

Algoritmos e Estruturas de Dados III

CIC210

Algoritmos em Grafos - Busca em Grafos

Haroldo Gambini Santos

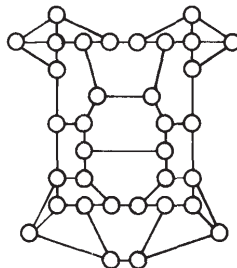
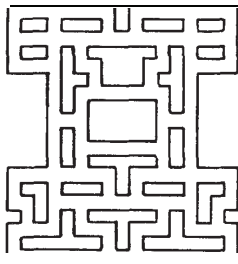
Universidade Federal de Ouro Preto - UFOP

28 de setembro de 2009

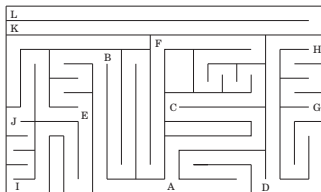
Atravessando Labirintos



Atravessando Labirintos

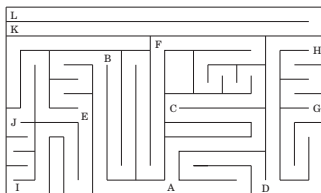


Busca em Grafos

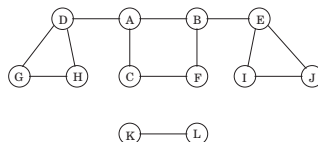


Um labirinto

Busca em Grafos



Um labirinto

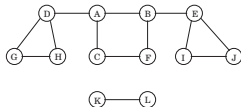


Grafo correspondente

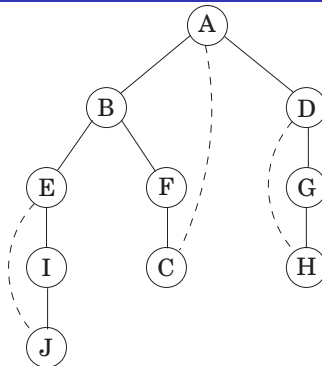
Exploração de Grafos - Partindo de uma fonte

- 1 /* inicializado em algum outro ponto do programa ou somente na primeira chamada recursiva de dfs */
 $visitado[v] = falso \quad \forall v \in V;$
- 2 função **explore**($V, A, s, visitado$)
 Saída: vetor *visitado* com informação para cada cada nó em V
 se o mesmo pode ser alcançado a partir do nó s
- 3 $visitado[s] = verdadeiro;$
- 4 **para todo** $(s, d) \in A$ **faça**
- 5 **se** $(visitado[d] = falso)$ **então**
- 6 **explore**($V, A, d, visitado$);
- 7 **fim**
- 8 **fim**

Exploração de Grafos - Partindo de uma fonte



Grafo de entrada



Floresta do percorrimento de **explora** com $s = A$

- linhas normais: **arestas da árvore** (AV): (u, v) é uma AV se v foi descoberto primeiro pela exploração de (u, v)
- linhas tracejadas: **arestas de retorno**: arestas que levam a um nó já conhecido

Busca em Profundidade - *Depth-First Search*

```
1 função dfs( $V, A, visitado$ )  
2  $visitado[i] = falso \quad \forall i \in V$ ;  
3 para todo  $v \in V$  faça  
4     se ( $visitado[v] = falso$ ) então  
5         explore( $V, A, v, visitado$ );  
6     fim  
7 fim
```

Busca em Profundidade - *Depth-First Search*

```
1 função dfs( $V, A, visitado$ )  
2  $visitado[i] = falso \quad \forall i \in V$ ;  
3 para todo  $v \in V$  faça  
4     se ( $visitado[v] = falso$ ) então  
5         explore( $V, A, v, visitado$ );  
6     fim  
7 fim
```

Complexidade de Tempo

$$O(|V| + |A|)$$

Busca em Profundidade - *Depth-First Search*

```
1 função dfs( $V, A, visitado$ )  
2  $visitado[i] = falso \quad \forall i \in V$ ;  
3 para todo  $v \in V$  faça  
4     se ( $visitado[v] = falso$ ) então  
5         explore( $V, A, v, visitado$ );  
6     fim  
7 fim
```

Complexidade de Tempo

$$O(|V| + |A|)$$

Note que cada aresta é examinada duas vezes.

