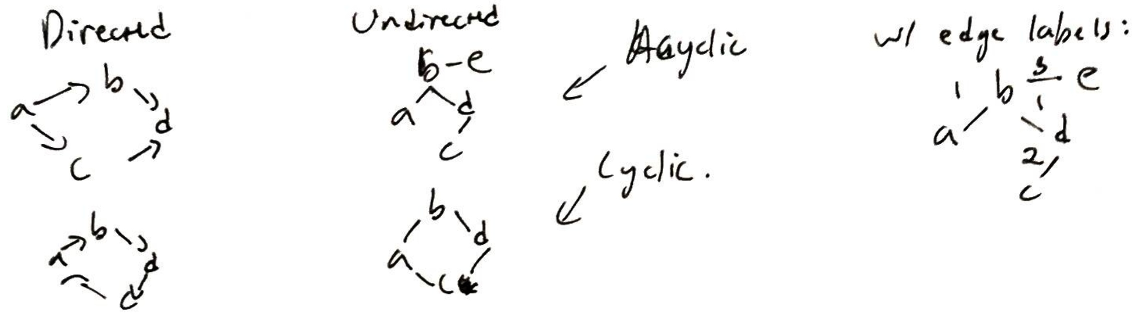
CS lec 27 Graphs

Graphs = generalization of L.L., maps, etc.
↳ is a set of nodes connected pairwise by edges.




edges = pair of vertices.
vertices w/ an edge between are adjacent.
↳ can have labels / weights.

path = sequence of vertices connected by edges.
cycle = path where the first & last vertices are same.
- a graph w/ at least 1 cycle is cyclic.
2 vertices are connected if there is a path between them. If all vertices are connected, the graph is connected.
Degree of vertex = how many edges touch it.

## Graph Representation

Common convention: Number nodes, use the number. To lookup a vertex by label, use a map<Label, Integer>

Api used:

Graph:
- Graph(int V) → creates graph w/ V vertices.
- addEdge(int v, int w) → add an edge v-w
- Iterable adj(int v) → vertices adjacent to v
- int V() → # of vertices
- int E() → # of edges.

Representation #1:

Adjacency matrix.

- place "true"/ 1 wherever there's a connection.

| s\t | 0 | 1 | 2 |
|---|---|---|---|
| 0 | 0 | 1 | 1 |
| 1 | 0 | 0 | 1 |
| 2 | 0 | 0 | 0 |

→ 

For undirected graph: each edge is represented twice in matrix.
simplicity for space.

```
for (int v=0; v<G.V(); v++){
  for (int w: G.adj(v)) {
    print(v+"-"+w);
  }
}
```

→ runtime is $V^2$ in total.
for each = V, outer = V.
$V^2$.

Most Graph Representations:

Adjacency Lists.

- Maintain array of lists indexed by vertex #.
- Most popular Representation

| |
|---|
| 0 → [1, 2] |
| 1 → [2] |
| 2 |

→ 

Runtime of for each = $\Omega(1)$, $O(V)$

How many times is for each run? V times.

Best case: $\Theta(V)$ worst: $\Theta(V^2)$

Runtime overall = $\Theta(V+E)$

E can grow quickly → $V^2$ time

E can grow slowly → V time.

Some runtimes:

| idea | addEdge(s,t) | for(w:adj(v)) | printgraph() | hasEdge(s,t) | Space used |
|---|---|---|---|---|---|
| Adjacency matrix | $\Theta(1)$ | $\Theta(V)$ | $\Theta(V^2)$ | $\Theta(1)$ | $\Theta(V^2)$ |
| ~~Adja~~ list of edges | $\Theta(1)$ | $\Theta(E)$ | $\Theta(E)$ | $\Theta(E)$ | $\Theta(E)$ |
| adjacency list | $\Theta(1)$ | $\Theta(1)$ to $\Theta(V)$ | $\Theta(V+E)$ | $\Theta(\text{degree}(v))$ | $\Theta(E+V)$ |

Adjacency lists are most common because
1) Many algorithms use adj(s)
2) Most graphs are sparse (not many edges in bucket)

Depth First Traversal

~~Prgm~~ is (s connected to t?)

- Mark s.
- Does s == t? if so, return true.
- Check all of s's unmarked neighbors for connectivity to t.

Depth-first → go deep instead of wide.

Common way to do things w/ graphs:

pass graph into another object that records & holds data.

Paths {
   Paths(Graph G, int s) → Does all the work, find all paths from G.
   boolean hasPathTo(int v) → is there a path from s to v?
   Iterable pathTo(int v) → path from s to v if any.
}

* Missed some notes at end of lecture