Grafos

Buscas e Implementação em linguagem C

Maria Adriana Vidigal de Lima

FACOM - UFU

Sumário

Busca em Grafo

- Um algoritmo de busca percorre um grafo andando pelos arcos de um vértice a outro. Cada arco é percorrido no máximo uma vez.
- Para a navegação seja realizada de forma organizada e sistemática, foram propostos **algoritmos de busca**.
- As buscas em grafo são usadas em muitas aplicações e permitem conhecer a estrutura do grafo, encontrar caminhos, classificar arestas etc.
- São dois os algoritmos básicos de busca em grafos: **busca em largura** e **busca em profundidade**.

Busca em Grafo

Visão geral dos dois percursos básicos:

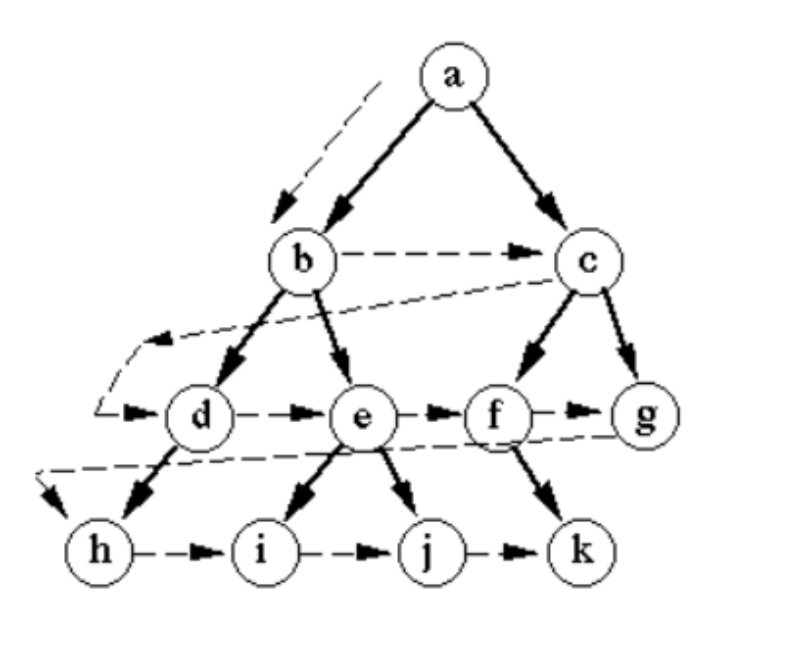


Figura 1: Percurso em Largura

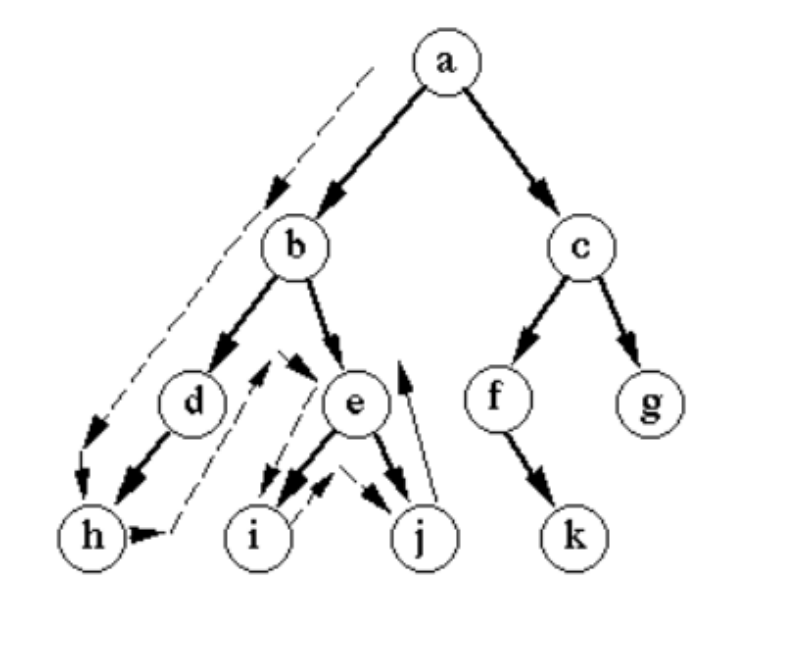


Figura 2: Percurso em Profundidade

Busca em Largura

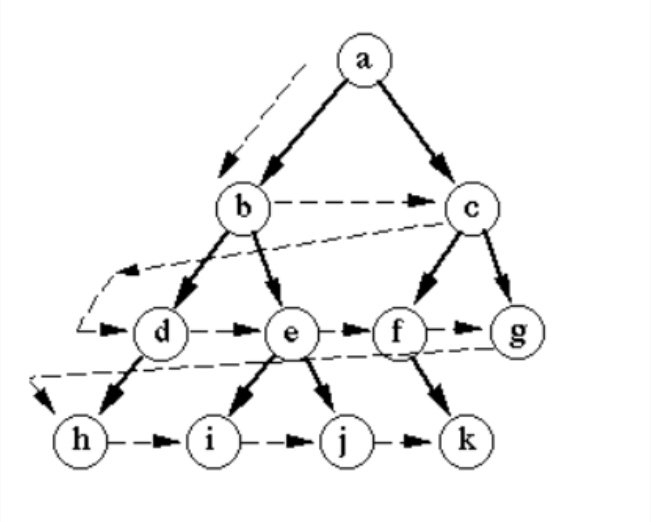
Busca em Largura

Uma busca em largura em um grafo $G = (V, A)$ é um percurso com início em um vértice w, denominado **raiz da busca**. A partir de w, são visitados todos os vértices alcançáveis, em ordem crescente de distância.

Vértice Alcançável

Um vértice w em um grafo $G = (V, A)$ é **alcançável** a partir de um vértice v se existe uma aresta $(v, w) \in A$.

O algoritmo examina sistematicamente todos os vértices através de níveis, em busca de uma solução.



Busca em Largura: Algoritmo

```
1  Entradas: Grafo (G), Vértice Raiz (s)
2  inicio
3     variável local Fila: TipoFila
4     variável local Visita: Vetor [nro de vértices do grafo]
5     criar Fila
6     preencher todo o vetor Visita com (-1)
7     com o vértice s de G faça
8        insira s na primeira posicao do vetor Visita
9        insira s em Fila
10    enquanto Fila não está vazia faça
11        seja v1 o primeiro vértice de Fila
12        para cada w vizinho de v1 faça
13              se w não está no vetor Visita então
14                    insira w em Visita
15                    insira w em Fila
16             fim se
17       fim para
18       retira v1 de Fila
19    fim enquanto
20    libera Fila
21    imprime Visita
22 fim
```

Busca em Largura: Exemplo de execução

Percurso em largura a partir do vértice 0:

