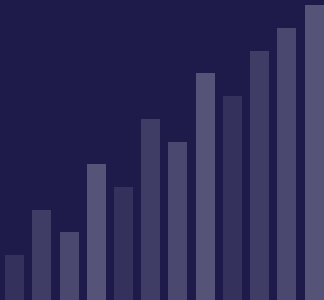

AULA 29 · BLOCO 4: GRAFOS

Busca em profundidade e ordenação topológica

Seções 22.3 e 22.4

Estruturas de dados e seus algoritmos · José Dihego · IFBA



Roteiro da aula

Ao final, você será capaz de

- ▶ Executar a DFS e registrar tempos de descoberta e de término
- ▶ Classificar arestas (árvore, retorno, avanço, cruzamento)
- ▶ Ordenar topologicamente um grafo acíclico pela ordem de término
- ▶ Medir $V + E$ operações e o efeito do cache no tempo



Teoria

DFS, estrutura de parênteses, tipos de arestas, ordenação topológica



Prática em C

O exemplo de se vestir (Figura 22.7) em C



Experimento

DFS iterativa em DAGs de até 2 milhões de vértices



Conexões

Compiladores, builds, planilhas e verificação de deadlocks

Busca em profundidade

- ▶ Vai fundo: explora o vizinho mais recente antes de voltar (pilha, ou recursão).
- ▶ $\text{DFS-VISIT}(u)$: marca u cinza, $u.d \leftarrow \text{tempo}$; visita os vizinhos brancos; marca preto, $u.f \leftarrow \text{tempo}$.
- ▶ Recomeça de todo vértice ainda branco: produz uma **floresta**.
- ▶ Custo $\Theta(V + E)$.

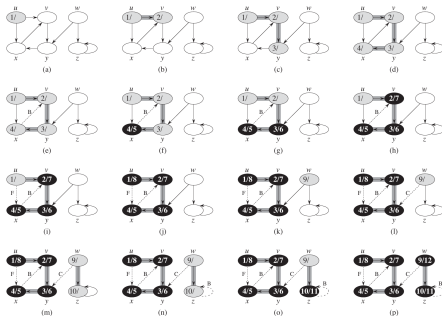


Figura 22.4 de Cormen et al. (2009) — DFS num grafo direcionado: d/f = tempos de descoberta e término; B, F, C marcam arestas de retorno, avanço e cruzamento.

Parênteses e tipos de arestas

Teorema 22.7 (parênteses): para u e v , os intervalos $[u.d, u.f]$ e $[v.d, v.f]$ são disjuntos ou um contém o outro.

Tipos de aresta (u, v) :

- ▶ **árvore:** v descoberto por u ;
- ▶ **retorno:** v é ancestral (cinza);
- ▶ **avanço e cruzamento:** as demais.

Lema 22.11: o grafo é acíclico $\iff$ a DFS não encontra aresta de retorno.

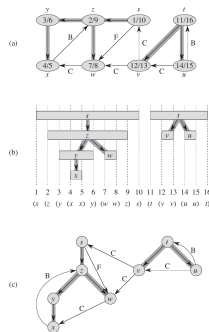


Figura 22.5 de Cormen et al. (2009) — (a) tempos da DFS; (b) os intervalos $[d, f]$ formam uma estrutura de parênteses; (c) arestas classificadas.

Ordenação topológica

Num grafo direcionado acíclico (DAG), uma ordem linear em que toda aresta (u, v) tem u antes de v .

TOPOLOGICAL-SORT(G): rode a DFS; ao terminar cada vértice, insira-o na **frente** de uma lista.

Teorema 22.12: num DAG, toda aresta (u, v) tem $v.f < u.f$. Custo $\Theta(V + E)$.

