Grafos

Busca

Percorrendo um grafo

- Percorrer um grafo é uma tarefa fundamental
- Pense no caso de se procurar uma certa informação associada a um vértice/aresta num grafo
- Deve-se ter uma forma sistemática de visitar as arestas e os vértices
- O algoritmo deve ser suficientemente flexível para se adequar à diversidade de grafos

Percorrendo um grafo

- **Eficiência**: Não deve haver repetições (desnecessárias) de visitas a um vértice e/ou aresta

- **Correção**: Todos os vértices e/ou arestas devem ser visitados, se o objetivo for passar por todos

Algoritmo básico de busca em grafo

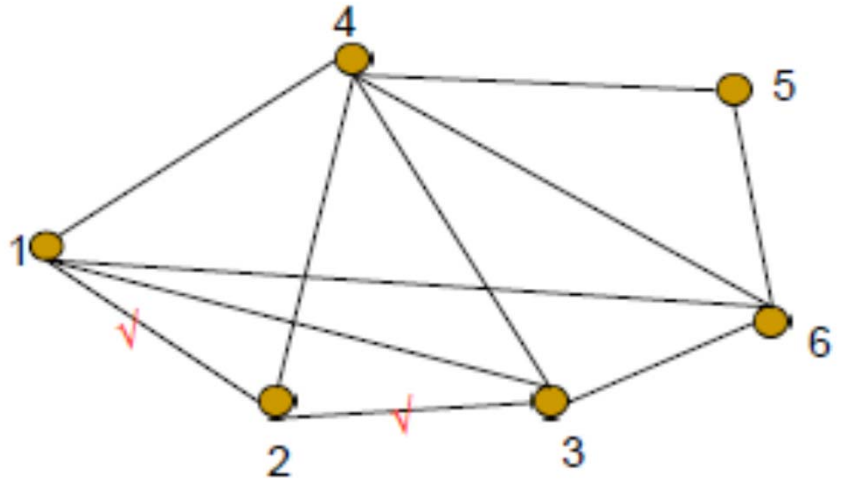
- Utiliza o conceito de marcar os vértices, de modo a registrar que ele já foi visitado.
- Seja G um grafo conexo em que todos vértices não estão marcados (não foram ainda visitados)
- Passo Inicial:
  - escolher e marcar um vértice arbitrário v;
- Passo Geral:
  - selecionar (explorar) uma aresta (v,w) incidente a um vértice marcado v e que não tenha sido selecionada anteriormente
  - Se w é não marcado, marca-se w
- O processo termina quando todas as arestas de G tiverem sido selecionadas

Algoritmo Geral de Busca num Grafo Conexo

- Dado um Grafo (V,A) conexo:

```
início
        escolher e marcar um vértice inicial;
        enquanto existir algum vértice v marcado e
        incidente a uma aresta (v,w) não explorada,
        faça escolher o vértice v e explorar
                (marcar) a aresta (v,w)
                se w é não marcado
                então marcar w
fim
```

Exemplo



Vértice inicial: 1 (raiz da busca)

Marca (v)	Explora (aresta)
1	(1,2)
2	(2,3)

.....

Busca em grafos

- Existem vários tipos de busca que podemos realizar em um grafo. Os três principais:
  - Busca em profundidade
  - Busca em largura
  - Busca pelo menor caminho

Busca em largura
BFS – *Breadth First Search*

Funcionamento

- Partindo de um vértice inicial, a busca explora todos os vizinhos de um vértice. Em seguida, para cada vértice vizinho, ela repete esse processo, visitando os vértices ainda inexplorados