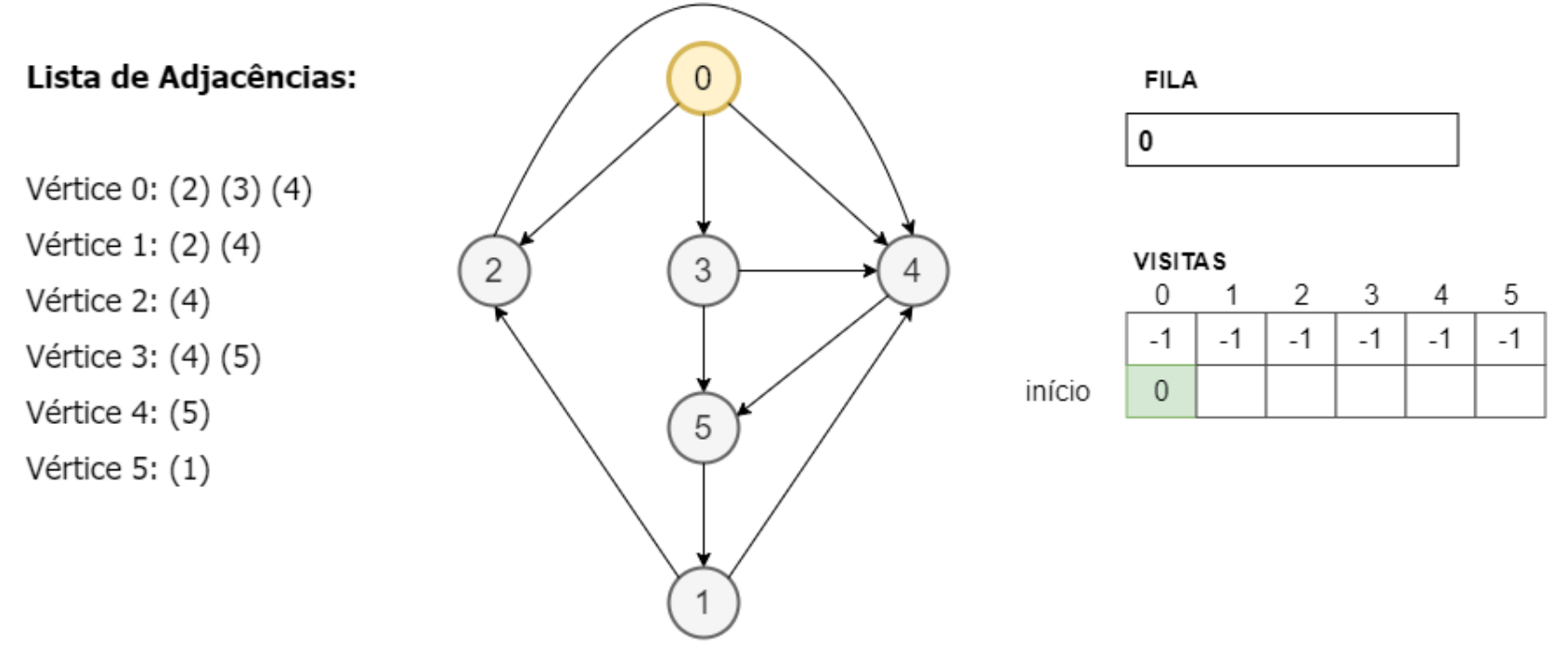


Figura 3: Início de caminhada em largura

Busca em Largura: Exemplo de execução

O primeiro passo consiste em encontrar os vértices alcançáveis a partir do vértice 0, inserir na fila e anotar como visitados:

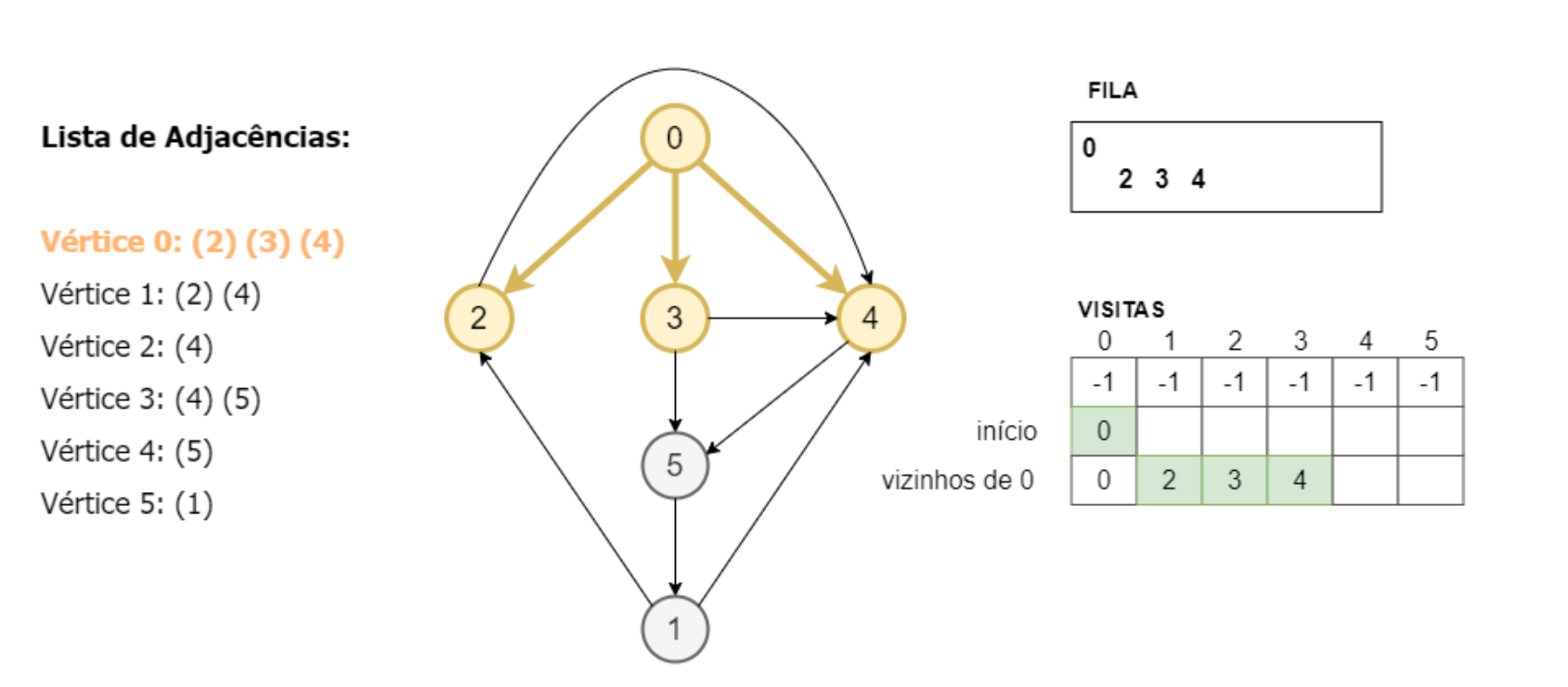


Figura 4: Descoberta dos vizinhos de 0

Busca em Largura: Exemplo de execução

O vértice 2 é retirado da fila, e são buscados os alcançáveis a partir de 2.
O arco $(2, 4)$ é encontrado, porém o vértice 4 já foi visitado.

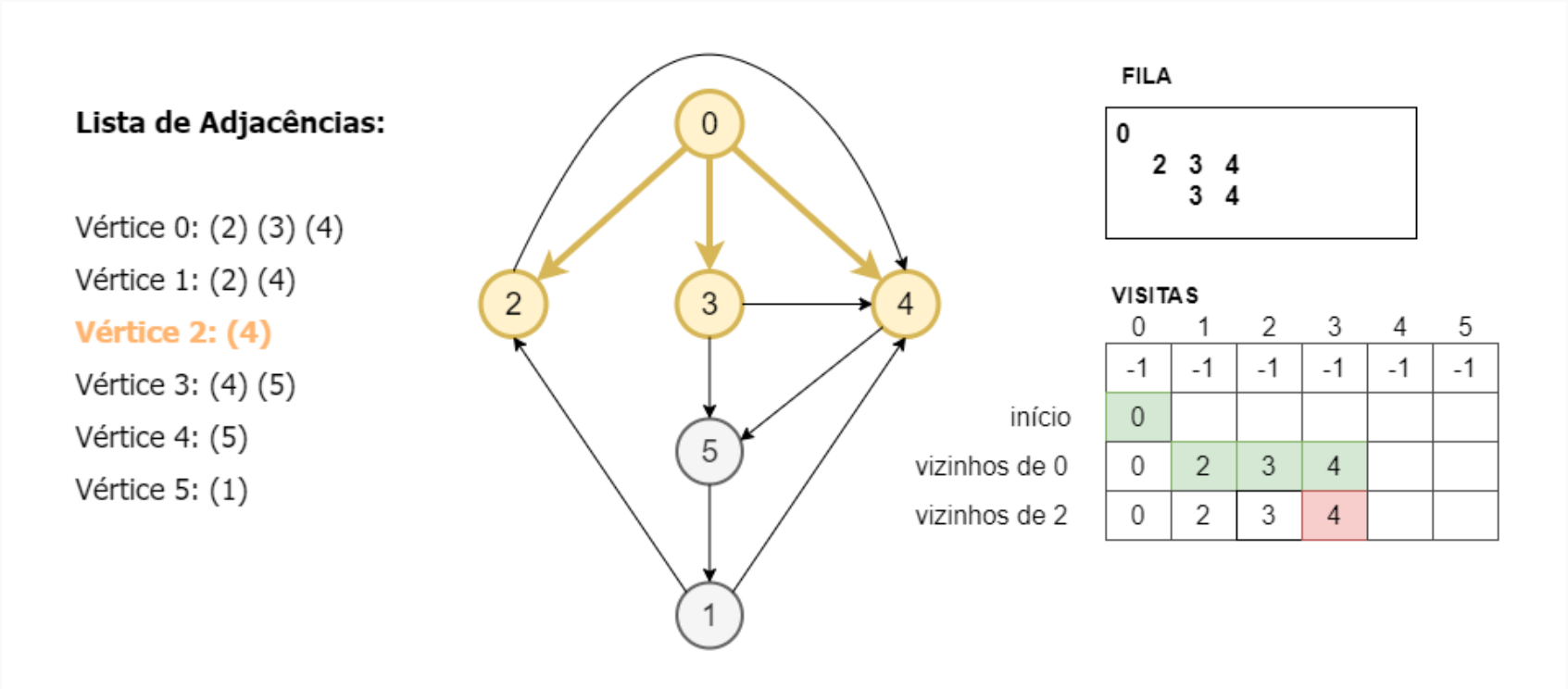


Figura 5: Descoberta dos vizinhos de 2

Busca em Largura: Exemplo de execução

O vértice 3 é retirado da fila, e os vértices alcançáveis são 4 e 5. Apenas o vértice 5 ainda não foi visitado: 5 é colocado na fila e tem sua visita anotada.