Busca em Profundidade

Estado dos vértices durante a busca

- Visitação:

E1: Concluída

E2: Em Execução

E3: Vértice Ainda Desconhecido

Propriedades

Busca em Profundidade

Estado dos vértices durante a busca

- Visitação:

E1: Concluída

E2: Em Execução

E3: Vértice Ainda Desconhecido

Propriedades

- Nunca se encontrará uma aresta apontando para qualquer vértice em **E1**

Busca em Profundidade

Estado dos vértices durante a busca

- Visitação:

E1: Concluída

E2: Em Execução

E3: Vértice Ainda Desconhecido

Propriedades

- Nunca se encontrará uma aresta apontando para qualquer vértice em **E1**
- Chamada recursiva será feita sempre que um vértice em **E3** for encontrado

Busca em Profundidade

Estado dos vértices durante a busca

- Visitação:

E1: Concluída

E2: Em Execução

E3: Vértice Ainda Desconhecido

Propriedades

- Nunca se encontrará uma aresta apontando para qualquer vértice em **E1**
- Chamada recursiva será feita sempre que um vértice em **E3** for encontrado
- Arestas para nós em **E2** são aquelas que conectam o vértice corrente com algum vértice do caminho do mesmo para a raiz

Busca em Profundidade

Categorizando o Grafo

Ciclos: existirão sempre que uma **Aresta de Retorno** for encontrada

Componentes Conexos: cada chamada não recursiva para **explore** irá processar uma árvore diferente

Busca em Amplitude

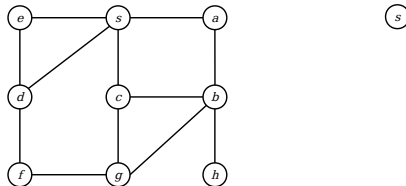
Funcionamento

- Atuação em camadas:
 - inicialmente são considerados os nós com distância 0 (o nó fonte s);
 - o algoritmo procede: na iteração 1 são encontrados os nós com distância 1; prosseguindo, de modo genérico, na iteração d será adicionada uma camada com todos os nós com distância d ;
 - cada novo nó descoberto é adicionado no final de uma fila Q (operação *enfileira*);
 - os nós dessa fila são removidos, com a operação *pegaPrimeiroFila*.

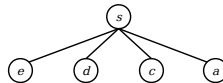
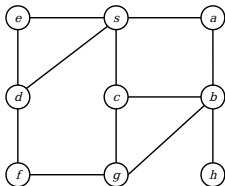
Busca em Amplitude - *Breadth-first Search*

```
1 função bfs( $V, A, s$ )
2    $dist[i] = \infty \quad \forall i \in V$ ;
3    $dist[s] = 0$ ;
4   inicializaFila( $Q$ );
5   enfileira( $Q, s$ );
6   enquanto tamanhoFila( $Q$ ) > 0 faça
7      $u = \textit{pegaPrimeiroDaFila}(Q)$ ;
8     para todo  $(u, v) \in A$  faça
9       se  $dist[v] = \infty$  então
10         enfileira( $Q, v$ );
11          $dist[v] = dist[u] + 1$ ;
12     fim
13 fim
14 fim
```

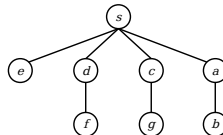
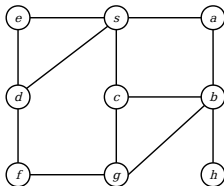
Execução de Exemplo



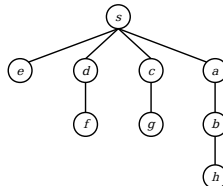
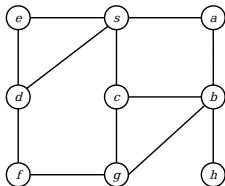
Execução de Exemplo



Execução de Exemplo



Execução de Exemplo



Busca em Amplitude

Corretude:

- em uma iteração d todos os nós com distância $\leq d$ tem suas distâncias corretamente calculadas e todos os outros nós tem distância $=\infty$, a fila contém exatamente os nós com distância d ;

Complexidade

- $O(|V| + |E|)$
 - $2 \times |V|$ operações na fila
 - arestas processadas $2 \times |E|$ (grafos não direcionados)

DFS × BFS

DFS - Busca em Profundidade

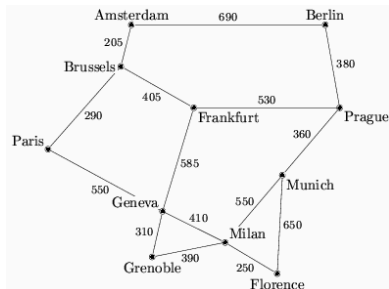
- incursões **profundas** no grafo, voltando somente quando não existem mais nós desconhecidos pela frente;
- boas propriedades já discutidas;
- possível problema: levar muitas iterações para encontrar nó próximo;
- uso de pilha.

