

Buscas e Caminhos Mínimos em Grafos

Roberto Sales

- Permitem responder perguntas sobre conectividade
 - ① O vértice u está conectado ao vértice v / existe caminho entre u e v .
 - ② É possível transitar entre qualquer par de vértices (u, v) num dado grafo direcionado G ? (problema da conectividade forte)
 - ③ Existe a garantia de que, se uma aresta for removida, o grafo continuará conexo? (problema das pontes)

- Permitem responder perguntas sobre propriedades do grafo
 - ① O grafo não-direcionado G contém ciclos?
 - ② O grafo G é um grafo bipartido?
- Permitem resolver problemas de otimização
 - ① Caminhos mais curtos num grafo
 - ② Caminhos mais longos em alguns grafos com características específicas
 - ③ Caminho entre dois vértices u, v tal que a aresta de menor capacidade (gargalo) é a maior possível

Busca em Profundidade

Busca em Profundidade (DFS)

Estratégia

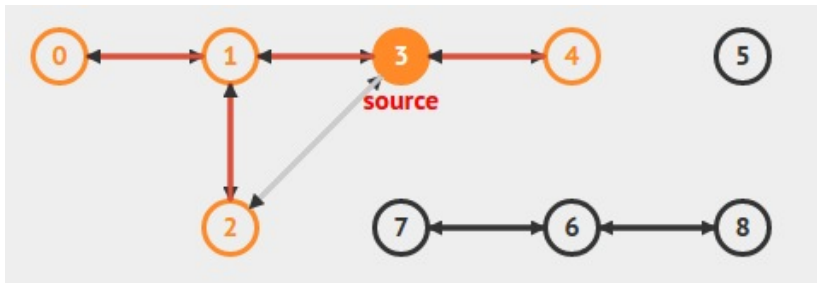
Visitar os vértices do grafo de forma recursiva, seguindo sempre pelo próximo vértice não explorado, até que não seja mais possível continuar. Em geral, visita todo vértice e processa toda aresta no máximo uma vez e, portanto, pára.

Complexidade: $O(|V| + |E|)$

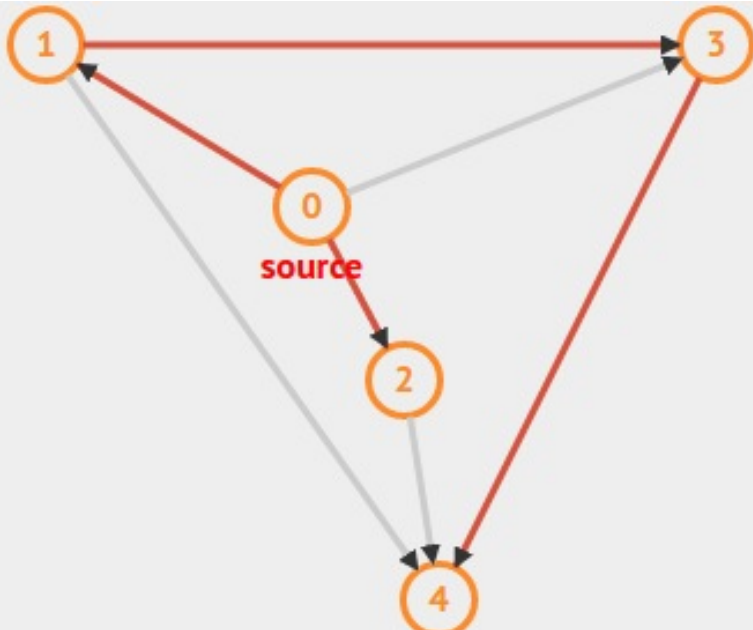
DFS clássica

```
bool vis[];  
vector<int> adj[];  
  
void dfs(int u){  
    vis[u] = true;  
    for(int v : adj[u])  
        if(!vis[v]) dfs(v);  
}  
  
// no caso de várias componentes  
for(int i = 0; i < n; i++)  
    if(!vis[i]) dfs(i);
```

Busca em Profundidade (DFS)



Busca em Profundidade (DFS)



Aplicações da DFS

Não resolve

- Problemas de caminhos mínimos em grafos gerais
- Problemas de caminhos mais longos em grafos gerais

Resolve

- Bicolorir um grafo
- Checar conectividade, contar componentes conexas
- Otimizações em árvores

Variações

- Circuito euleriano
- Caminhos aumentantes (fluxos)
- Componentes biconexas, pontes e pontos de articulação
- Componentes fortemente conexas

Bicoloração

```
int color[]; // inicializado em -1
vector<int> adj[];
bool fail;

void dfs(int u, int c /* 0 ou 1 */){
    color[u] = c;
    for(int v : adj[u])
        if(color[v] == -1) dfs(v, c^1);
        else if(color[v] == c) fail = true;
}
```