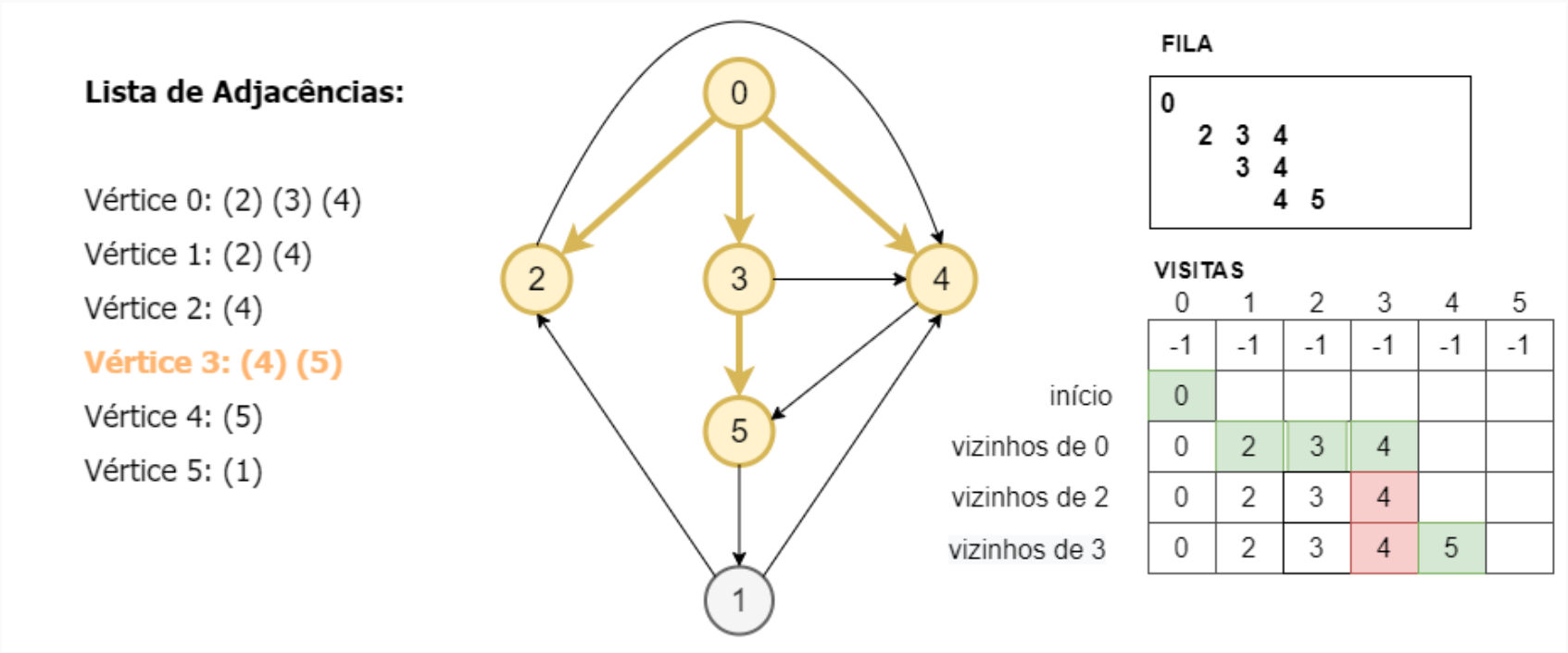


Figura 6: Descoberta dos vizinhos de 3

Busca em Largura: Exemplo de execução

O vértice 4 é retirado da fila, e seu único vértice alcançável é o 5, que já foi visitado.

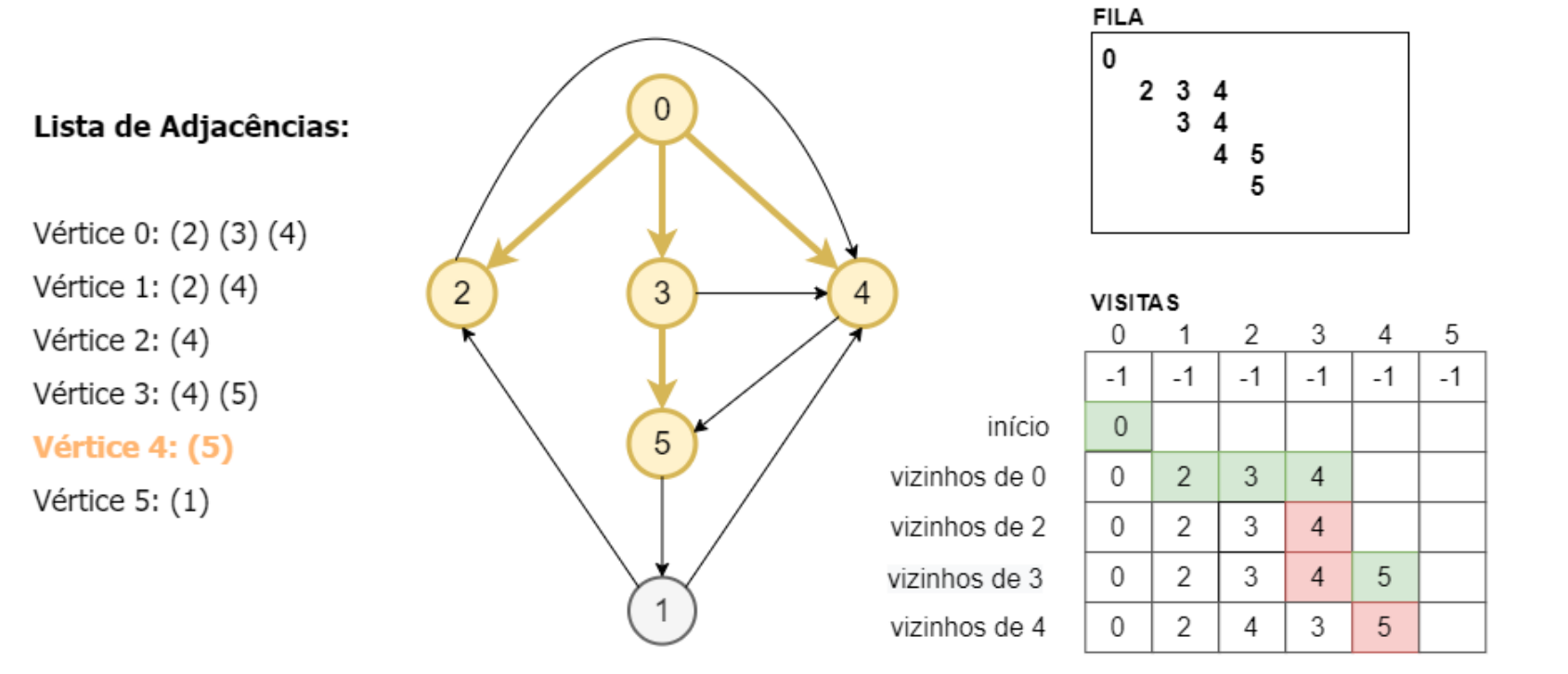


Figura 7: Descoberta dos vizinhos de 4

Busca em Largura: Exemplo de execução

O vértice 5 é retirado da fila, e este alcança o vértice 1, ainda não visitado. O vértice um é enfileirado e tem sua visita marcada.