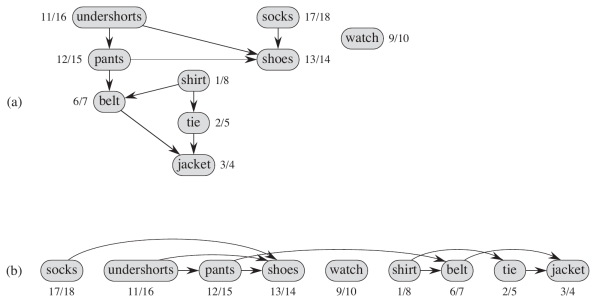


Figura 22.7 de Cormen et al. (2009) — (a) o que vestir antes de quê, com tempos d/f ; (b) os vértices em ordem decrescente de término.



Prática em C

- ▶ DFS recursiva com cores e tempos
- ▶ Numeração escolhida para reproduzir a Figura 22.7
- ▶ Imprime a ordem e os tempos d/f

DFS e ordenação topológica em C (1/3)

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  /* O grafo da Figura 22.7: o que vestir antes de quê. Uma aresta u -> v diz
5     "u antes de v". A DFS numera o fim de cada vértice; em ordem decrescente de
6     término, os vértices formam uma ordenação topológica. */
7  #define V 9
8  /* Numeração escolhida para a DFS (vértices em ordem, vizinhos em ordem) repetir
9     os tempos d/f da Figura 22.7: camisa 1/8, gravata 2/5, paletó 3/4, ... */
10 const char *nome[V] = {"camisa", "gravata", "paletó", "cinto", "relógio",
11                        "cuecas", "calças", "sapatos", "meias"};
12 int adj[V][V];           // matriz de adjacência (grafo pequeno)
13 int cor[V], d[V], f[V], tempo; // 0 branco, 1 cinza, 2 preto
14 int ordem[V], k_ordem = V;
```

DFS e ordenação topológica em C (2/3)

```
16 void visita(int u) {                                // DFS-VISIT
17     cor[u] = 1;
18     d[u] = ++tempo;                                  // descoberta
19     for (int v = 0; v < V; v++)
20         if (adj[u][v] && cor[v] == 0) visita(v);
21     cor[u] = 2;
22     f[u] = ++tempo;                                  // término
23     ordem[--k_orden] = u;                            // insere na frente da lista
24 }
```

DFS e ordenação topológica em C (3/3)

