Estruturas de Dados

Busca em Grafos

Universidade Estadual Vale do Acaraú – UVA

Paulo Regis Menezes Sousa

paulo_regis@uvanet.br

Busca em profundidade

Busca em largura

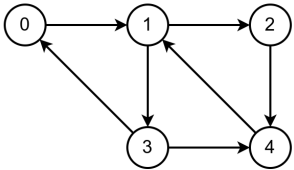
- Um algoritmo de busca (ou de varredura) é qualquer algoritmo que visita todos os vértices de um grafo andando pelas arestas de um vértice a outro.
- Estudaremos algoritmo de busca em profundidade (= *depth-first search*), ou busca **DFS**.
- O algoritmo de busca DFS visita todos os vértices do grafo e guarda a ordem que cada vértice é encontrado.
- A ordem dos vértices ajuda a compreender a forma do grafo.

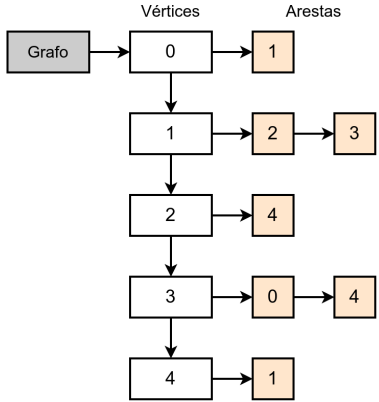
```
1  Vertex **Graph_dfs(Graph *g, void *startValue, int (*cmp)(void*,void*)) {
2      Vertex **path = NULL, *v = NULL;
3      int i, count = 0;
4
5      if (g && startValue) {
6          v = Graph_findVertexByValue(g, startValue, cmp);
7
8          if (v) {
9              path = calloc(g->n, sizeof(Vertex*));
10             dfs(v, path, &count);
11         }
12     }
13
14     return path;
15 }
```

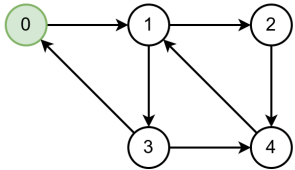
Código 1: Função recursiva de busca em profundidade

```
1  static void dfs(Vertex *v, Vertex **path, int *count) {
2      Vertex *k = NULL;
3
4      if (v) {
5          v->visited = 1;
6          path[*count] = v;
7
8          while (k = notVisited(v->first)) {
9              *count += 1;
10             dfs(k, path, count);
11         }
12     }
13 }
```

```
1  static Vertex *notVisited(Edge *edge) {
2      Edge *e = NULL;
3
4      if (edge) {
5          e = edge;
6
7          while (e) {
8              if (e->head->visited == 0)
9                  return e->head;
10             e = e->next;
11         }
12     }
13
14     return NULL;
15 }
```

