IFT2015 :: A17 Miklós Cs˝urös 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 × V, where the cell A [u, v] includes information on the uv ridge.

Adjacency lists. Representation by **adjacency lists** is a set [1] Adj [u] lists for each node u that stores the set {v: uv ∈ E}. Memory Usage:卷 (| E | + | V |), and it's better than the matrix in For a *sparse* graph with E = O (| V | 2). In practice, it is possible to store Adj as an array, or a linked list: the structure must support the neighbors.

1 (en) : adjacency list

6.2 Chart paths

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

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,. . . , N - 1} for the par-course (by "coloring"). We examine both approaches to explore a graph: course by depth and course by width.

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 profonderu

(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 **if** [t] = green **Then** DFS (t); Parent [a] s // ← *visit the neighboring* t
D4 color [s] ← red // *post-visit* s

departure
J V *t* R *5* R
1 *8* *4* *6* *edge of bond*
J *2* J *3* J *7* R
next edge to explore *fish bone return*
active summit (s)

100 %

F IG. 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 blush t.

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 $s \leftarrow 0, 1, \ldots n-1$ color **do if** [s] = green **Then** DFS (s)

DFS for paths (250 vertices)

F IG. 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 partionnée into two sets V_1 and V_2 (V_1 $V_2 = \cup$ V; $\cap$ V_1 V_2 = 3) 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 V_2 and $u \in v \in V_1$. 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 ← 0, 1,. . . , N - 1 do color [u] Green ←
   DFS- BIP (s)                    // test whether the connected component of s is bipartite
 color [i] ← yellow
 for st ∈ Adj [u] do
    if color [a] = green Then                     // connecting edge.
       Partition [t] = 3 - partition [s]             // 1 ↔ 2
       DFS- BIP (t)
    else if color [a] = yellow Then             // parent or edge back
       partition if [s] = Sheet [t] then die ( "Cycle of odd length")
```

F IG. 4: Bipartite graph: vertices partition-born 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** [3] 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.

D4 color [s] ← red; P.push (u)

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

3 (en) : topological sort

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

F IG. 6: Order of visit and topological sorting. [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.

```
BFS (s)
for u ← 0, 1,. . . , N - 1 do color [u] ← yellow
d [i] ← 0; Parent [s] s ←
Q.enqueue (s); color [s] ← yellow
while Q = 3 C
    u ← Q.dequeue ()
    for uv ∈ Adj [u] do
        if color [v] = green Then
            d [v] ← d [u] + 1; Parent [v] ← u
            Q.enqueue (v); color [v] ← yellow
        end
    end
    color [u] ← red
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 % 4 (en) : breadth first

BFS for shortest paths (250 vertices)

F IG. 7: BFS on a graph of 250 nodes. [SW 2011]

departure

0 1 3 4

1 1 2 3

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