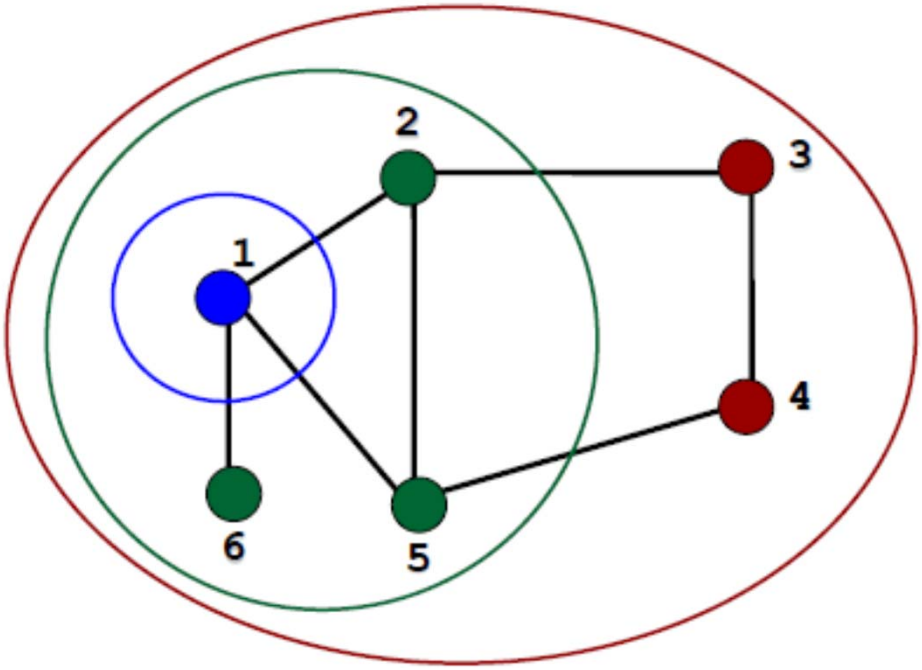
- Em outras palavras, esse tipo de busca se inicia em um vértice e então visita todos os seus vizinhos antes de se aprofundar na busca. Esse processo continua até que
  - o alvo da busca seja encontrado
  - não existam mais vértices a serem visitados.

Busca em largura

- Dentre todos os vértices marcados e incidentes a alguma aresta ainda não explorada, escolher aquele menos *recentemente* alcançado na busca

- Dessa forma, os vértices são armazenados numa fila de modo a serem processados "*first in first out*"

- Percorre-se o grafo como se houvesse uma onda na água!



Busca em largura

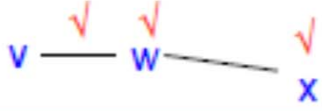
- Esse algoritmo faz uso do conceito de fila
  - O grafo é percorrido de maneira sistemática, primeiro marcando como “visitados” todos os vizinhos de um vértice e em seguida começa a visitar os vizinhos de cada vértice na ordem em que eles foram marcados.

- Para realizar essa tarefa, uma fila é utilizada para administrar a visitação dos vértices
  - o primeiro vértice marcado (ou marcado a mais tempo) é o primeiro a ser visitado.

Algoritmo busca em largura

Dado G(V,A), conexo:

escolher uma raiz s de V
definir uma fila Q, vazia
marcar s
inserir s em Q
enquanto Q não vazia faça
    seja v o 1o. vértice de Q
    para cada w ∈ ListaAdjacencia(v) faça
        se w é não marcado então
(I)   visitar (v,w)
                   marcar w
                   inserir w em Q
(II)   senão se w ∈ Q então visitar (v,w) /*w alcançado por outro caminho*/
            /*senão já processou w e portanto (w,v)*/
    /*fim_para*/
    retirar v de Q
/*fim_enquanto*/



Busca em largura

Observações

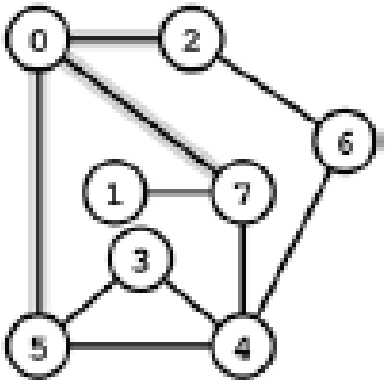
- Todos os nós com distância k a um nó v são visitados antes dos nós com distância k+1(garantido pelo uso da fila)

- Descobre todos os vértices alcançáveis a partir de v (portanto, pode ser usada para achar caminhos)

- A busca em largura resulta no **caminho mais curto** entre o vértice inicial e um vértice qualquer x

Busca em largura - exemplo

- Representar G por sua lista de adjacências e realizar a busca em largura. Eis o estado da fila (coluna esquerda) no início de cada iteração:

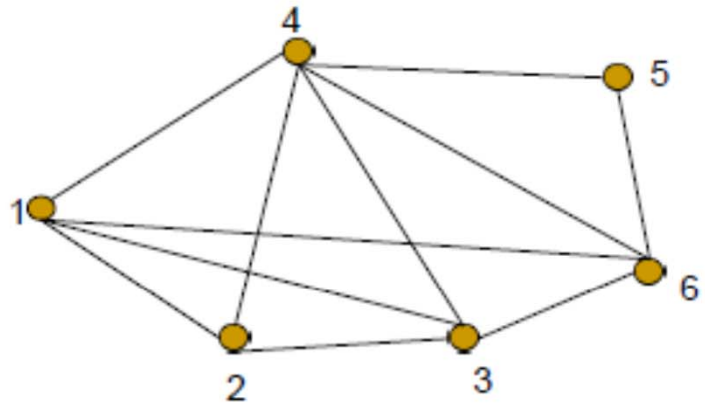



```
queue

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

Portanto, os vértices são visitados na ordem 0 2 5 7 6 3 4 1

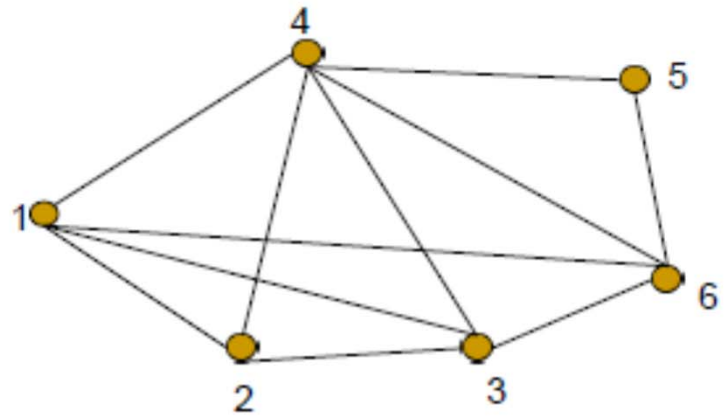
Busca em largura - exemplo



Listas de Adjacências:
1: (4,2,3,6)
2: (1,4,3)
3: (2,1,4,6)
4: (1,2,3,6,5)
5: (4,6)
6: (3,1,4,5)

Vértice inicial: 1 (raiz da busca)
Q:
Vértices Marcados:
Arestas Visitadas:

Busca em largura - exemplo

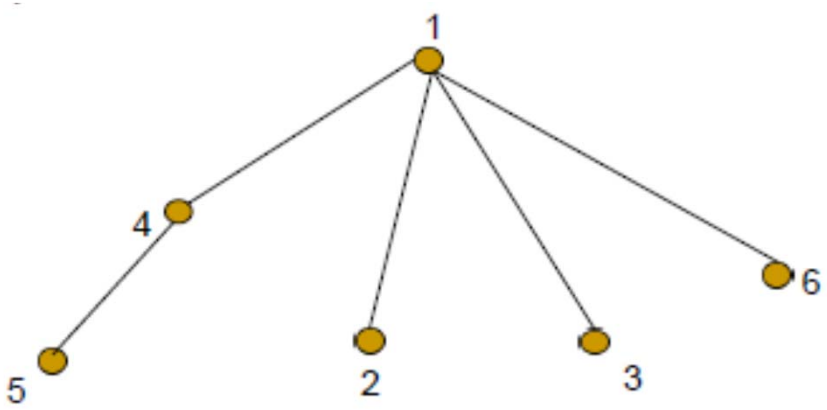


Listas de Adjacências:
1: (4,2,3,6)
2: (1,4,3)
3: (2,1,4,6)
4: (1,2,3,6,5)
5: (4,6)
6: (3,1,4,5)

Vértice inicial: 1 (raiz da busca)
Q: (1,4,2,3,6,5)
Vértices Marcados: 1,2,3,6,5
Arestas Visitadas: (1,4) (1,2) (1,3) (1,6) (4,2) (4,3) (4,6) (4,5) (2,3) (3,6) (6,5)

Árvore geradora do grafo

- Seja E_T o conjunto das arestas visitadas em (I). O Grafo T(V,ET) é uma **árvore geradora** de G (também chamada de árvore de largura de G).



Uma árvore geradora $G' = (V', E')$ de um grafo é um subgrafo gerador ($V' = V$ e $E' \subseteq E$) que é uma árvore.

Complexidade do BFS

O(|V| + |E|), ou seja, linear em relação ao tamanho da representação do grafo por listas de adjacências

- Todos os vértices são enfileirados/desenfileirados no máximo uma vez; o custo de cada uma dessas operações é O(1), e elas são executadas O(|V|) vezes