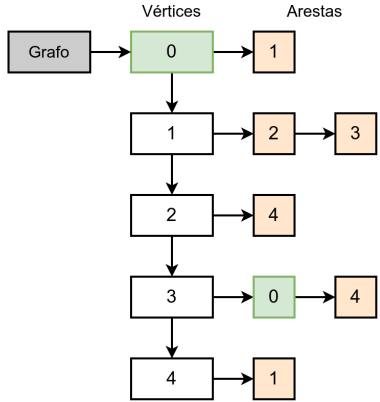

0
1
2
3
4

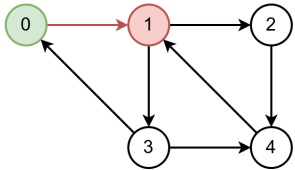

Vértices
Arestas
Grafo
0
1
1
2
3
2
4
3
0
4
4
1



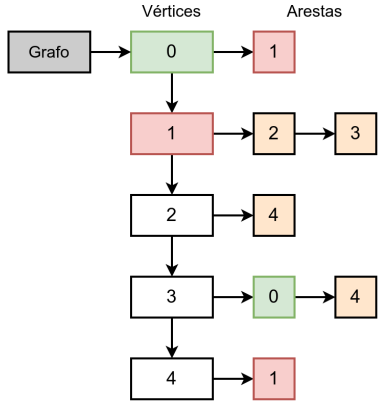
Vértices encontrados

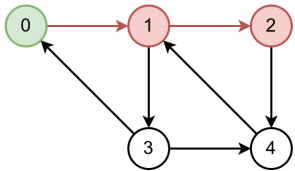
0



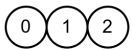


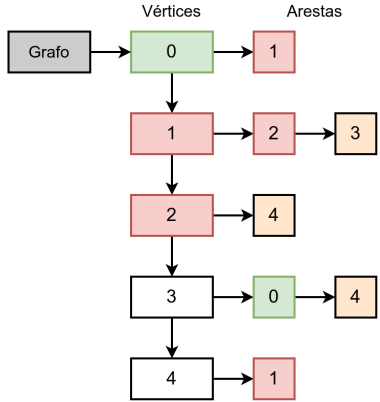
Vértices encontrados

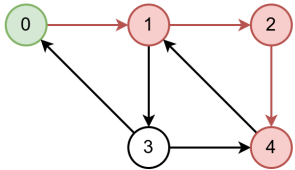




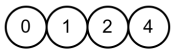
Vértices encontrados

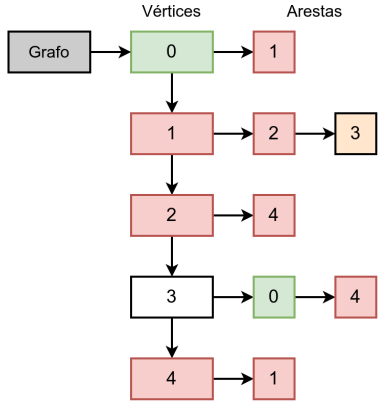


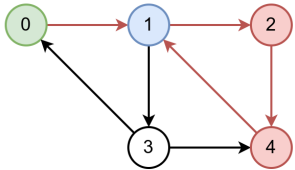




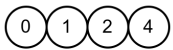
Vértices encontrados

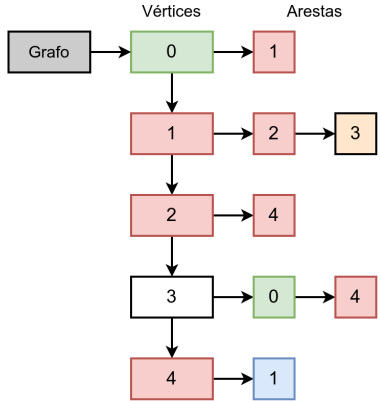


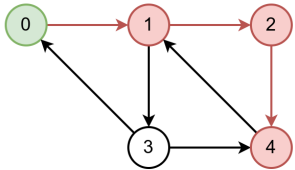




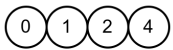
Vértices encontrados

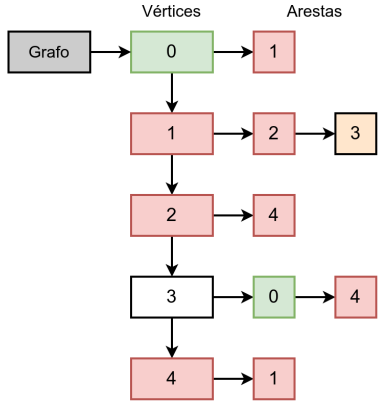


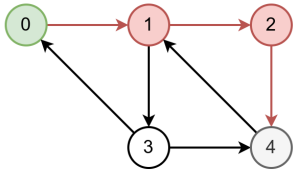




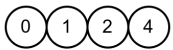
Vértices encontrados

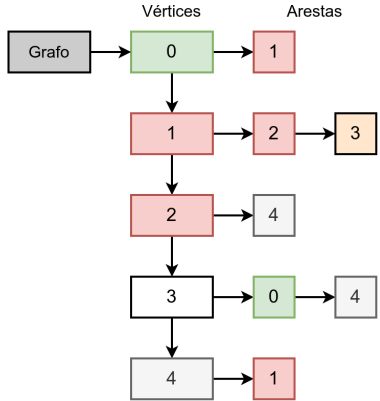


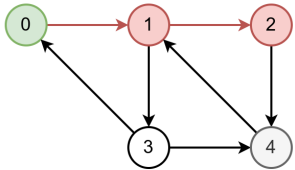




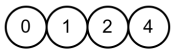
Vértices encontrados

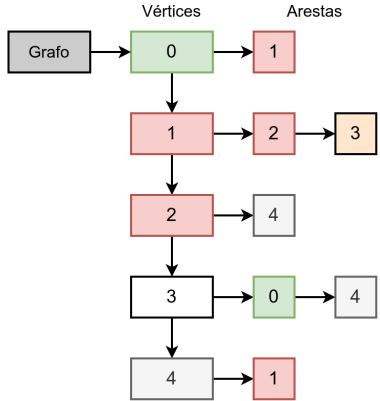


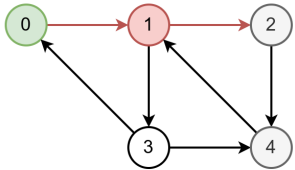




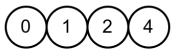
Vértices encontrados