```
26 int main(void) {
27     int arestas[][2] = {{0, 1}, {0, 3}, {1, 2}, {3, 2}, {5, 6}, {5, 7},
28                         {6, 3}, {6, 7}, {8, 7}};          // u -> v: vista u antes de v
29     for (int k = 0; k < 9; k++) adj[arestas[k][0]][arestas[k][1]] = 1;
30     int mostrar;
31     if (scanf("%d", &mostrar) != 1 || mostrar < 1 || mostrar > V) return 1;
32     for (int u = 0; u < V; u++)                          // DFS: inicia em todo vértice branco
33         if (cor[u] == 0) visita(u);
34     for (int i = 0; i < mostrar; i++)
35         printf("%s%s", nome[ordem[i]], i < mostrar - 1 ? " " : "\n");
36     for (int u = 0; u < V; u++) printf("%s %d/%d\n", nome[u], d[u], f[u]);
37     return 0;
38 }
```



Experimento

- ▶ DAG aleatório com $E = 8V$, ordem escondida por uma permutação
- ▶ DFS iterativa (a recursiva estouraria a pilha)
- ▶ Verificação: toda aresta respeita a ordem produzida

DFS em DAGs grandes

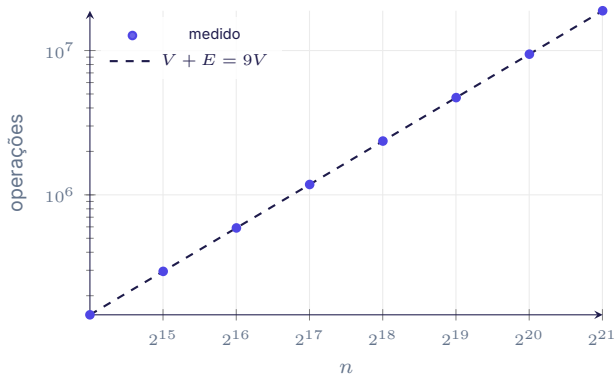
Hipótese. A DFS descobre cada vértice uma vez e examina cada aresta uma vez: com $E = 8V$, exatamente $9V$ operações. No modelo RAM, o tempo também seria linear. Mas a DFS salta pela memória (como a lista ligada da aula 14): esperamos o tempo crescer **mais** que linearmente quando o grafo sair do cache.

Método

1. $V = 2^{14}$ a 2^{21} ; arestas sempre de rótulo menor para maior (acíclico)
2. Pilha explícita com a posição na lista de cada vértice
3. Conferência da ordem topológica em todas as arestas
4. Tempo com CLOCK_MONOTONIC

🧪 EXPERIMENTO

Operações: exatamente $V + E$



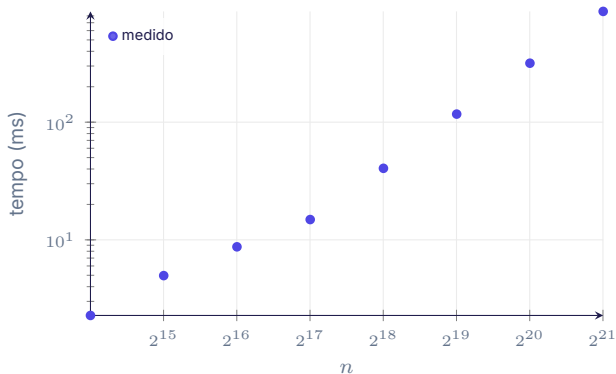
n	medido	razão
16.384	147.456	1,000
32.768	294.912	1,000
65.536	589.824	1,000
131.072	1.179.648	1,000
262.144	2.359.296	1,000
2.097.152	18.874.368	1,000

razão = medido $\div$ $V + E = 9V$

✓ desvio máx. 0,0%

Contagem exata, e a ordem produzida respeitou todas as 16 milhões de arestas do maior grafo.

Tempo: o cache aparece



Até $\approx 10^5$ vértices o tempo é linear;
depois cresce mais rápido: cada
vértice novo está longe na memória.

EXPERIMENTO

O que o experimento mostrou

A contagem é **exatamente** $V + E$ e a ordem topológica é sempre válida. O tempo, porém, só é linear enquanto o grafo cabe no cache: depois, cada acesso aleatório custa uma ida à RAM. Bibliotecas de grafos de alto desempenho renumeram os vértices para que vizinhos fiquem próximos na memória.

➤ Reproduza: `gcc -O2 aula29_experimento.c -o exp -lm && ./exp`



Conexões

- ▶ Compiladores e builds
- ▶ Planilhas
- ▶ Sistemas operacionais
- ▶ Gestão de projetos e dados

Onde isso aparece na computação

Compiladores e builds

make, Maven, Bazel e gerenciadores de pacotes (npm, apt) ordenam topologicamente as dependências; um ciclo é um erro de configuração.

Planilhas

Ao mudar uma célula, a planilha recalcula as dependentes em ordem topológica e detecta referências circulares com a DFS.

Sistemas operacionais

Detectar *deadlock* é achar um ciclo no grafo de espera entre processos e recursos: uma aresta de retorno na DFS.

Gestão de projetos e dados

Diagramas PERT/CPM, pipelines de dados (Airflow) e redes neurais (o grafo de computação do PyTorch) são DAGs executados em ordem topológica.

Para casa

Exercícios

1. (22.3-2) Mostre a DFS do grafo da Figura 22.6 com tempos e tipos de arestas.
2. (22.3-7) Reescreva a DFS com uma pilha explícita, sem recursão.
3. (22.4-1) Mostre a ordenação topológica do DAG da Figura 22.8.
4. (22.4-3) Decida se um grafo não direcionado tem ciclo em $O(V)$, independente de E .
5. Renumere os vértices do experimento em ordem de BFS antes da DFS e meça de novo o tempo.

Leitura

Cormen et al., seções 22.3 e 22.4.

Código

josedihego.net/estruturas