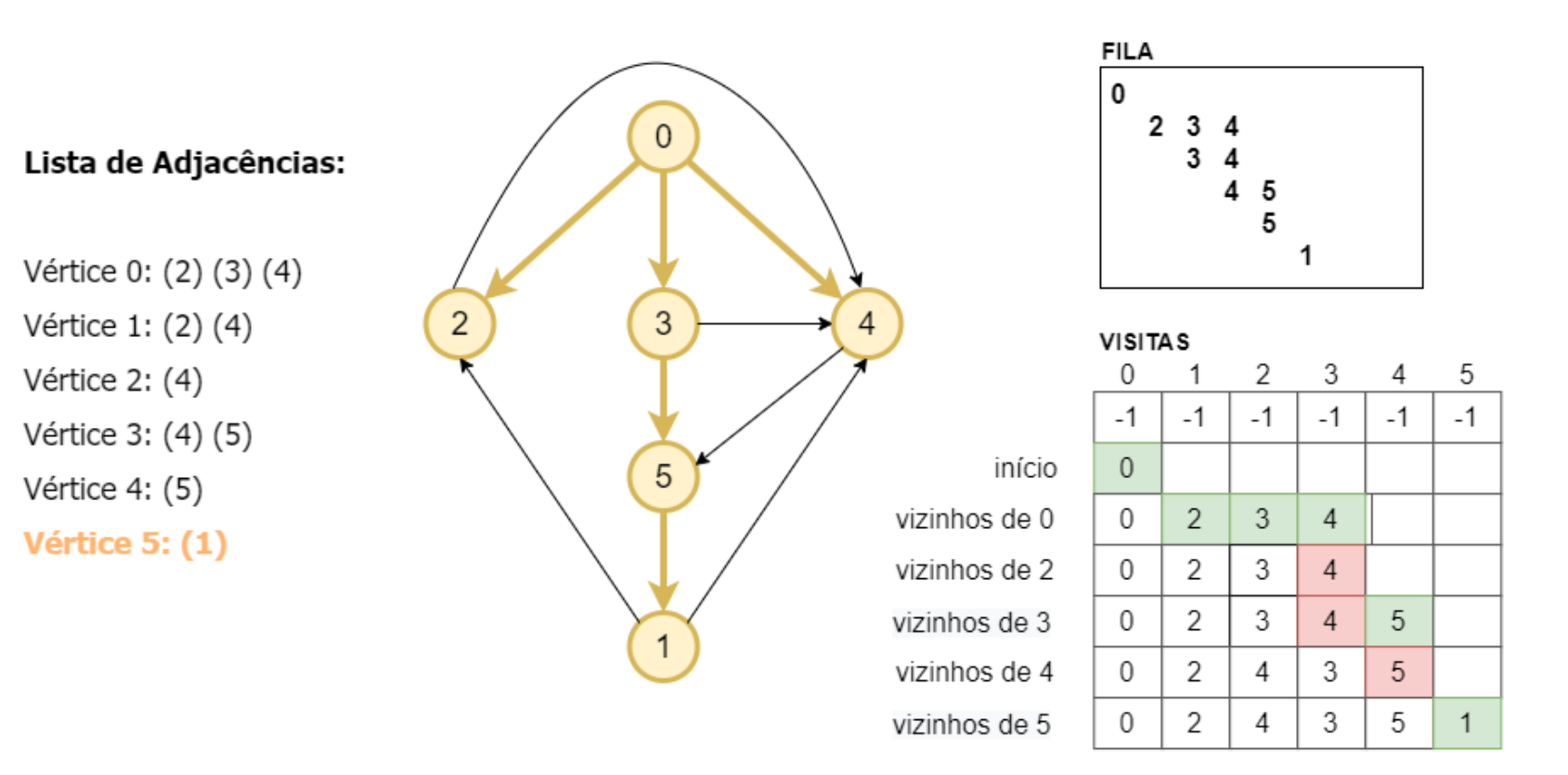


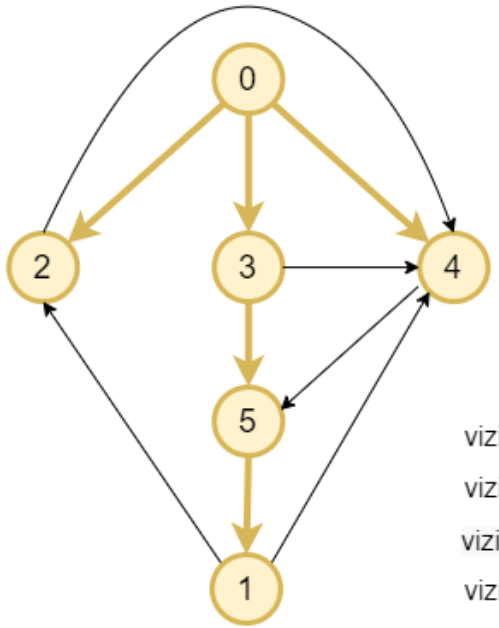
Figura 8: Descoberta dos vizinhos de 5

Busca em Largura: Exemplo de execução

Os alcançáveis a partir do vértice 1 são os vértices 2 e 4, já previamente visitados. Assim, o percurso é encerrado com o resultado da caminhada em largura: 0 2 3 4 5 1

Lista de Adjacências:

Vértice 0: (2) (3) (4)
Vértice 1: (2) (4)
Vértice 2: (4)
Vértice 3: (4) (5)
Vértice 4: (5)
Vértice 5: (1)



FILA

0					
	2	3	4		
		3	4		
			4	5	
				5	
					1

VISITAS

	0	1	2	3	4	5
	-1	-1	-1	-1	-1	-1
início	0					
vizinhos de 0	0	2	3	4		
vizinhos de 2	0	2	3	4		
vizinhos de 3	0	2	3	4	5	
vizinhos de 4	0	2	3	4	5	
vizinhos de 5	0	2	3	4	5	1
vizinhos de 1	0	2	3	4	5	1
	0	**2**	**3**	**4**	**5**	**1**

Figura 9: Descoberta dos vizinhos de 1 e ordem de descoberta

Busca em Largura: Implementação em C

```
1  void BuscaEmLargura(Grafo G, int v){
2     struct noVert *nv;
3     struct noAdj *na;
4     int *visitados;
5     TipoFila *fila;
6     int i=0, cont = 0, vt;
7
8     // Criar e iniciar o vetor de visitados com -1 para todos os vértices
9     visitados = (int *) malloc (G->NumVert * sizeof (int));
10    for (i=0; i<G->NumVert; i++){
11       visitados[i] = -1;
12    }
13    // Anotar como visitado o vértice de origem (busca)
14    visitados[cont]=v;
15
16    // Iniciar a fila e inserir o vértice de origem
17    fila = (TipoFila *) malloc (sizeof(TipoFila));
18    IniciaFila(fila);
19    Enfileira(v,fila);
```

Busca em Largura: Implementação em C

```
20      while(!Vazia(fila)){
21          vt = Desenfileira(fila);
22          // Encontrar vt na lista de adjacências de G e buscar vizinhos
23          for (nv = G->vertices; nv!=NULL; nv = nv->prox)
24               if(vt == nv->vert)
25                    for (na = nv->ladj; na != NULL; na = na->prox)
26                         // Enfileirar vizinhos e marcar como visitados
27                         if (FoiVisitado(na->vert,visitados,cont)==0){
28                              Enfileira(na->vert,fila);
29                              cont++;
30                              visitados[cont]=na->vert;
31                         }
32      }
33      imprimeVisita(visitados,cont);
34      free(fila);
35  }
```

Busca em Largura: Desempenho

- ⇒ Cada iteração em `while(!Vazia(fila))` elimina um vértice `v` da fila e percorre os vizinhos de `v`: cada vértice entra e sai apenas uma vez da fila, e cada arco é percorrido apenas uma vez. Assim, tem-se um consumo de tempo $O(V + A)$.
- ⇒ O algoritmo de busca em largura encontra a solução mais rasa, não necessariamente a melhor.