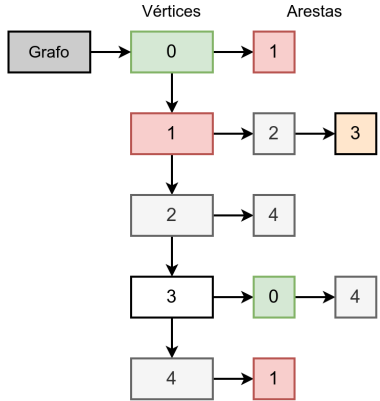


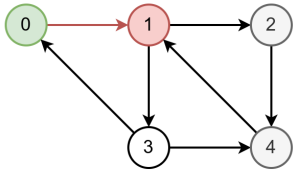




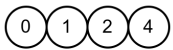
Vértices encontrados

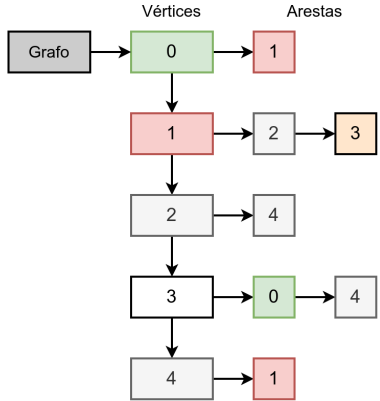


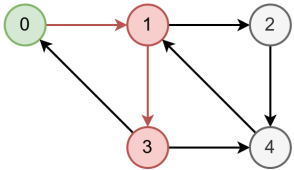




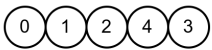
Vértices encontrados

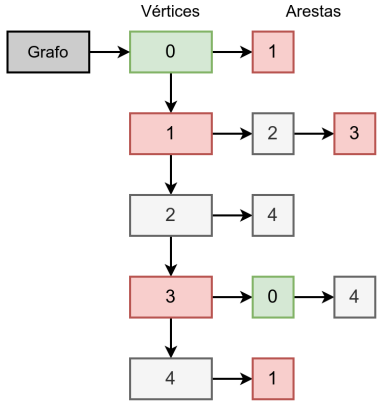


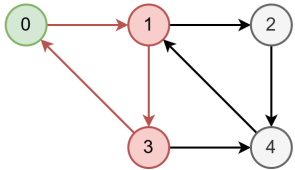


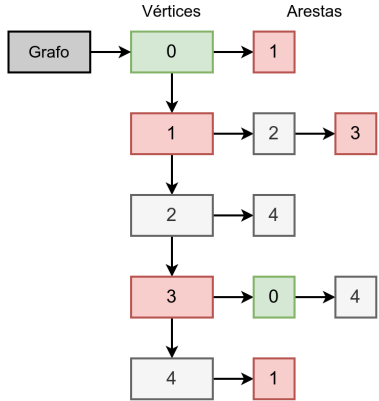


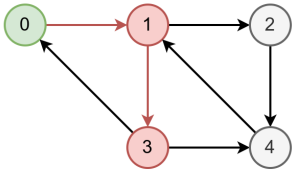
Vértices encontrados



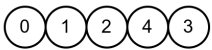


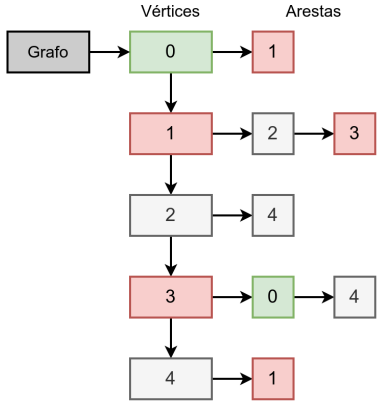

0
1
2
3
4
Vértices encontrados
0
1
2
4
3

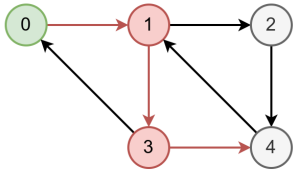

Vértices
Arestas
Grafo
0
1
1
2
3
2
4
3
0
4
4
1



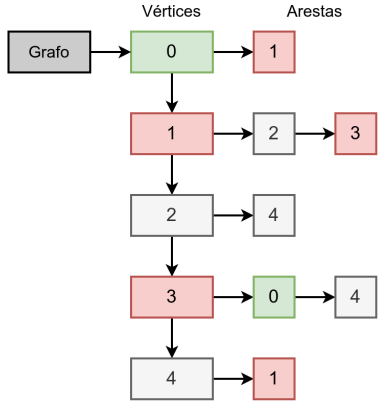
Vértices encontrados

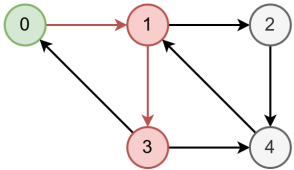




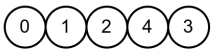


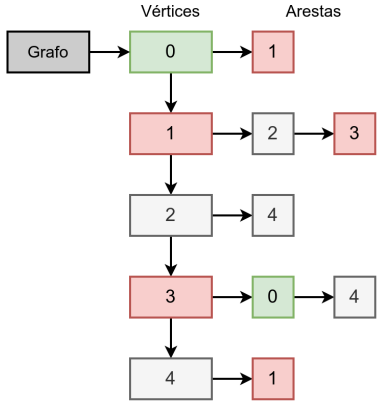
Vértices encontrados

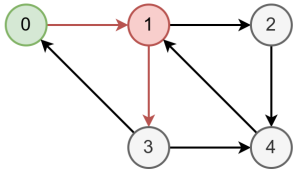




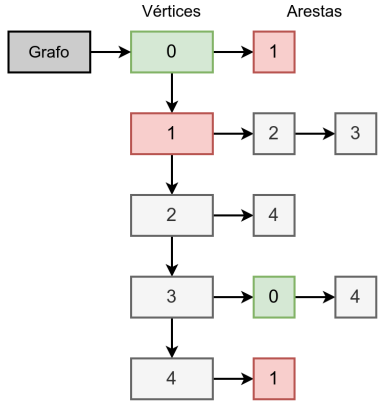
Vértices encontrados

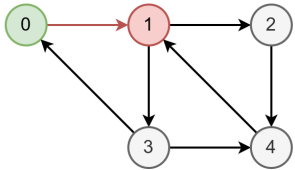




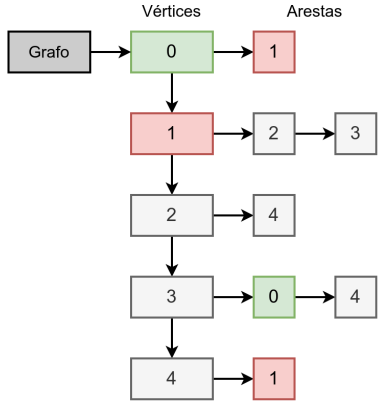


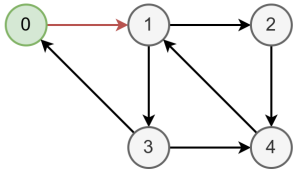
Vértices encontrados



