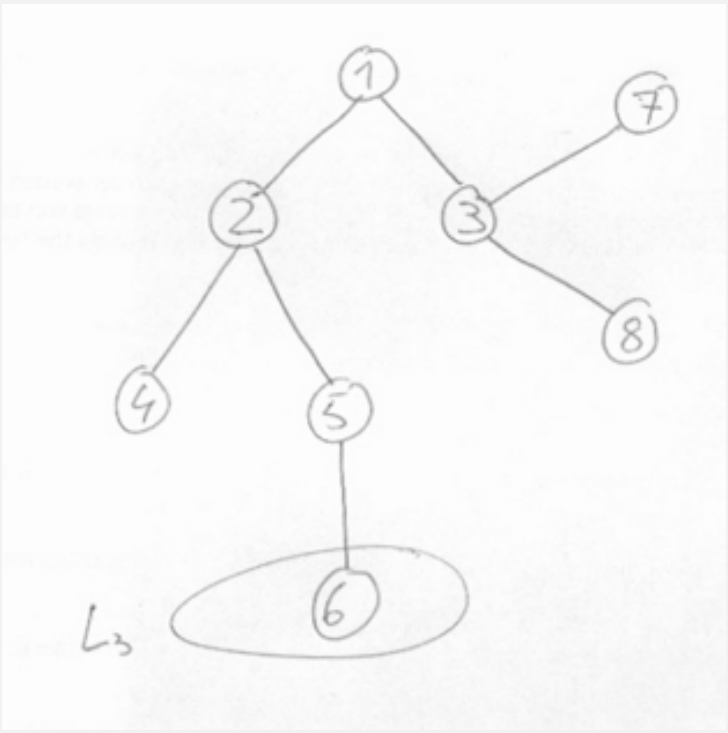
BFS

- Formalmente:
- $L_0 = \{s\}$
- Assumindo que temos $L_0, L_1, L_2, \ldots, L_j$, então L_{j+1} consiste no conjunto de nós que não pertencem a nenhuma das camadas anteriores, e que têm um arco para um nó da camada L_j.

L_0, L_1, L_2



L_3



BFS

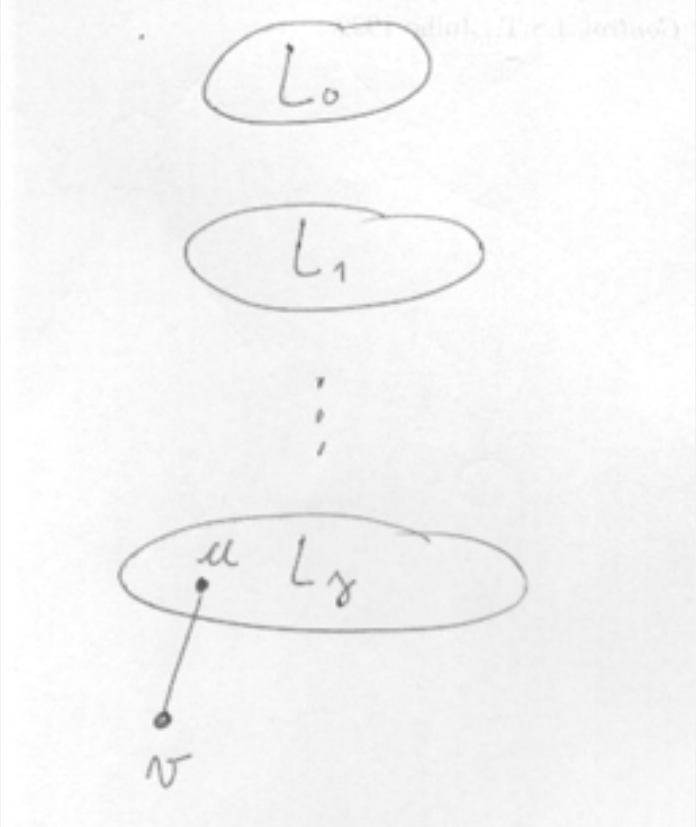
- L_j contém os nós que estão a uma distância j de s.
- Se um nó não aparece em nenhuma das camadas, então não existe um caminho de s para esse nó.

