

IFT2015 :: A17

Miklós Csűrös

October 2, 2017

## 6. Graphs

## 6.1 Graphs and Representation

**Definition 6.1.** An **undirected graph** is a pair  $(V, E)$  where  $E \subseteq \binom{V}{2}$  (Non-ordered pairs).  $V$  is the set of **Sum-Food** and  $E$  is the set of **edges**.

**Definition 6.2.** A **directed graph** is a pair  $(V, E)$  where  $E \subseteq V \times V$  (ordered pairs).  $V$  is the set of **nodes** or **vertices** and  $E$  is the set of **arcs**.

**Adjacency matrix.** This is a matrix  $V \times V$ , where the cell  $A[u, v]$  includes information on the  $uv$  ridge.

**Adjacency lists.** Representation by **adjacency lists** is a set  $\text{Adj}[u]$  lists for each node  $u$  that stores the set  $\{v: uv \in E\}$ .

Memory Usage:  $\mathcal{O}(|E| + |V|)$ , and it's better than the matrix in for a *sparse* graph with  $E = \mathcal{O}(|V|^2)$ . In practice, it is possible to store  $\text{Adj}$  as an array, or a linked list: the structure must support the neighbors.

1 [\(en\)](#) : adjacency list

## 6.2 Chart paths

It runs through a graph from a **start node**  $s$ , following the logic of tree traversal algorithms (§4.3). In order to recognize cycles, is marked vertices  $V = \{0, 1, \dots, N-1\}$  for the par-course (by "coloring"). We examine both approaches to explore a graph: course by depth and course by width.

F IG. 1: Adjacency lists (in boxes) in a graph with 4 vertices. [\[Wikimedia Commons\]](#)*In depth*

In the **course** of algorithm 2 **deep** (*depth-first search*), it color the vertices by green at the beginning, then by yellow (visiting prefix) and fi-red (in postfixed visit). In the following version, we store parent bond  $[u]$  for each node that gives the vertex from which  $u$  reaches for the first time (when it is *discovered*  $u$ ). Except for application wants to follow the links, just keep the coloring of the vertices for make the route.

2 [\(en\)](#) : Course in profondeur

```
(init) the parent [s] ← s; for u ← 0, 1, . . . , N - 1 do color [u] Green ←
DFS (s) // long path from node s
D1 color [i] ← yellow // pre-visit s
D2 for st ∈ Adj [s] do // for each vertex adjacent to t s
D3 color [t] = green Then DFS (t); Parent [a] s // ← visit the neighboring t
D4 color [s] ← red // post-visit s
```

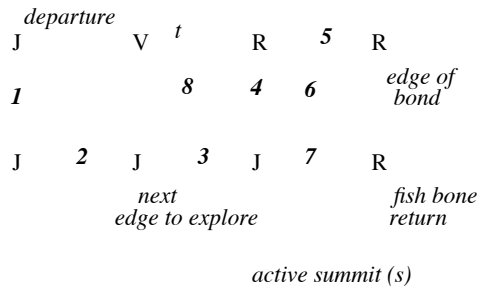


FIG. 2: Depth Course (DFS). When visiting a ridge  $st$  for the pre- the color of the vertex  $t$  can be **Green**:  $st$  and top edge  $t$  discovered for the first time (that is, a **connecting** edge) **yellow**:  $st$  edge first discovered time, but  $t$  is known (this is an edge **re-Tower**), or **red**, ever since we visit all the adjacent edges, including  $ts$ , before push  $t$ .

100%

**Calculation time.** The route ends in  $O(|V| + |E|)$  time: consider each edge exactly twice (line D2), and each vertex crayon exactly three times.

**Related components.** The whole graph is explored by launching DFS from any summit that remains green green.

**for**  $s \leftarrow 0, 1, \dots, n-1$  **color** **do** **if**  $[s] = \text{green}$  **Then** DFS ( $s$ )

FIG. 3: DFS tree in a graph of 250 nodes. [SW 2011]

It is easy to see that the depth trail can be used to identify the related components of the graph. The connecting edges form a set of trees, or a **deep forest** covering all the tops. By following the parent links, there is a path between one vertex  $v$  and the starting vertex. (This is the path formed by the yellow vertices when  $v$  is discovered.)

**Bipartite graph.** A **bipartite graph** is an undirected graph  $(V, E)$  in wherein  $V$  can be partitionnée into two sets  $V_1$  and  $V_2$  ( $V_1 \cup V_2 = V$ ;  $V_1 \cap V_2 = \emptyset$ ) such that all the edges pass between  $V_1$  and  $V_2$ : if  $uv \in E$ , then  $u \in V_1$  and  $v \in V_2$  or  $u \in V_2$  and  $v \in V_1$ . We can test if a graph is bipartite during the journey: it is enough to partition the vertices when they are discovered, and to test whether the return edges pass between the two sets.

```
(init) Sheet [i] ← 1; for  $u \leftarrow 0, 1, \dots, N-1$  do color [u] Green ←
DFS-BIP (s) // test whether the connected component of s is bipartite
1 color [i] ← yellow
2 for  $st \in \text{Adj} [u]$  do
3 if color [a] = green Then
4 Partition [t] = 3 - partition [s] // connecting edge
5 DFS-BIP (t) // 1 ↔ 2
6 else if color [a] = yellow Then // parent or edge back
7 partition [s] = Sheet [t] then die ("Cycle of odd length")
```

FIG. 4: Bipartite graph: vertices partition- in two (left and right sides of  $V_1$  and  $V_2$ , no ridges between vertices on the same side.

**Topological sorting.** Let  $G = (S, A)$  a **directed acyclic graph**. **Sorting topological** seeks a permutation of vertices such that if  $uv \in A$ , then  $u$  is before  $v$  in order. In fact, the inverse order of postfixes is a topological sorting. It is then sufficient to use a stack  $P$  to store the vertices in post-visit.

3 (en) : topological sort

D4 color [s] ← red; P.push (u)

At the end, P.pop scrolls the vertices in the order of topological sorting.

FIG. 5: An acyclic graph oriented at 13 nodes. [SW 2011]

Width

At a **width 4 courses** (*breadth-first search*), neighbors are threaded in a FIFO (tail) queue - depth path corresponds to the usage of a stack. In the version below, the distance *d* is maintained from top of source. Connecting edges form a **tree width** covering a related component. In this tree, rooted at the summit of *s* departure, while *d [u]* is the depth of node *u*.

```
1 BFS (s)
2 for u ← 0, 1, . . . , N - 1 do color [u] ← yellow
3 d [i] ← 0; Parent [s] s ←
4 Q.enqueue (s); color [s] ← yellow
5 while Q ≠ ∅
6   u ← Q.dequeue ()
7   for uv ∈ Adj [u] do
8     if color [v] = green Then
9       d [v] ← d [u] + 1; Parent [v] ← u
10      Q.enqueue (v); color [v] ← yellow
11   end
12 end
13 color [u] ← red
14 end
```

Calculation time. The route takes  $O(|V| + |E|)$  with adjacency lists.

Shorter paths. At the end of the route, *d [u]* is the minimum length of a path between *s* and *u* for every vertex *s*. This shortest can be traced path by following the parent links.

100% (en) : breadth first

BFS for shortest paths (250 vertices)  
nodes. [SW 2011]

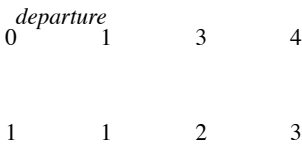


Fig. 8: The course in width discovers the distance from the starting point.