Busca em Profundidade

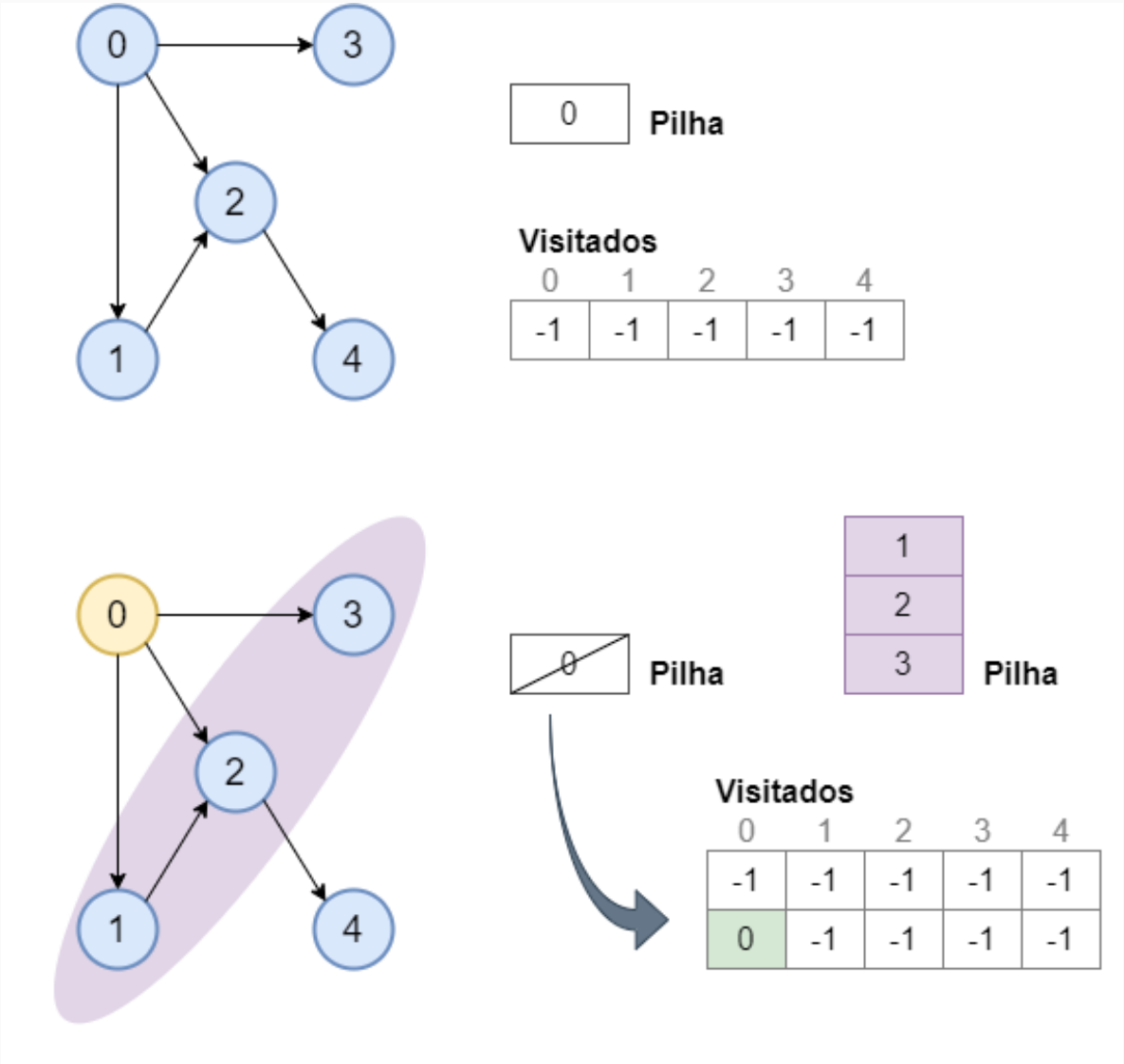
- A **busca em profundidade** é um algoritmo para caminhar no grafo, com a estratégia de buscar, o mais profundamente sempre que possível.
- Na busca, as arestas são exploradas a partir do vértice v mais recentemente descoberto, que ainda possui arestas não exploradas saindo dele. Quando não existem mais arestas a serem exploradas, a busca retrocede (anda para trás).
- Se todos os vértices já foram descobertos, então é o fim.
- A busca em profundidade é a base para os algoritmos de:
  - ordenação topológica, e
  - obtenção dos componentes fortemente conectados

Busca em Profundidade - Algoritmo usando Pilha

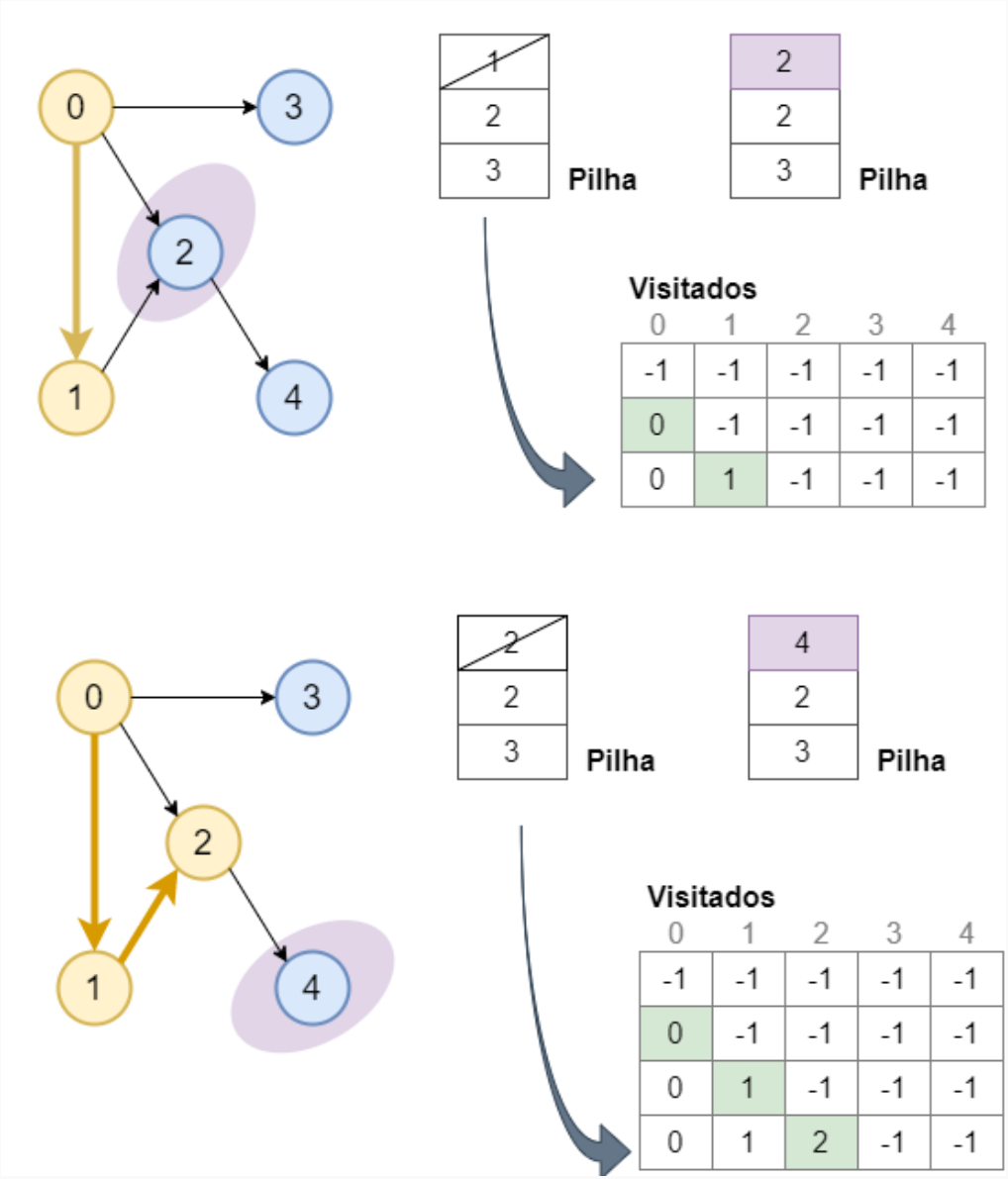
Uma possível implementação para um caminho em profundidade pode ser feita usando Pilha e um vetor de Visitados, da seguinte forma:

1. Inserir o vértice de início da busca na Pilha.
2. Obter o vértice (vt) do topo da pilha e adicionar o mesmo ao vetor Visitados.
3. Obter a lista de vértices alcançáveis a partir de vt e adicionar os que ainda não foram visitados (não estão no vetor Visitados) na Pilha.
4. Repetir passos 2 e 3 até que a Pilha esteja vazia.