Em resumo

BFS determina o conjunto de nós para os quais existe um caminho a partir de um nó de origem s, e também calcula os caminhos mais curtos de s para esses nós.

BFS Tree

- BFS produz uma árvore (BFS Tree) com o nó s na raíz, e que contém todos os nós para os quais existe um caminho a partir de s.
- Para cada nó v ($\neq s$), consideremos o momento em que v é "descoberto" pelo algoritmo BFS. Isso acontece quando $u \in L_j$ está a ser examinado e verificamos que existe um arco (u, v) em que v ainda não foi visitado.
- Nesse momento adicionamos (u, v) à árvore (u é pai de v).



Implementação de BFS

- Usamos uma fila FIFO para guardar os nós que estão na "crista da onda".
- Cada nó tem um atributo booleano *visited*, que indica se o nó já foi visitado.
- (Esta implementação é ligeiramente diferente da que está no livro CLRS. Em vez do atributo *visited*, o livro usa um atributo *color* que pode assumir os valores WHITE, GRAY, BLACK.)

Implementação de BFS

- Input: um grafo G e um nó de partida s.
- Output:
  - $v.d \rightarrow$ distância de s para v, $\forall_v \in V$.
    (distância é o número mínimo de arcos de s para v).
  - $v.\pi \rightarrow$ pai de v (na BFS Tree), $\forall_v \in V$.

Pseudocódigo

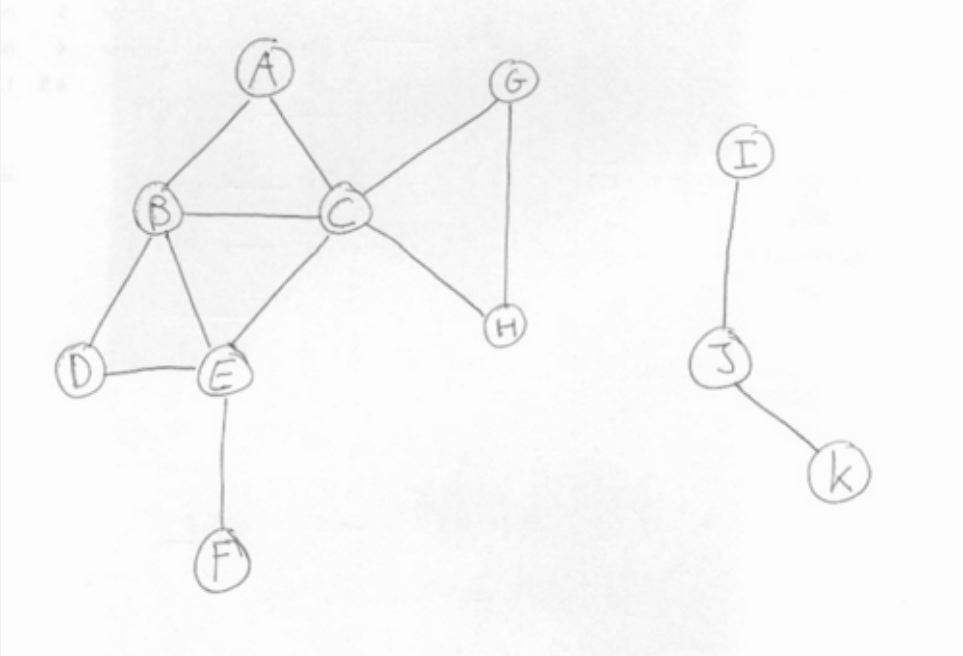
$\text{BFS}(G, s)$
  for each $u \in G.V - \{s\}$
    $u.visited = \text{FALSE}$
    $u.d = \infty$
    $u.\pi = \text{NIL}$
  $s.visited = \text{TRUE}$
  $s.d = 0$
  $s.\pi = \text{NIL}$
  $Q = \emptyset$
  $\text{ENQUEUE}(Q, s)$
  $\vdots$

Continua na próxima folha.