BFS - Busca em Amplitude

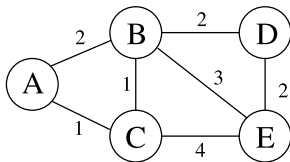


- busca progride em “largura”:
certifica-se de que vizinhos próximos sejam visitados primeiro;
- sem reinício: interessam apenas os nós alcançáveis a partir de s
- uso de fila.

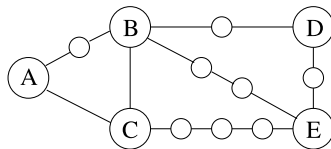
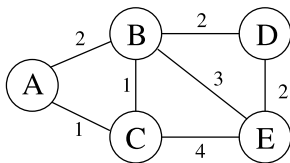
Grafos com Pesos - Computando caminhos mínimos



Usando BFS



Usando BFS



Algoritmo do Alarme

- Coloque o alarme do nó s para o tempo 0
- Repita enquanto houverem alarmes:
 - pegue o nó u com alarme no menor tempo t
 - para cada vizinho v de u faça:
 - se não houver alarme para v ajuste o alarme de v para $t + dist(u, v)$
 - se houver alarme para v e for maior que o tempo $t + dist(u, v)$, ajuste o mesmo para esse tempo menor

Algoritmo do Alarme

- Computa caminhos mínimos para qualquer grafo com pesos positivos
- Como implementar sistema de alarmes ?

Sistema de Alarmes

Fila de Prioridades - operações que necessitamos

- `inicializa(heap, alarmes)`
- `diminui(heap, elemento, alarmeMenor)`
- `pegaMenor(heap)`

Algoritmo de Dijkstra

```
1  função dijkstra( $V, A, d, s$ )
2  /*  $d_a$  indica a distância do arco (ou aresta)  $a \in A$  */
   Saída:  $dist_u$  e  $prev_u$  : comprimento e antecessor do nó  $u$  no caminho mínimo
          $s \rightarrow u$ 
3  para todo  $u \in V$  faça
4       $dist_u = \infty$ ;
5       $prev_u = \text{nulo}$ ;
6  fim
7   $dist_s = 0$ ;
8  inicializa(heap, dist);
9  enquanto tamanho(heap) > 0 faça
10      $v = \text{pegaMenor}(heap)$ ;
11     para todo  $u : (v, u) \in A$  faça
12         se  $dist_u > dist_v + d_{(v,u)}$  então
13              $dist_u = dist_v + d_{(v,u)}$ ;
14              $prev_u = v$ ;
15             diminui(heap, u,  $dist_u$ );
16     fim
17 fim
18 fim
```

Complexidade

- Estruturalmente idêntico ao BFS
- Operação mais difícil em fila: sem as operações em tempo constante `pegaPrimeiroDaFila` e `enfileira`
- Operações:
 - $|V|$ operações de `pegaMenor`
 - $|V| + |E|$ operações de `diminui`
 - *Heap* binária: `diminui` fica $O(\log |V|)$

Caminhos Mínimos entre Todos os Pares

$dist(i, j, k)$: o comprimento do caminho mais curto entre i e j tal que apenas os nós $\{1, \dots, k\}$ podem ser usados como intermediários.

Relação de recorrência

$$dist(i, j, k) = \min\{dist(i, k, k-1) + dist(k, j, k-1), dist(i, j, k-1)\}$$

O Algoritmo de Floyd-Warshall

```
1  função fwarshall( $V, A, d, s$ )
2  para todo  $u \in V$  faça
3      para todo  $v \in V$  faça
4           $dist(u, v, 0) = \infty$ ;
5      fim
6  fim
7  para todo  $(u, v) \in A$  faça
8       $dist(u, v, 0) = d_{(u,v)}$ ;
9  fim
10 para  $k = 1$  até  $|V|$  faça
11     para  $i = 1$  até  $|V|$  faça
12         para  $j = 1$  até  $|V|$  faça
13              $dist(i, j, k) =$ 
14                  $\min\{dist(i, k, k-1) + dist(k, j, k-1), dist(i, j, k-1)\}$ ;
15         fim
16     fim
17 fim
```