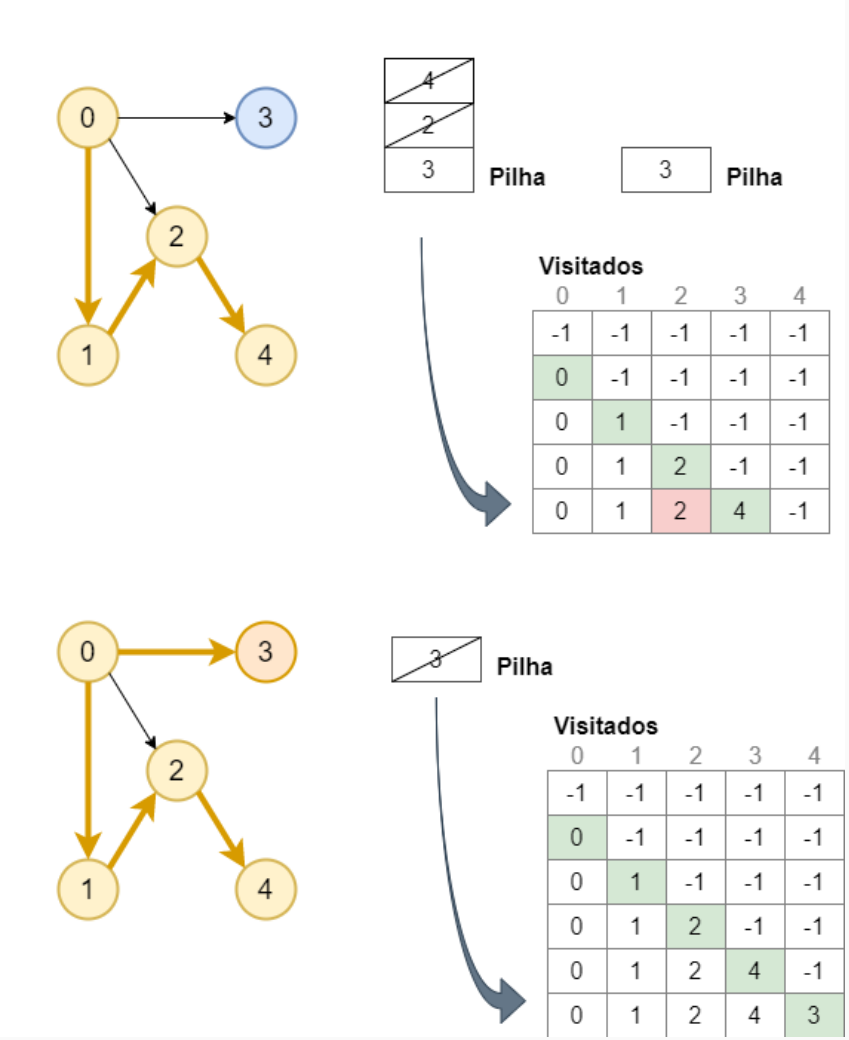
Busca em Profundidade - Algoritmo usando Pilha - Exemplo



Busca em Profundidade - Algoritmo usando Pilha - Exemplo



Busca em Profundidade - Algoritmo usando Pilha - Exemplo



Ordem da visita em profundidade: 0 1 2 4 3

Busca em Profundidade - Algoritmo usando Pilha

```
1  Entradas: Grafo G, Vértice v
2  variável local Pilha: TipoPilha
3  variável local Visita: vetor [nro de vértices de G]
4  para cada v em V[G] faça
5       visitado[v] = -1
6  fim para
7  criar Pilha
8  empilhar(v, Pilha)
9  enquanto (NãoVazia(Pilha)) faça
10     u = desempilha(Pilha)
11     se u não está no vetor Visita então
12          inserir u em Visita
13          para cada w alcançável a partir de u faça
14               se w não está no vetor Visita então
15                    empilhar(w, Pilha)
16               fim se
17          fim para
18     fim se
19 fim enquanto
```

Busca em Profundidade - Algoritmo usando Pilha

```
1  void BuscaEmProfundidade (Grafo G, int v) {
2     struct noVert *nv;
3     struct noAdj *na;
4     int *visitados;
5     TipoPilha *pilha;
6     int i=0, cont = 0, vt;
7
8     //Iniciar o vetor de visitados com -1 para todas as entradas
9     visitados = (int *) malloc (G->NumVert * sizeof (int));
10    for (i=0; i<G->NumVert; i++){
11       visitados[i] = -1;
12    }
13    imprimeVisita(visitados, G->NumVert-1);
14    //Iniciar a fila e inserir o vértice de origem
15    pilha = (TipoPilha *) malloc (sizeof(TipoPilha));
16    IniciaPilha(pilha);
17    Empilha(v,pilha);
```

Busca em Profundidade - Algoritmo usando Pilha

```
18      while(!VaziaPilha(pilha)){
19          vt = Desempilha(pilha);
20          if (FoiVisitado(vt,visitados,cont)==0){
21             visitados[cont]=vt;
22             cont++;
23             for (nv = G->vertices; nv!=NULL; nv = nv->prox)
24                 if(vt == nv->vert)
25                    for (na = nv->ladj; na != NULL; na = na->prox)
26                         if (FoiVisitado(na->vert,visitados,cont)==0){
27                              Empilha(na->vert,pilha);
28                         }
29          }
30      }
31      imprimeVisita(visitados,cont-1);
32      free(pilha);
33  }
```

Busca em Profundidade - Algoritmo usando Pilha

Funções auxiliares:

```
1  int FoiVisitado(int vert, int *vet, int tam){
2    int i;
3    for (i = 0; i<=tam; i++){
4      if (vet[i] == vert)
5          return 1;
6    }
7    return 0;
8  }
9
10 int imprimeVisita(int *vet, int tam){
11   int i;
12   printf("\nOrdem da visita: ");
13   for (i = 0; i<=tam; i++){
14      printf("(%d) ", vet[i]);
15   }
16   return 0;
17 }
```

Busca em Profundidade - Desempenho

As implementações para caminhos em largura e em profundidade apresentadas são bastante parecidas. Desta forma, é fácil ver que a complexidade da busca em profundidade também é $O(|V| + |A|)$, isto é, linear no tamanho da representação do grafo por listas de adjacências.

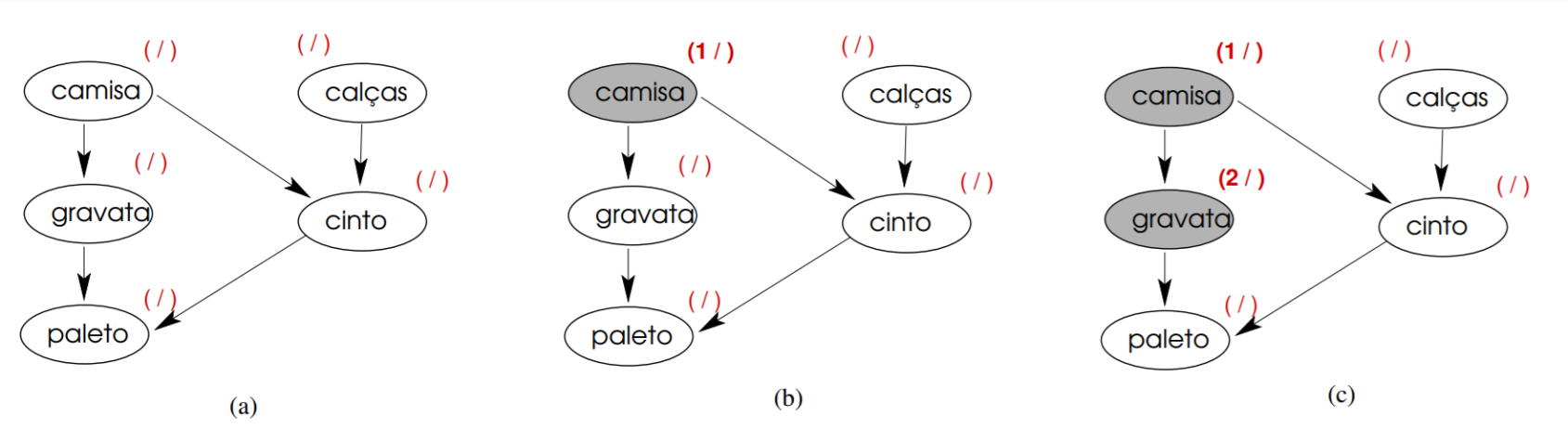
Busca em Profundidade - Marcando o Tempo

- O tempo de descoberta e de encerramento de um nó podem ser informações importantes a serem utilizadas em aplicações que utilizam busca em profundidade.
- A **ordenação topológica** é um exemplo de aplicação da busca em profundidade com tempos marcados num grafo direcionado acíclico. Os vértices são ordenados linearmente para indicar precedências entre eventos (vértices).