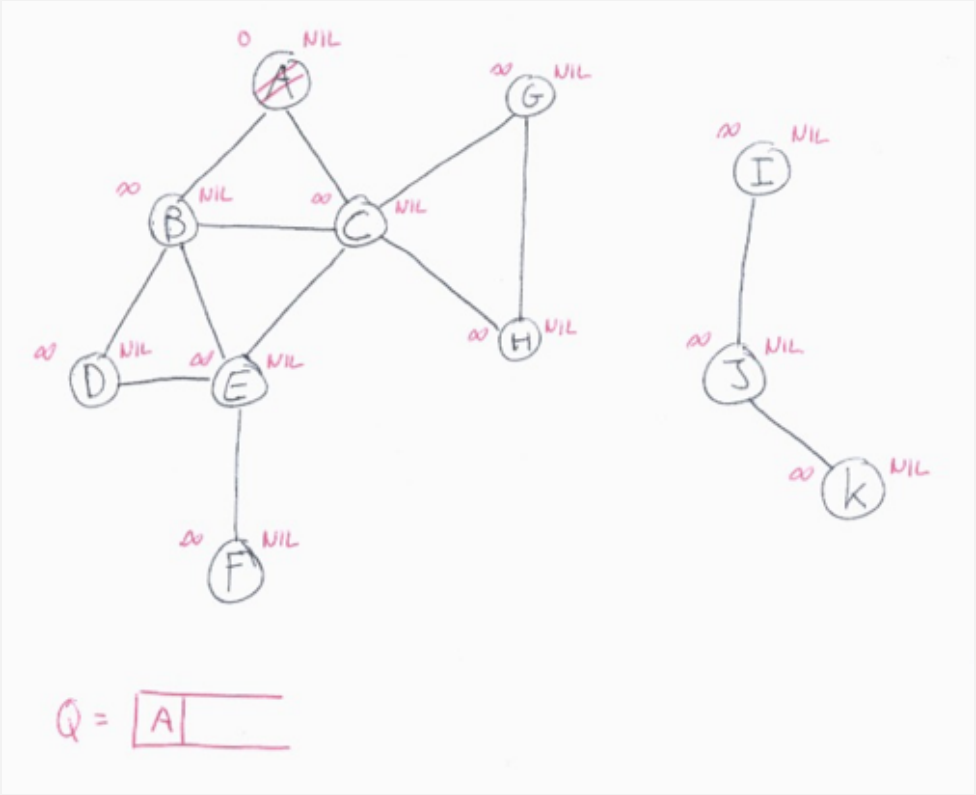
Pseudocódigo (cont.)

```
BFS(G, s)
    ⋮
    while Q ≠ ∅
        u = DEQUEUE(Q)
        for each v ∈ G.Adj[u]
            if not v.visited
                v.visited = TRUE
                v.d = u.d + 1
                v.π = u
                ENQUEUE(Q, v)
```

