Department of Computer Science and Engineering
Analysis of Algorithms
Kjell Post



---

# Homework 4 — Graph algorithms

**Sections in CLRS:** pp. 525–530 (repeat section B.4 if necessary)

**Exercises:** 22.1–1

**Sections in CLRS:** pp. 531–534 (up to "shortest paths")

- We skip the correctness proofs and instead concentrate on breadth-first search and its properties.
- The colors GRAY, WHITE and BLACK are mostly used to argue about the correctness of breadth-first search and are not necessary in an implementation. In addition, we will not build predecessor graphs so you can ignore the $\pi$ variable in the algorithm. This leads to the following simplified algorithm which uses $d = \infty$ instead of WHITE.

```
BFS(s)
    for each u ∈ V − {s}
        do d[u] ← ∞
    d[s] ← 0
    Q ← {s}
    while Q ≠ ∅
        do remove u from Q
           for each v ∈ Adj[u]
               do if d[v] = ∞
                  then d[v] ← d[u] + 1
                       put v onto Q
```

- Note that because one is interested in the distance from the start vertex $s$, BFS doesn't necessarily visit all the vertices in the graph.

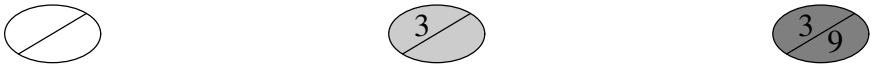
**Exercises:** 22.2–2, 22.2–3

**Sections in CLRS:** pp. 540–543, 546–547

- Again, ignore the predecessor graphs ($\pi$) and proof of correctness. Follow the algorithm in the book (page 478) but ignore line 3 in DFS and line 6 in DFS-VISIT.
- Note that DFS visits all vertices due to the outer loop. The results can however vary depending on how vertices are selected.
- For a directed graph, edges are marked as belonging to one of the following categories: tree edges, back edges, forward edges and cross edges. For undirected graphs there are only tree and back edges.

**Exercises:** 22.3–2, 22.3–7

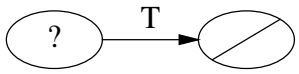
- For 22.3–2 you do not have to produce a sequence of pictures like in Figure 22.4 in the book: it is enough to mark $d[u], f[u]$ for each vertex $u$ and to classify the edges as described above. You do not even have to color the vertices if you write $d[u], f[u]$ next to each vertex. For instance, a gray vertex would have $d[u]$ defined, but not $f[u]$.



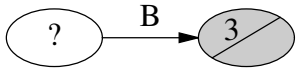

WHITE GRAY BLACK

The edge category can be summarized with the following situations:

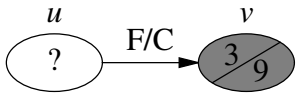
Tree edge




Back edge




$d[u] < d[v]$: Forward edge (F)
$d[u] > d[v]$: Cross edge (C)




**Sections in CLRS:** pp. 549–551

**Exercises:** 22.4–1, 22.4–3, 22.4–5

- If you succeed in giving an algorithm for 22.4–3 but think that it runs in $O(V + E)$, chances are your analysis is not tight enough. Consider an acyclic, undirected forest with $|E| \leq |V| - 1$ (theorem B.2, page 1085 in the book). How many edges will your algorithm visit in the worst case?

**Sections in CLRS:** We will not consider section 22.5 "Strongly connected components" but you should know what a SCC is (see also page 1082) and that they can be identified in $O(V+E)$ time.

Generally speaking, bredth and depth-first search are useful algorithms because they produce useful information for other algorithms like topological sorting, identifying planar graphs (those that can be drawn on a paper without intersecting edges), etc.

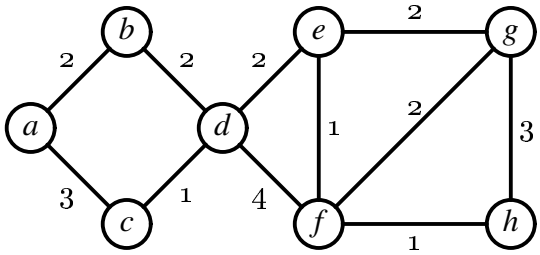



Figure 1: An undirected graph.

**Sections in CLRS:** pp. 561–566

**Exercises:** 23.1–1

**Sections in CLRS:** 570–573 (Prim's algorithm)

- If MST-PRIM is difficult to understand you can study the following "higher-level" version of Prim's algorithm first.

  MST-PRIM$(G,w,r)$
  ▷ Suppose that $V=\{1,2,\ldots,n\}$.
  ▷ $(U,V-U)$ partitions the graph in two halves.
  $U \leftarrow \{1\}$
  **while** $U \neq V$
  **do** let $(u,v)$ be the cheapest edge such that $u \in U$ and $v \in (V-U)$.
  $\pi[v] \leftarrow u$ ▷ $u$ is a parent of $v$ in the MST
  $U \leftarrow U \cup \{v\}$
  **return** $\pi$ ▷ return the MST

  The trick is of course to repeatedly find the cheapest edge crossing $(U,V-U)$ while the border is moving along. The book employs an efficient way to find the cheapest edge by having a priority $Q$, sorted by the cost of those edges crossing $(U,V-U)$.

**Exercises:** Find a minimum spanning tree for the graph in figure 1 using Prim's algorithm.

**Sections in CLRS:** Read pp. 580–583, skip "representing shortest paths" and continue with pp. 585–587. Then study Dijkstra's algorithm, pp. 595–599, but don't go through the proof until you have done exercise 24.3–1.

**Exercises:** 24.3–1, 24.3–2, 24.3–3.

**Sections in CLRS:** Read pp. 620–627. Again you do not need to concern yourself with representing shortest paths (page 621, line 11 and down to the bottom).

**Exercises:** 25.1–2. Think about why FASTER-ALL-PAIRS-SHORTEST-PATHS works even if $n$ is not an exact power-of-two.

**Sections in CLRS:** Study pp. 629–632 (up to "Constructing a shortest path") carefully.

**Exercises:** 25.2–1, 25.2–4, 25.2–6.

- Exercise 25.2–1 may seem like a lot of work, but will give you the proper insight into solving the other two exercises.

  Also show the cheapest path from vertex 3 to vertex 1.

- Exercise 25.2–4 describes the version of Floyd-Warshall's algorithm that is used in practice. The important question here is whether you can do it using only *one* matrix $D$.

- If you get stuck on 25.2–6 I would suggest that you construct a small example with a negative cycle.