Busca em Profundidade - Aplicação - Ordenação Topológica

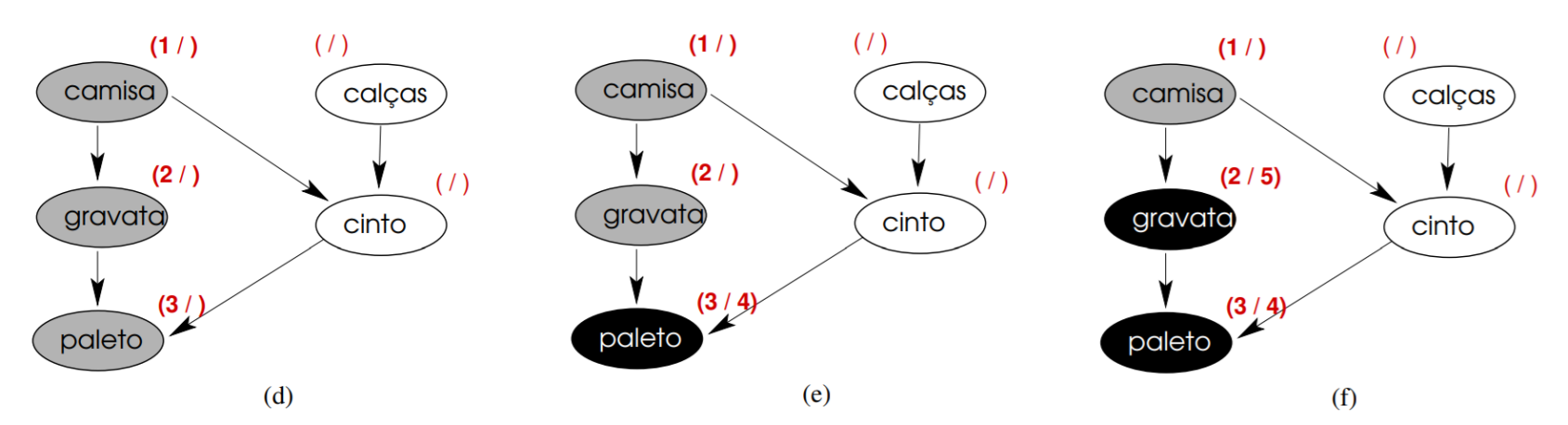
O algoritmo de Busca em Profundidade pode ser utilizado/adaptado para a marcação dos tempos de descoberta e término de vértice.

Exemplo de marcação dos tempos:



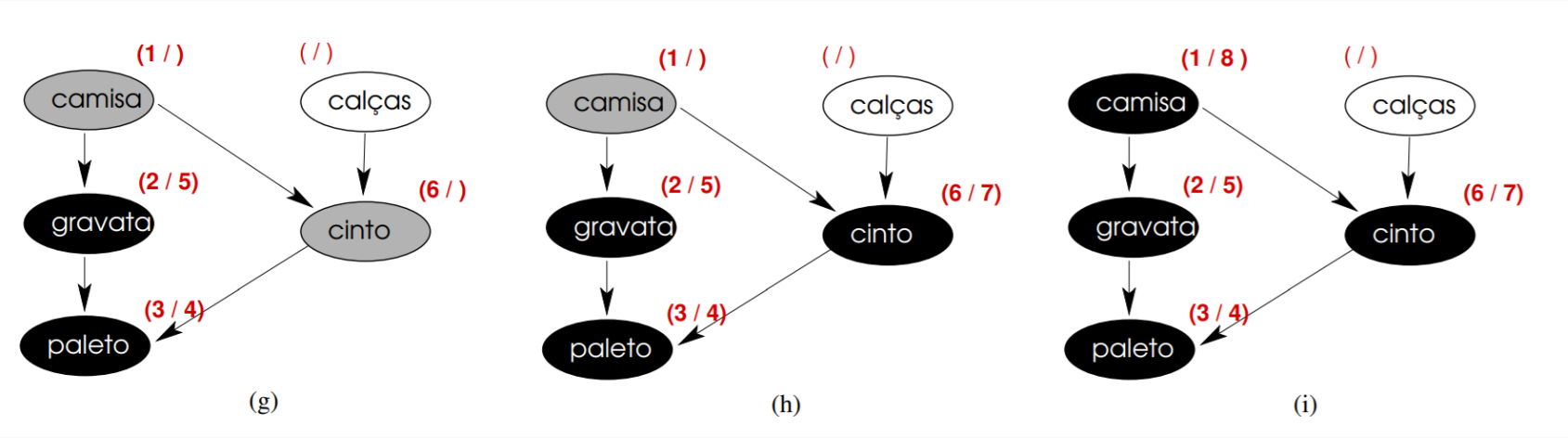
Busca em Profundidade - Aplicação - Ordenação Topológica

continuação da marcação dos tempos:



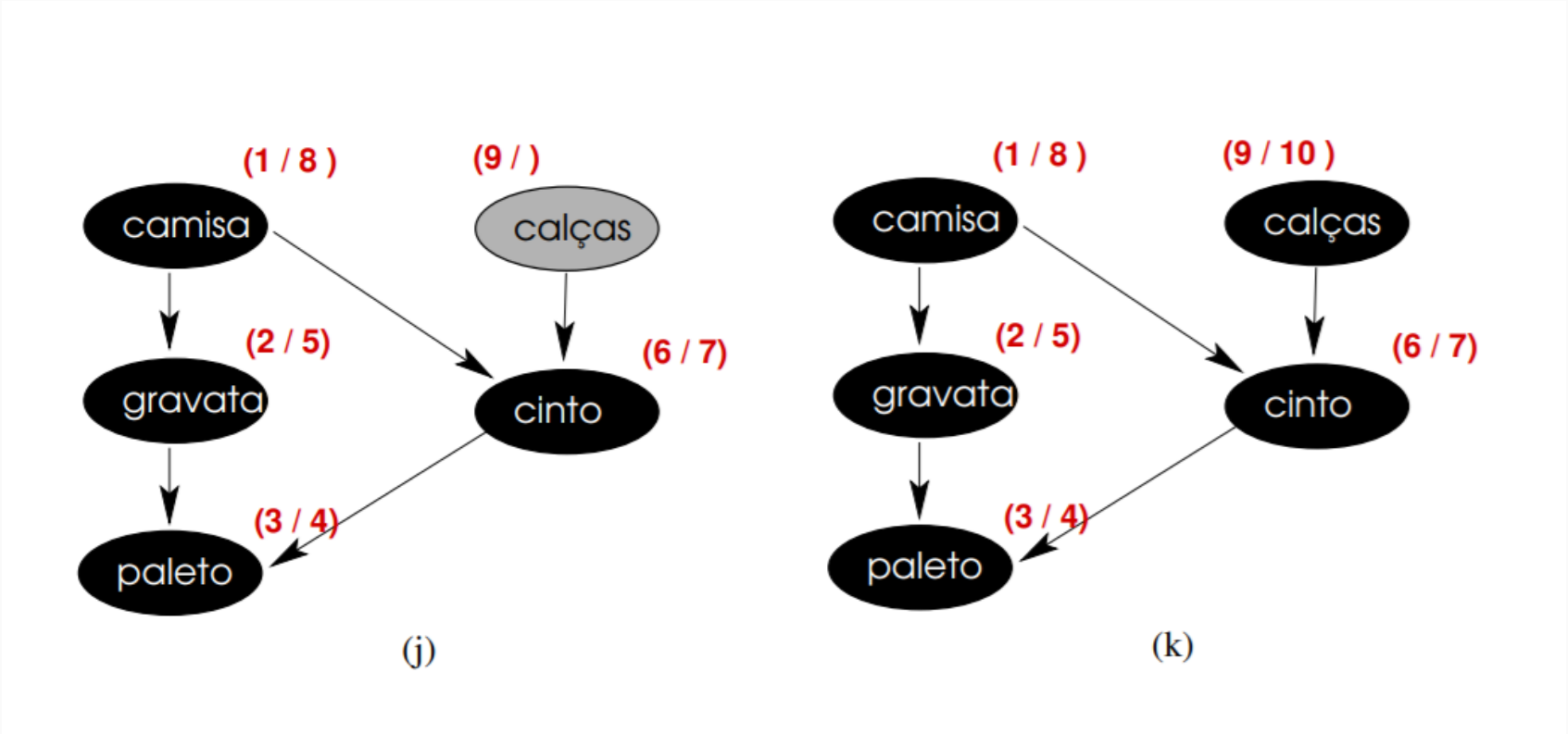
Busca em Profundidade - Aplicação - Ordenação Topológica

continuação da marcação dos tempos:



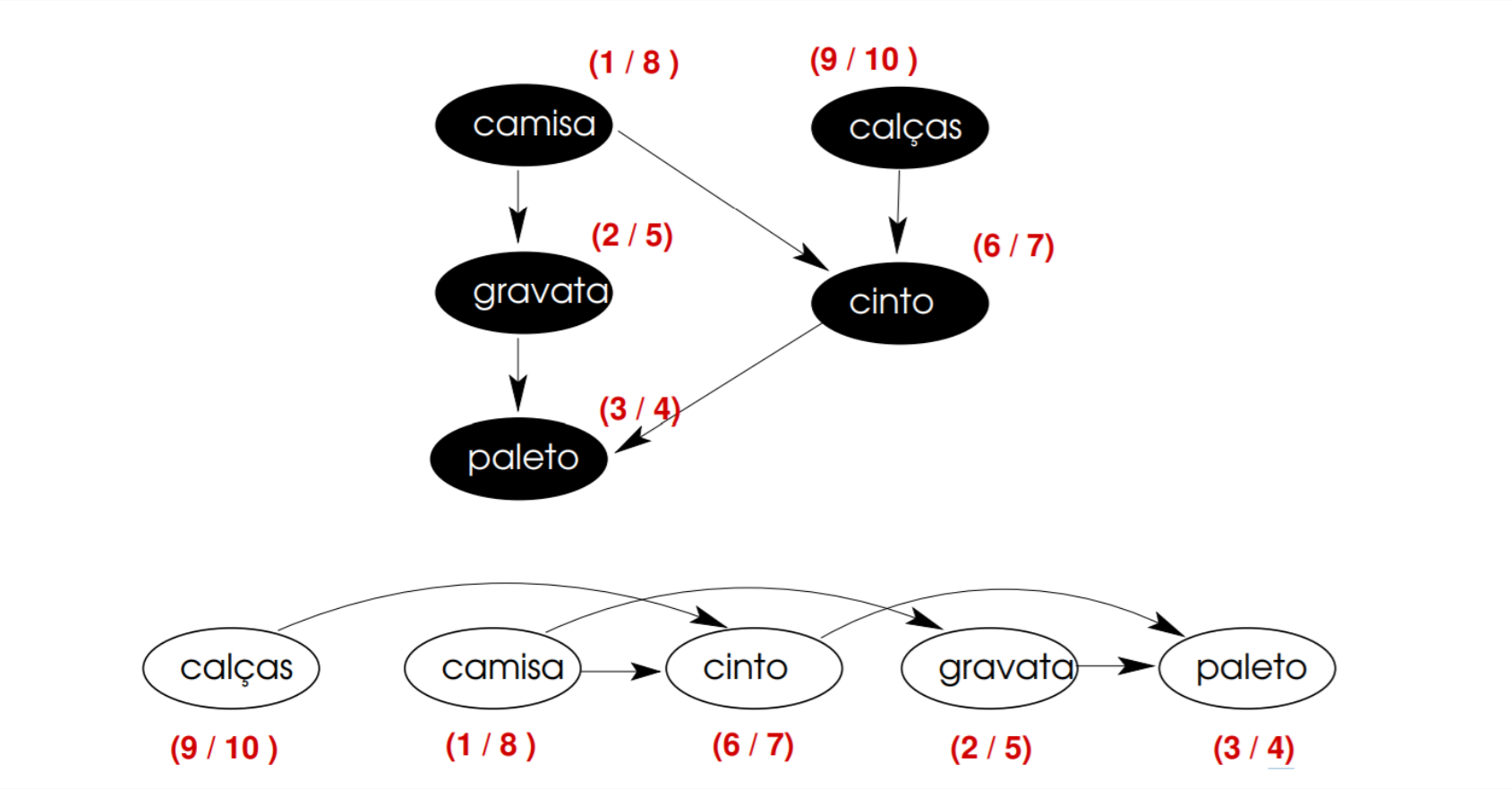
Busca em Profundidade - Aplicação - Ordenação Topológica

continuação da marcação dos tempos:



Busca em Profundidade - Aplicação - Ordenação Topológica

Término da marcação dos tempos:



A ordenação topológica do grafo organiza os vértices da esquerda para a direita, em **ordem decrescente de tempo de término**.

Busca em Profundidade - Aplicação - Ordenação Topológica

O algoritmo simples a seguir ordena topologicamente um grafo acíclico direcionado utilizando o algoritmo de Busca em Profundidade:

```
1 OrdenacaoTopologica(G)
2 inicio
3   Chamar BuscaEmProfundidade(G) para calcular o tempo de término t[v]
4     para cada vértice v.
5   A medida que cada vértice é terminado, inserir o vértice à frente
6     numa lista ligada.
7   Retornar a lista ligada de vértices.
8 fim
```

Busca em Profundidade - Adaptação para marcar tempos

```
1  BuscaProfundidadeTempo(Grafo g, vertice v)
2     tempo = 0
3     Criar o vetor visitados de tipo visita (vertice, tempoDescoberta,
4           tempoFinalizacao) e iniciar os tempos com -1
5     Criar pilha P
6     Empilhar v na pilha P
7     enquanto pilha P não for vazia faça
8         vt = vértice no topo da pilha P
9         k = posicão de vt em visitados
10        se visitados[k].tempoDescoberta = -1 então
11           visitados[k].tempoDescoberta = tempo++
12           para todos os vértices w alcançáveis a partir de vt faça
13              p = posicao de w no vetor visitados
14              se visitados[p].tempoDescoberta = -1 então
15                 Empilhar w na pilha P
16           se nenhum vértice w foi empilhado em P então
17             visitados[k].tempoFinalizacao = tempo++
18             Desempilhar pilha P
19        senão
20           visitados[k].tempoFinalizacao = tempo++
21           Desempilhar pilha P
```

Busca em Profundidade - Adaptação para marcar tempos

Implementação em Linguagem C - Estrutura especial para armazenar os tempos de descoberta e tempos de finalização dos vértices durante o percurso em profundidade:

```
1 struct visitaTempo {
2   int vert;
3   int tempoDescoberta;
4   int tempoFinalizacao;
5 };
```

Busca em Profundidade - Adaptação para marcar tempos

```
1  void PercursoProfundidadeTempo (Grafo G, int v) {
2     struct noVert *nv;
3     struct noAdj *na;
4     struct visitaTempo *visitados;
5     TipoPilha *pilha;
6     int tam=0, tempo=0, vt, finaliza, r;
7
8     //Iniciar o vetor visitados com -1 para todos os tempos dos vértices
9     visitados = (struct visitaTempo *) malloc (G->NumVert *
10                          sizeof (struct visitaTempo));
11    for (nv=G->vertices; nv!=NULL; nv=nv->prox){
12       visitados[tam].vert = nv->vert;
13       visitados[tam].tempoDescoberta = -1;
14       visitados[tam].tempoFinalizacao = -1;
15       tam++;  //Tamanho do vetor
16    }
17    //Iniciar a fila e inserir o vértice de origem
18    pilha = (TipoPilha *) malloc (sizeof(TipoPilha));
19    IniciaPilha(pilha);
20    Empilha(v,pilha);
```

Busca em Profundidade - Adaptação para marcar tempos

```
21      while(!VaziaPilha(pilha)){
22          vt = Topo(pilha);
23          if (FoiVisitadoTempo(vt,visitados,tam)==0){
24             MarcaVisitaTempo(vt, tempo, 'D', visitados,tam); tempo++;
25             for (nv = G->vertices; nv!=NULL; nv = nv->prox)
26                 if (vt == nv->vert){
27                    finaliza = 1;
28                    for (na = nv->ladj; na != NULL; na = na->prox)
29                         if (FoiVisitadoTempo(na->vert,visitados,tam)==0){
30                              Empilha(na->vert,pilha); finaliza = 0; }
31                    if (finaliza == 1){
32                        r = MarcaVisitaTempo(vt,tempo,'F',visitados,tam);
33                        if (r == 1) tempo++;
34                        vt = Desempilha(pilha); } }
35          }
36          else { //Vértice no topo da pilha já visitado
37              r = MarcaVisitaTempo(vt,tempo,'F',visitados,tam);
38              if (r == 1) tempo++;
39              vt = Desempilha(pilha); }
40      }
41      imprimeVisitaTempo(visitados,tam);  free(pilha);
42  }
```

Busca em Profundidade - Adaptação para marcar tempos

Funções auxiliares:

```
1   int MarcaVisitaTempo(int vt, int pos, char tipo, struct visitaTempo
2                        *vet, int tam){
3   for (int i = 0; i<tam; i++)
4     if (vet[i].vert == vt)
5           if (tipo == 'D')
6               vet[i].tempoDescoberta = pos;  return 1;
7           else {
8               if (vet[i].tempoFinalizacao == -1){
9                   vet[i].tempoFinalizacao = pos;  return 1; }
10              else return 0;
11          }
12  return 0;
13  }
14
15 int FoiVisitadoTempo(int vert, struct visitaTempo *vet, int tam){
16    for (int i = 0; i<tam; i++)
17      if ((vet[i].vert == vert) && (vet[i].tempoDescoberta != -1))
18         return 1;
19    return 0;
20  }
```

Bibliografia

- Nívio Ziviani. Projeto de Algoritmos com Implementações em Pascal e C.
- Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein. Algoritmos - Teoria e Pratica.