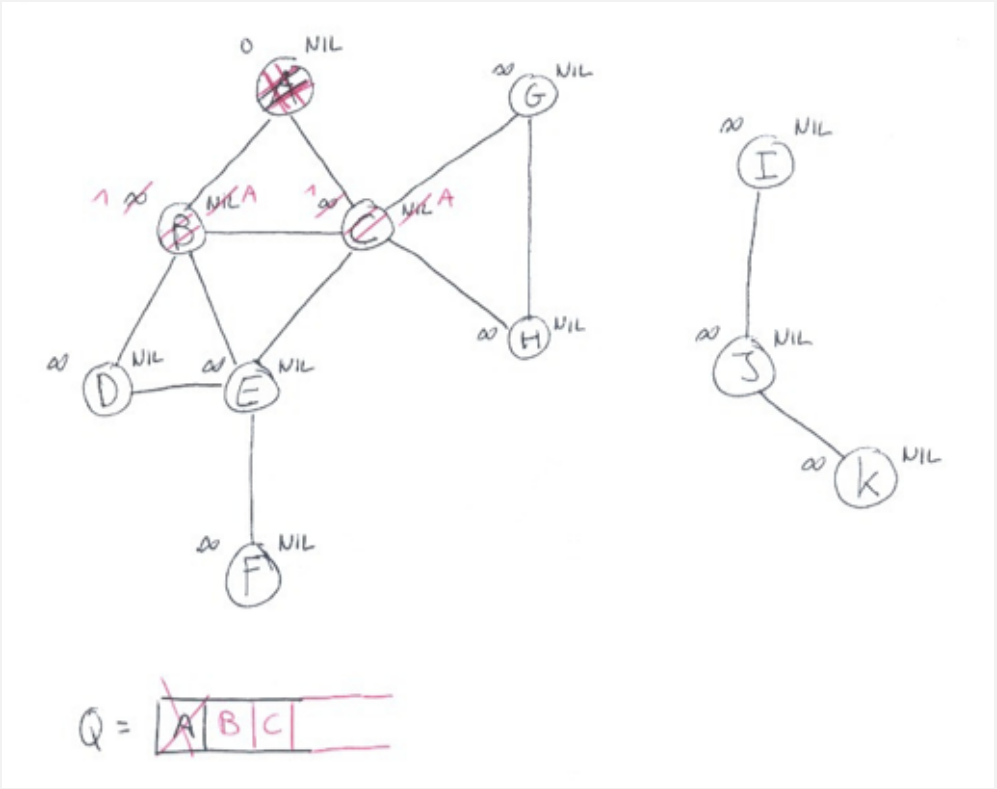

Exemplo de execução: s é o nó A



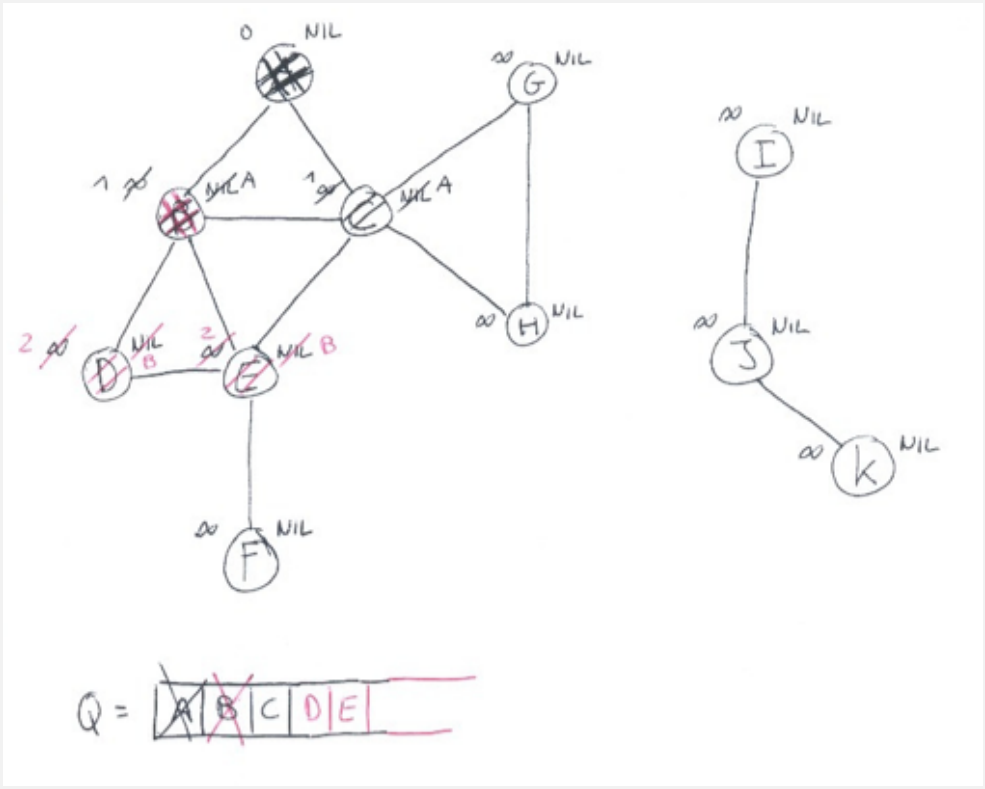
Exemplo de execução



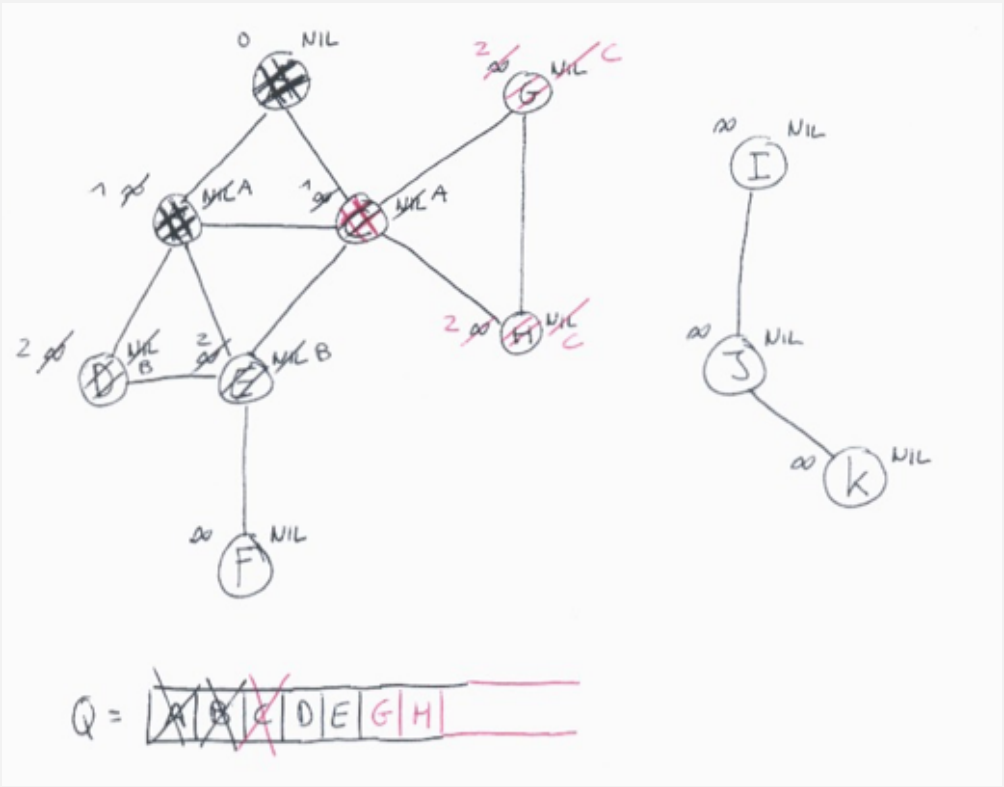
Exemplo de execução



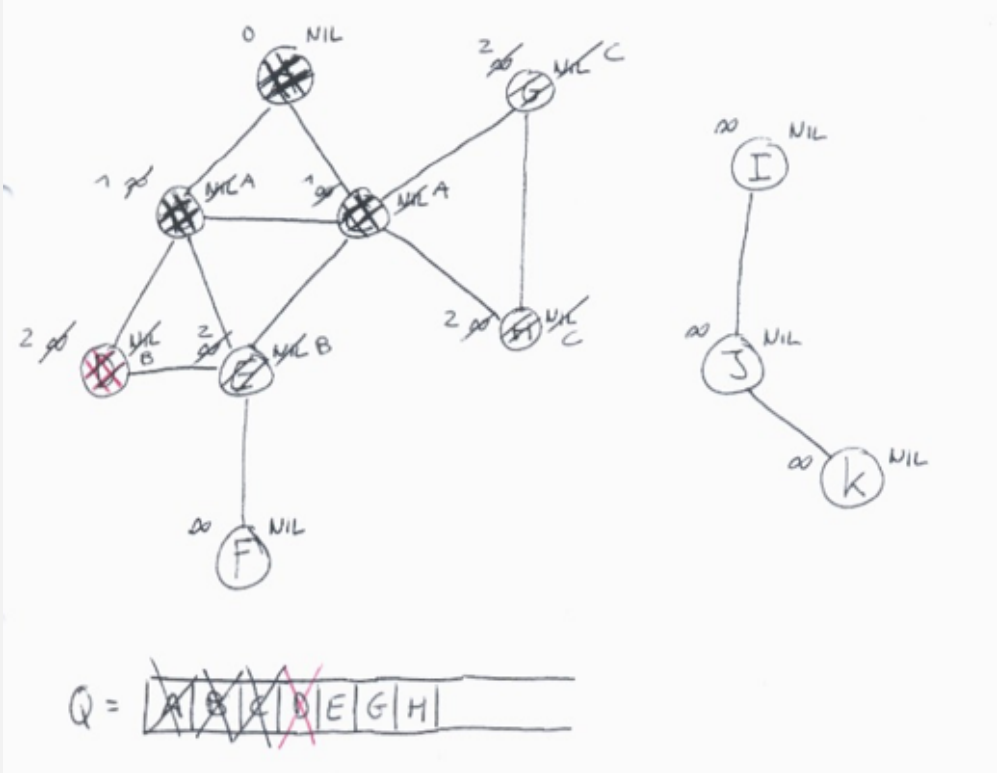
Exemplo de execução



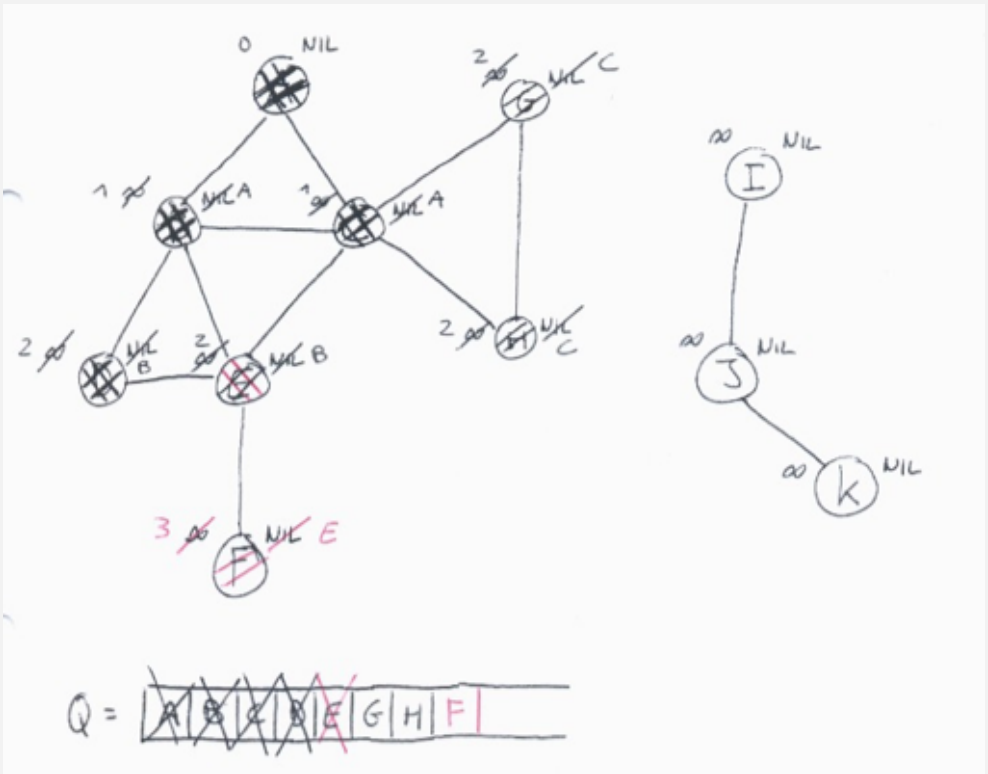
Exemplo de execução



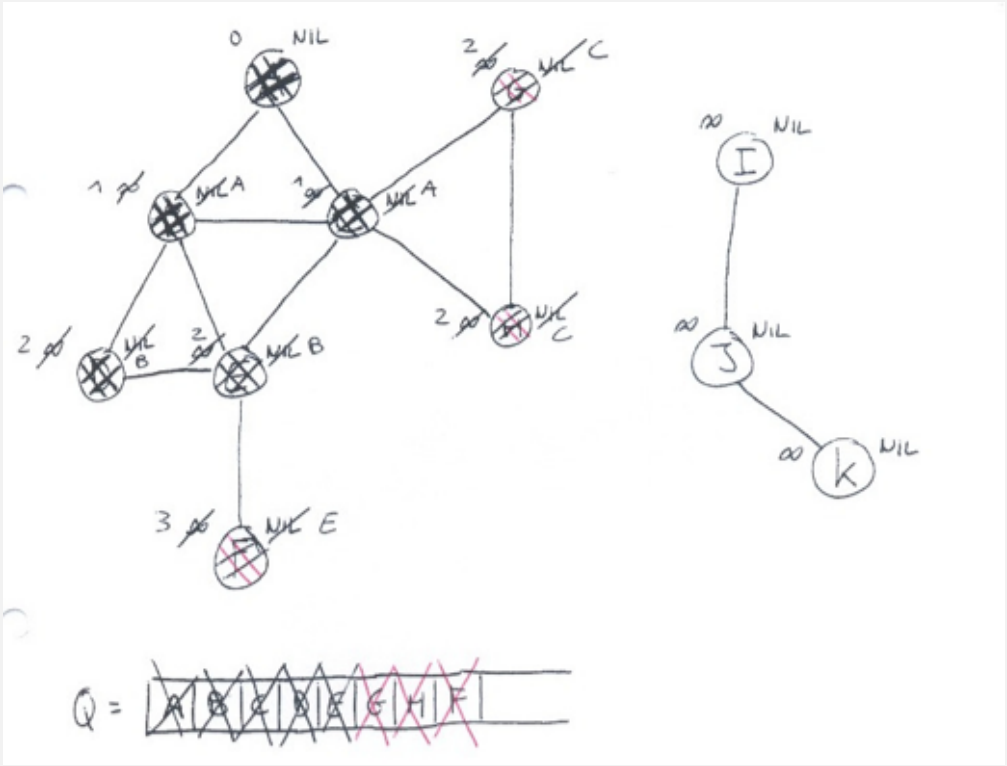
Exemplo de execução



Exemplo de execução



Exemplo de execução



Implementação de BFS