- A lista de adjacências de cada vértice é percorrida no máximo uma vez (quando o vértice é desenfileirado); o tempo total é O(|E|) (soma dos comprimentos de todas as listas, igual ao número de arestas)

Aplicações busca em largura

- Achar todos os vértices conectados a apenas um componente;
- Achar o menor caminho entre dois vértices;
- Testar se um grafo é bipartido;
- Roteamento: encontrar um número mínimo de hops em uma rede.
  - os hops são os vértices intermediários no caminho correspondente à conexão;
- Encontrar número mínimo de intermediários entre 2 pessoas.

Busca em profundidade

Busca em profundidade

Funcionamento

- Partindo de um vértice inicial, a busca explora o máximo possível cada um dos vizinhos de um vértice antes de retroceder (*backtracking*)

- Em outras palavras, esse tipo de busca se inicia em um vértice e se aprofunda nos vértices vizinhos deste até encontrar um dos dois casos:
  - o alvo da busca
  - um vértice sem vizinhos que possam ser visitados

Busca em profundidade

Backtracking

- O grafo é percorrido de maneira sistemática até que a busca falhe, ou se encontre um vértice sem vizinhos
  - Nesse momento entra em funcionamento o mecanismo de *backtracking*: a busca retorna pelo mesmo caminho percorrido com o objetivo de encontrar um caminho alternativo.

- Trata-se de um mecanismo usado em linguagens de programação como Prolog.

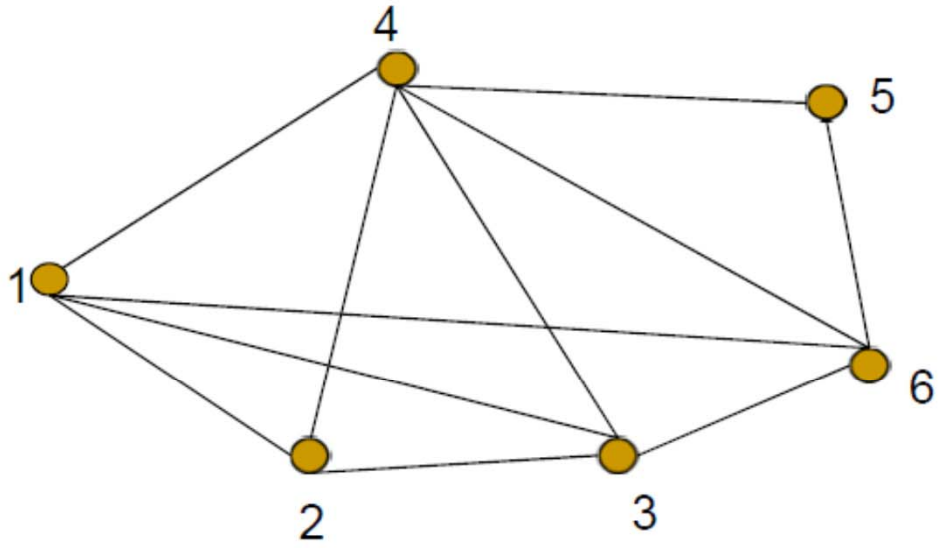
