Teoria dos Grafos - BCC 204

Haroldo Gambini Santos

Universidade Federal de Ouro Preto - UFOP

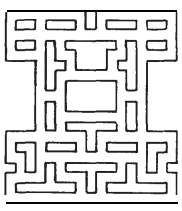
27 de março de 2011

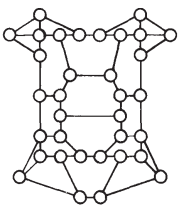
Notas

Atravessando Labirintos

Notas

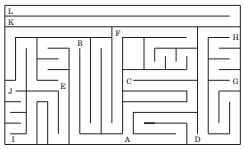
Atravessando Labirintos



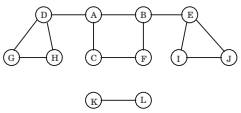


Notas

Busca em Grafos



Um labirinto



Grafo correspondente

Notas

Como não se perder ?

Método do século XIX

- considere que o caminhante possui uma linha enrolada suficientemente grande;
- considere a existência de **lâmpadas**, inicialmente desligadas, em cada **interseção**;
- **portas**, inicialmente fechadas, ao início de cada **ligação** com outra interseção;
  - as portas tem uma janela por onde é possivel ver se na interseção da outra extremidade há uma lâmpada acesa.
- objetivo: **acender** todas as lâmpadas e **abrir** todas as portas.

Notas

Como não se perder ?

Método do século XIX (cont.)

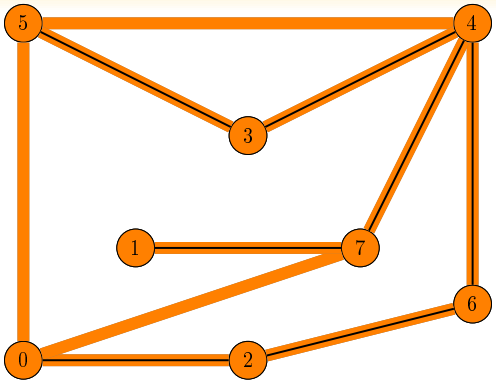
i a intersecção corrente não apresenta portas fechadas, vá para **iii**; senão, abra qualquer porta fechada levando para outra intersecção;

ii caso veja que a intersecção do outro lado da ligação está com a luz acessa, tente outra porta (passo **i**); caso contrário, siga o caminho, desenrolando a linha, ligue a luz e vá parao passo (i);

iii se todas as portas da intersecção corrente estão fechadas, cheque se você está no ponto inicial; se sim, **pare**; caso contrário, use a linha para voltar pela passagem que lhe trouxe até a intersecção corrente e vá para o passo **i**.

Notas

Exemplo



Notas

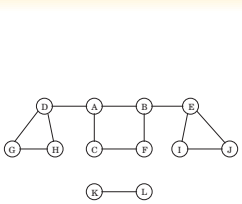
Exploração de Grafos - Partindo de uma fonte

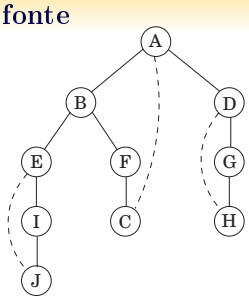
1 /* inicializado em algum outro ponto do programa ou somente na primeira chamada recursiva de dfs */
$visitado[v] = falso \quad \forall v \in V$;

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**

Notas

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

Notas

Busca em Profundidade - Depth-First Search

```
1 função dfs(V, A, visitado)
2 visitado[i] = falso  ∀i ∈ V;
3 para todo v ∈ 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.

Notas

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

Notas

Busca em Profundidade

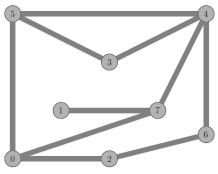
Categorizando o Grafo

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

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

Notas

Chamadas Recursivas



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


Notas

Exercícios

1. Qual a complexidade assintótica para DFS quando usada uma lista de adjacências ?
2. Qual a complexidade assintótica para DFS quando usada uma matriz de adjacências ?
3. Faça um esquema com as chamadas recursivas de DFS considerando o grafo com as arestas abaixo:
   `3-7 1-4 7-8 0-5 5-2 3-8 2-9 0-6 4-9 2-6 6-4`
4. Para o grafo anterior, desenhe a floresta de percorrimento, indicando arestas de árvore e arestas de retorno.

Notas

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*.

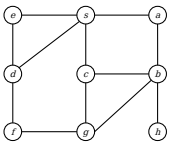
Notas

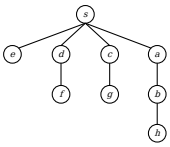
Busca em Amplitude - Breadth-first Search

```
1  função bfs(V, A, s)
2  dist[i] = ∞  ∀i ∈ V;
3  dist[s] = 0;
4  inicializaFila(Q);
5  enfileira(Q, s);
6  enquanto tamanhoFila(Q) > 0 faça
7      u = pegaPrimeiroDaFila(Q);
8      para todo (u, v) ∈ A faça
9          se dist[v] = ∞ então
10             enfileira(Q, v);
11             dist[v] = dist[u] + 1;
12         fim
13     fim
14 fim
```

Notas

Execução de Exemplo





Notas

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=∞, 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)

Notas

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

Notas

Notas

Notas