- BFS pode não visitar todos os nós do grafo (basta o grafo não ser conexo). No exemplo, os nós I, J, K não são visitados.
- Depois de correr BFS, obtemos a BFS Tree através da informação contida no atributo π dos nós.
- O conjunto de arcos $\{(v.\pi, v) : v \neq s\}$ dá-nos a BFS Tree.
- No exemplo, {(A,B),(A,C),(B,D),(B,E),(C,G),(C,H),(E,F)}.

Complexidade de BFS

- Cada nó entra na fila Q uma (ou zero) vezes. Também só sai da fila uma vez.
- ENQUEUE e DEQUEUE $= \Theta(1)$.
- Logo, o tempo de operações na fila $= O(V)$.
- A lista de adjacência de cada nó só é percorrida quando o nó sai da fila (no máximo uma vez). Como a soma do tamanho de todas as listas de adjacência $= \Theta(E)$, gasta-se $O(E)$ a percorrer as listas.
- Custo de inicialização $= \Theta(V)$.
- Tempo Total $= O(V + E)$.

Caminho mais curto

- Após correr BFS, podemos determinar o caminho mais curto entre s e um outro nó v.

```
PRINT-PATH(G, s, v)
  if v == s
      print s
  elseif v.π == NIL
      print "No path from" s "to" v
  else PRINT-PATH(G, s, v.π)
      print v
```

Componentes conexas

- O conjunto de nós descobertos por BFS é o conjunto de nós que estão ligados a s por algum caminho.
- Esse conjunto de nós (seja ele R) é uma componente conexa de G que contém s.
- Para saber se existe um caminho de s para t, basta verificar se $t \in R$.
- Se corrermos BFS a partir de outro nó (que nunca foi visitado), obtemos outra componente conexa.
- Facilmente se modifica o código de BFS para obtermos todas as componentes conexas do grafo.