Busca em profundidade
DFS – *Depth-First Search*

- Explora-se profundamente cada vértice do grafo:
  - Todos os vértices adjacentes ao vértice recém-visitado são visitados imediatamente após o mesmo.
  - Ao contrário da busca em largura, onde os vértices adjacentes “irmãos” são visitados antes de dos vértices de suas próprias listas de adjacência)

- A busca em profundidade é obtida do método básico, onde a seleção do próximo vértice marcado obedece a:
  - *Dentre todos os vértices marcados e incidentes a alguma aresta ainda não explorada, escolher aquele mais recentemente alcançado na busca*

- Dessa forma, os vértices são armazenados numa **pilha** de modo a serem processados *“last in first out”*

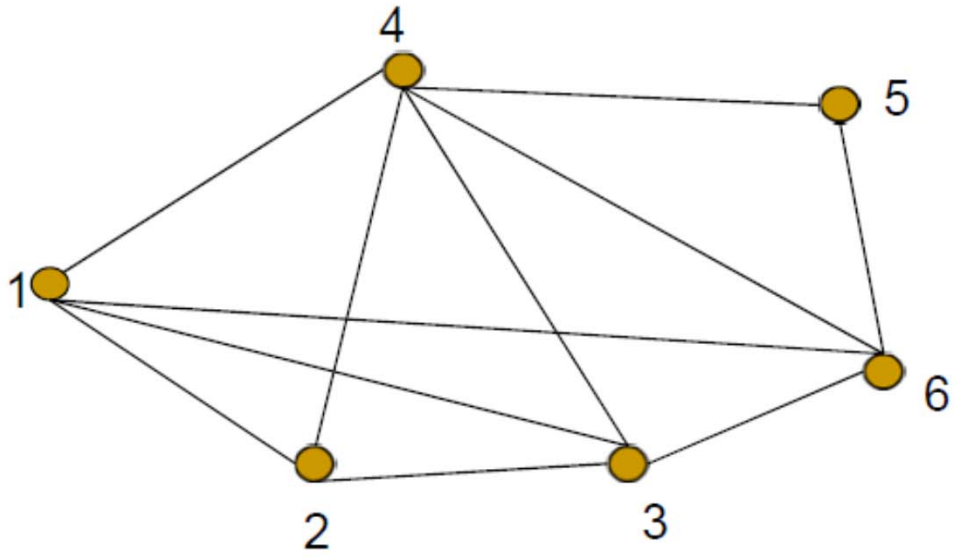
Busca em profundidade - exemplo



Listas de Adjacências:
1: (4,2,3,6)
2: (1,4,3)
3: (2,1,4,6)
4: (1,2,3,6,5)
5: (4,6)
6: (3,1,4,5)

Vértice inicial: 1 (raiz da busca)
P:
Vértices Marcados:
Arestas Visitadas:

Busca em profundidade - exemplo



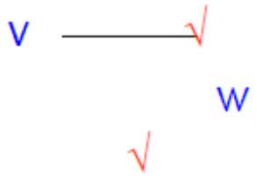
Listas de Adjacências:
1: (4,2,3,6)
2: (1,4,3)
3: (2,1,4,6)
4: (1,2,3,6,5)
5: (4,6)
6: (3,1,4,5)

Vértice inicial: 1 (raiz da busca)
Pilha: (1,4,2,3,6,5)
Vértices Marcados: 1,4,2,3,6,5
Arestas Visitadas: (1,4) (4,2) (2,1) (2,3) (3,1) (3,4) (3,6) (6,1) (6,4) (6,5) (5,4)

Algoritmo busca em profundidade

Dado G(V,A), conexo:

```
Prof(v)
    marcar v
    empilhar v
    para cada w ∈ ListaAdjacencia(v) faça
       se w é não marcado então
    (I)   visitar (v,w)
          Prof (w)          /*recursão*/
       senão  se w está na Pilha e v e w não são consecutivos na Pilha
     (II)     então visitar (v,w)
              /*senão aresta já visitada*/
    /*fim_para*/
     desempilhar v
fim /*Prof(v)*/
```



Algoritmo busca em profundidade Observações

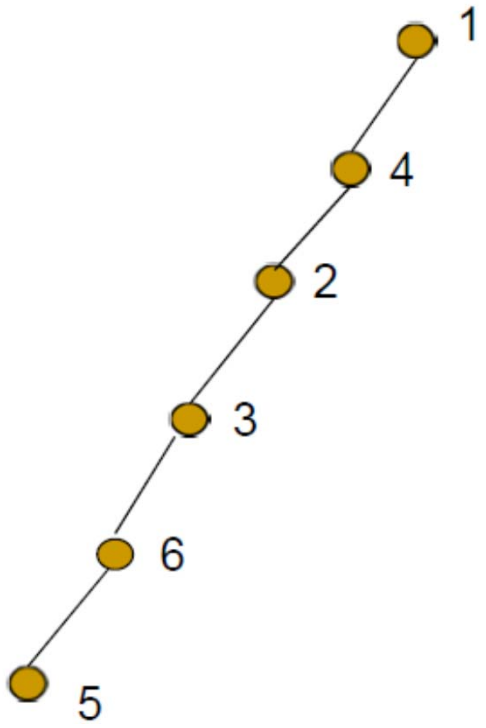
- A pilha de recursão é suficiente para indicar a ordem de processamento dos vértices

- Foi necessária uma pilha explícita apenas para verificar se 2 vértices são consecutivos na pilha (ou seja, olhar para o topo-1). Não temos acesso a essa informação na pilha de recursão.

- Esse teste é necessário apenas porque queremos visitar todas as arestas (e apenas uma vez). Se apenas os vértices nos interessam, esse teste não é necessário.

Árvore geradora do grafo

- Seja A_T o conjunto das arestas visitadas em (I). O Grafo T(V,AT) é uma árvore geradora de G (também chamada de árvore de profundidade de G).



Complexidade da busca em profundidade

O(|V| + |E|)

- Prof(v) é chamado exatamente uma vez para cada vértice de V
- Em Prof(v), o laço é executado |Ladj(v)| vezes, i.e., O(|E|) no total

Busca em profundidade – aplicações

- Encontrar componentes conectados e fortemente conectados;
- Ordenação topológica de um grafo;
- Procurar a saída de um labirinto;
- Verificar se um grafo é completamente conexo
  - por exemplo, a rede de computadores esta funcionando direito ou não;
- Implementar a ferramenta de preenchimento
  - balde de pintura do Photoshop

Exercícios

Exercício 1

- Seja G o dígrafo com 6 vértices definido pelos arcos listados a seguir.

  0-2 0-3 0-4 1-2 1-4 2-4 3-4 3-5 4-5 5-1.

Represente G por sua lista de adjacência e aplique o algoritmo de busca em largura. Qual é a ordem de visitação dos vértices? Apresente o estado da fila no início de cada iteração.

Resposta: 0 2 3 4 5 1

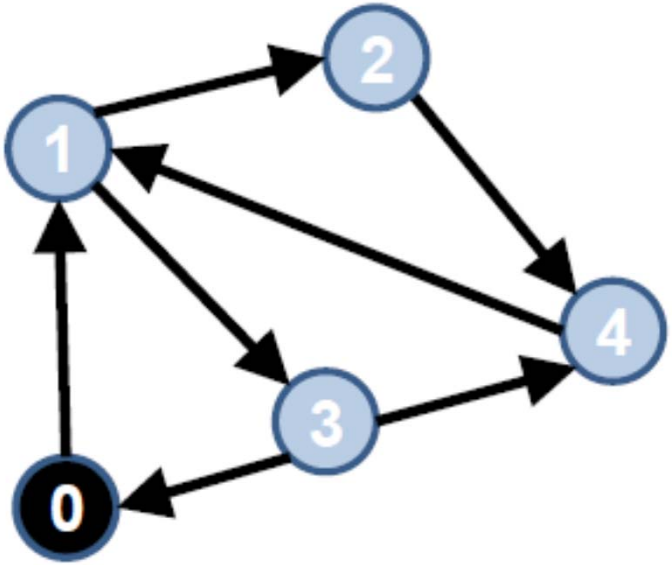
Exercício 2

- Seja G um dígrafo simétrico. A lista de arestas de G é dada por (cada aresta é um par de arcos):

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

Qual é a ordem de visitação dos vértices para a busca em largura e para a busca em profundidade?

Exercício 3



- Busca em largura
- Busca em profundidade

Exercício 4

- Apresentar a árvore geradora dos grafos dos exercícios anteriores.

Exercício 5

- Implementação da busca em largura com listas de adjacência
- Implementação da busca em profundidade com listas de adjacência

Referências

- Notas de aula Prof. Paulo Feofiloff:
  http://www.ime.usp.br/~pf/algoritmos_para_grafos/aulas/bfs.html

- Notas de aula Prof. André Backes:
  http://www.facom.ufu.br/~backes/gsi011/Aula09-GrafosBuscas.pdf

- Notas de aula Profa. Graça Nunes:
  http://wiki.icmc.usp.br/images/3/3a/4._GrafosBuscaLargura.pdf

  http://wiki.icmc.usp.br/images/6/67/5._GrafosBuscaProf.pdf