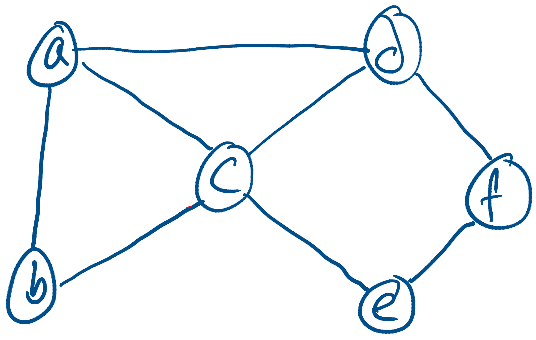
Week 13: Graphs

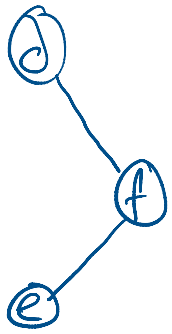
Friday, December 4, 2020 9:50 AM

ADT → nodes
→ vertices



a: b,c,d
b: a,c
c: a,b,d,e
d: a,c,f
e: c,f
f: d,e

	a	b	c	d	e	f
a	0	1	1	1	0	0
b	1	0	1	0	0	0
c	1	1	0	1	1	0
d	1	0	1	0	0	1
e	0	0	1	0	0	1
f	0	0	0	1	1	0



* undirected graph: symmetrical matrix

Types of Graphs:

→ undirected
→ can go both directions on an edge

→ directed
→ each edge has direction

→ weighted / unweighted
→ edges have a "cost" associated with using them

Data Structure Representations

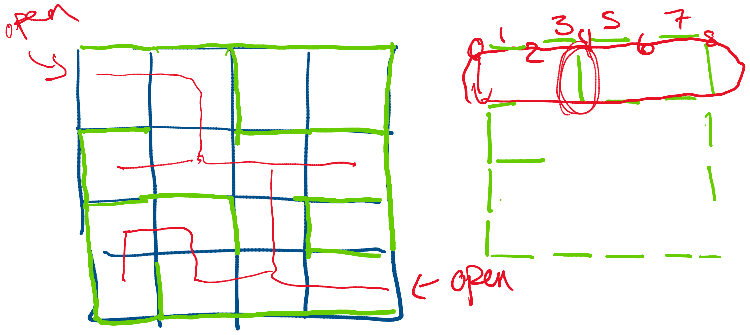
→ adjacency matrix
→ rows/columns for each node
→ true if there is an edge b/n 2 nodes

→ adjacency list
→ table/list of nodes
→ each node has a list of nodes adjacent to it

* hash map
* much more space efficient

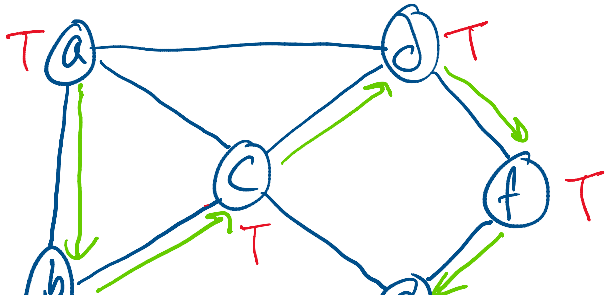
Operations

- neighbors(v):
  - adjacency list: O(deg(v))
  - matrix: O(n)
- check if 2 nodes (u,v) are adjacent
  - list: O(deg(v))
  - matrix: O(1)
    m(u,v)
- add node to graph
  - list: O(1) or O(deg(v))
  - matrix: O(n)
- remove node
  - list: O(deg(v))
  - matrix: O(n)



ropen: is there an opening in the row

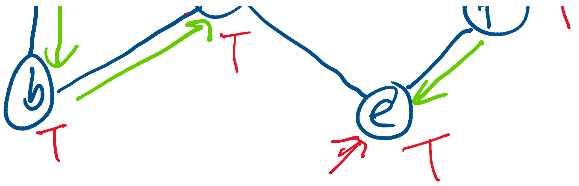
copen: is there an opening in the column?



DFS(G):

for each node v:
  visited[v] ← F *

for each node v: →*
  if !visited[v]: *
    explore(G,v)




```
if !visited[v]:
    explore(G,v)
```

```
explore(G,v):
visited[v] ← T
for each neighbor
    u of v:
    if !visited[u]:
    * explore(G,u)
```

Search methods

Depth-first Search

→ stack
→ starts at a node
  → visits a neighbor
  → then a neighbor of neighbor
  → keeps following a path until it hits a dead end
  → back tracks

order: a, b, c, d, f, e

* usually does not find best path
* will find a solution if it exists

Breadth-first

→ queue
→ starts at beginning node
  → visits all neighbors
  → chooses 1 neighbor & visits all of its neighbors

* level-search