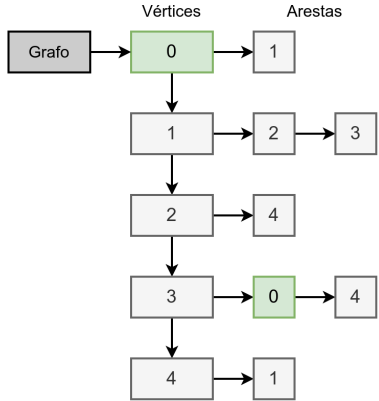


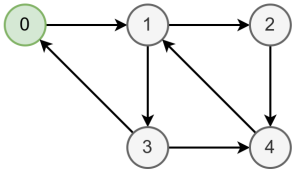
Vértices encontrados



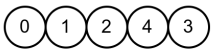


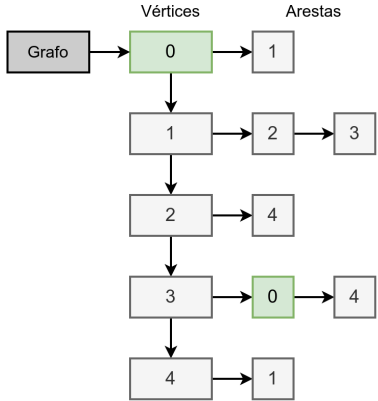
Vértices encontrados

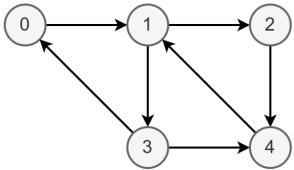




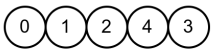
Vértices encontrados

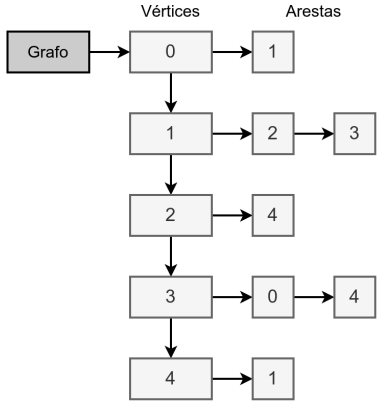






Vértices encontrados





- A função `dfs()` examina o leque de saída de cada vértice uma só vez.
- Portanto, cada aresta é examinada uma só vez.
- Assim, se o grafo tem V vértices e A arcos, `dfs()` consome tempo proporcional a $|V| + |A|$.
- A ordem em que a função `dfs()` descobre os vértices do grafo é chamada pré-ordem (= *preorder*).

- A busca em largura (= *breadth-first search*). começa por um vértice s, especificado pelo usuário.
- O algoritmo visita s, depois visita todos os vizinhos de s, em seguida todos os vizinhos dos vizinhos, e assim por diante.
- O algoritmo retorna a ordem em que os vértices são descobertos.
- No começo da primeira iteração, a fila contém o vértice s.

Código 2: Algoritmo de busca em largura

```
1  escolha um vértice inicial s no grafo G
2       marque s como visitado
3       insira s na fila F
4       enquanto F não está vazia faça
5            seja v o primeiro vértice de F
6            para cada w adjacente a v faça
7                 se w não foi visitado então
8                      marque w como visitado
9                      insira w em F
10           retire v de F
```

»Animação

```
1  Vertex **Graph_bfs(Graph *g, void *startValue, int (*cmp)(void*,void*)) {
2      Vertex **path = NULL;
3      Vertex *v = NULL, *k = NULL;
4      Queue  *q = NULL;
5      int count = 0;
6
7      if (g && startValue) {
8          if (v = Graph_findVertexByValue(g, startValue, cmp)) {
9
10             path = calloc(g->n, sizeof(Vertex*));
11             q = Queue_create(g->n);
12
13             v->visited = 1;
14             path[count++] = v;
```

```
16                Queue_push(q, v);
17
18                while (!Queue_isEmpty(q)) {
19                    v = Queue_begin(q);
20
21                    while (k = notVisited(v->first)) {
22                        k->visited = 1;
23                        path[count++] = k;
24                        Queue_push(q, k);
25                    }
26
27                    Queue_pop(q);
28                }
29            }
30        }
31
32        return path;
33    }
```