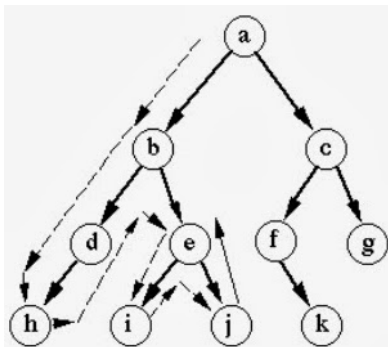
Busca em Largura

Busca em Largura (BFS)

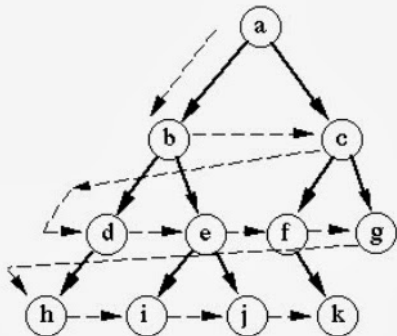
Estratégia

Ao invés de visitar os vértices de forma recursiva, iremos visitá-los em camadas. Em termos de implementação, a única mudança é que deixaremos de usar uma pilha implícita (recursão) em favor de uma fila.

BFS vs. DFS



Depth-first search



Breadth-first search

Busca em Largura (BFS)

BFS

```
bool vis[];  
queue<int> q;  
  
q.push(source);  
vis[source] = true;  
  
while(!q.empty()){  
    int u = q.front(); q.pop();  
    for(int v : adj[u]) if(!vis[v]){  
        vis[v] = true;  
        q.push(v);  
    }  
}
```

Busca em Largura (BFS)

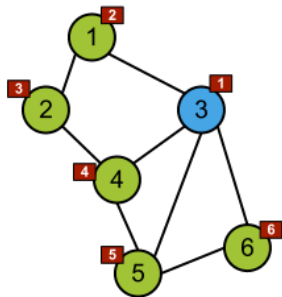
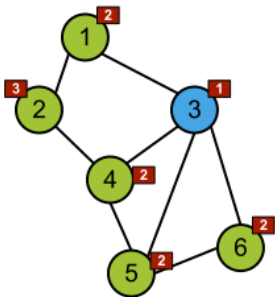
BFS (caminhos mínimos)

```
int dist[]; // inicializado para inf
queue<int> q;

q.push(source);
dist[source] = 0;

while(!q.empty()){
    int u = q.front(); q.pop();
    for(int v : adj[u]) if(dist[u]+1 < dist[v]){
        dist[v] = dist[u]+1;
        q.push(v);
    }
}
```

Breadth-First vs. Depth-First Search



Curiosidades

Um grafo não precisa estar explicitamente representado na memória para que um algoritmo de busca ou de caminhos mínimos seja executado nele.

Exemplos

- Problema de caminho mais curto num labirinto (BFS)
- Ladrilhos (Regional - Maratona de Programação 2016) (DFS/BFS)
- Mania de Par (Regional - Maratona de Programação 2015) (dijkstra)
- Contêineres (Regional - Maratona de Programação 2016) (dijkstra)

Qualquer coisa que expresse relações pode ser um grafo, fique atento!

Dijkstra

Motivação

Computar caminhos mínimos a partir de um vértice u em um grafo com arestas de custo **não-negativo**.

Subestrutura ótima dos caminhos mínimos

Para todo caminho $u-v-w$, onde (v, w) é uma aresta do grafo em questão, se $u-v-w$ é um caminho mínimo, então $u-v$ é um caminho mínimo.

```

1: procedure DIJKSTRA( $V, E, s$ )
2:   for all  $u \in V \setminus \{s\}$  do
3:      $d_u = \infty$ 
4:   end for
5:    $d_s = 0$ 
6:    $S = \emptyset$ 
7:   while  $S \neq V$  do
8:     extract  $u \notin S$  such that  $d_u$  is minimum
9:      $S = S \cup \{u\}$ 
10:    for  $v$  adjacent to  $u$  do                                ▷ relaxing step
11:      if  $d_u + w(u, v) < d_v$  then
12:         $d_v = d_u + w(u, v)$ 
13:      end if
14:    end for
15:  end while
16: end procedure

```

XOR Equations

Dadas n variáveis booleanas x_1, x_2, \dots, x_n e m equações na forma $x_i \oplus x_j = z$, onde $z \in \{0, 1\}$, determine se o sistema tem uma solução. Em caso positivo, descubra uma.

Sequência Lexicograficamente Máxima

Dada uma sequência de inteiros s_1, s_2, \dots, s_n e um conjunto de operações válidas X , determine a maior sequência lexicograficamente que é possível obter usando um número finito de operações.

O conjunto X tem m pares na forma (i, j) . Tal par descreve uma operação de troca entre as posições i e j da sequência.