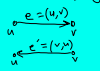
Basic Graph Algorithms

Digraph Representations
1. Adjacency matrix
2. Adjacency lists

Given a (di)graph G,
$G.V$ = set of vertices, say $\{v_1, \ldots, v_n\}$

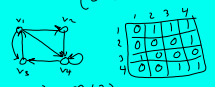
$G.E$ = set of edges $\subseteq G.V \times G.V$



Adjacency matrix: $n \times n$ 0-1 matrix
$n = |G.V|$
$G.V = \{v_1, \ldots, v_n\}$

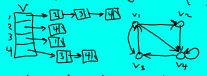
$A = (a_{ij})$ is the adj. matrix of G, then $\forall i,j \quad 1 \le i,j \le n$

$$a_{ij} = \begin{cases} 1 & \text{if } (v_i, v_j) \in G.E \\ 0 & \text{otherwise} \end{cases}$$



size is $\Theta(n^2)$

Adjacency list rep
$V[1..n]$ of linked lists,
the $V[i]$ list is of the vertices adjacent from v_i.
size is $\Theta(n+m)$ where $m = |G.E|$
more compact if, say $m = o(n^2)$



We will use the adjacency list rep exclusively. $n := |G.V| = V$, $m := |G.E| = E$

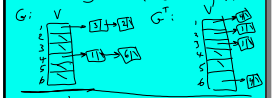
"linear time" means $\Theta(n+m)$
[note $0 \le m \le n^2$]

(undirected graphs are represented as digraphs where each undirected edge is rep by 2 directed edges:

The transpose G^T of a graph G is obtained by reversing the direction of each edge:

[take transpose of the adjacency matrix of G to get that of G^T.]

Computing G^T (adj. list rep)



Searching

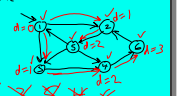
Breadth-First Search (BFS)
Depth-First Search (DFS)
Both linear time.

BFS(v) v = source vertex

Let Q be an empty queue of vertices
Visit(v) — application-dependent
mark v as visited
v.d := 0
v.π := NIL
Q.insert(v)
while Q is not empty:
u := Q.remove()
Visit u
mark u as visited
for all w adjacent from u
if w is unmarked:
w.d := u.d + 1
w.π := u
Q.insert(w)
end-if
end for-all
end-while

O(1) O(1) O(1)

Ex:
BFS(1)



Q: ~~X~~ ~~X~~ ~~X~~ ~~X~~ ~~X~~ ~~X~~

u.d = shortest distance from v to u (unweighted — # of edges)

Recovering a shortest path:
After w.d := u.d + 1
add the line w.π := u

In the example: draw in the red edges $(w.\pi, w)$ for all $w \ne v$
The resulting rooted tree is the BFS tree. [Run time is $\Theta(n+m)$]

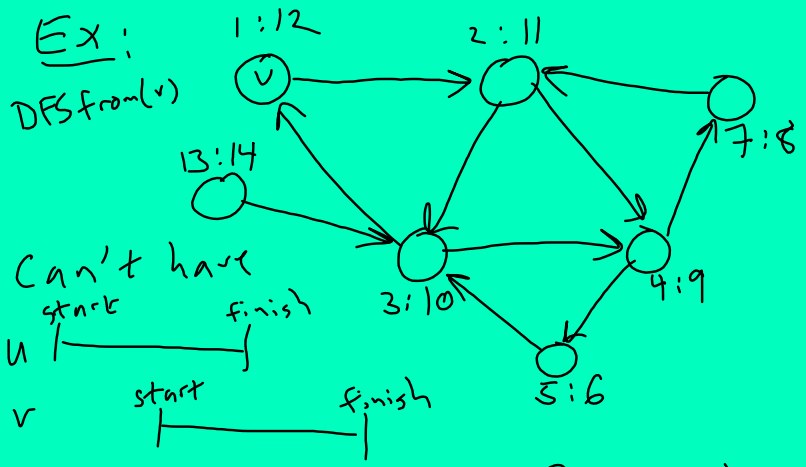
DFS

```
DFS(digraph G)
    For all v ∈ G.V, set
        v.start := v.finish := 0
    (global) timestamp := 0
    For each v ∈ G.V:
        if v.start == 0:
            DFSfrom(v)
```

```
DFSfrom(v)
    timestamp := timestamp + 1
    if v.start == 0:
        v.start := timestamp
        for each w adjacent
            from v:
            DFSfrom(w)
        timestamp := timestamp + 1
        v.finish := timestamp
```

Ex:

DFSfrom(v)



Can't have

[start, finish] intervals

for any two vertices are either disjoint or nested (one inside the other)