Graph Algorithms

§ 9.3.2 Dijkstra's

§ 9.5.1 Prim's

shortest path, minimum spanning tree

- code ends up being very similar
  (one line difference)
- this doesn't show up till $\boxed{\text{lab 12}}$
  ↳ Lab 12 uses STL's
  std::map< >
  (differs from text!!!)

- Definitions

$G = (V, E)$

graph G with vertices V &

edges E

- each edge is a pair (v, w)

such that $v, w \in V$

- if the pair is ordered

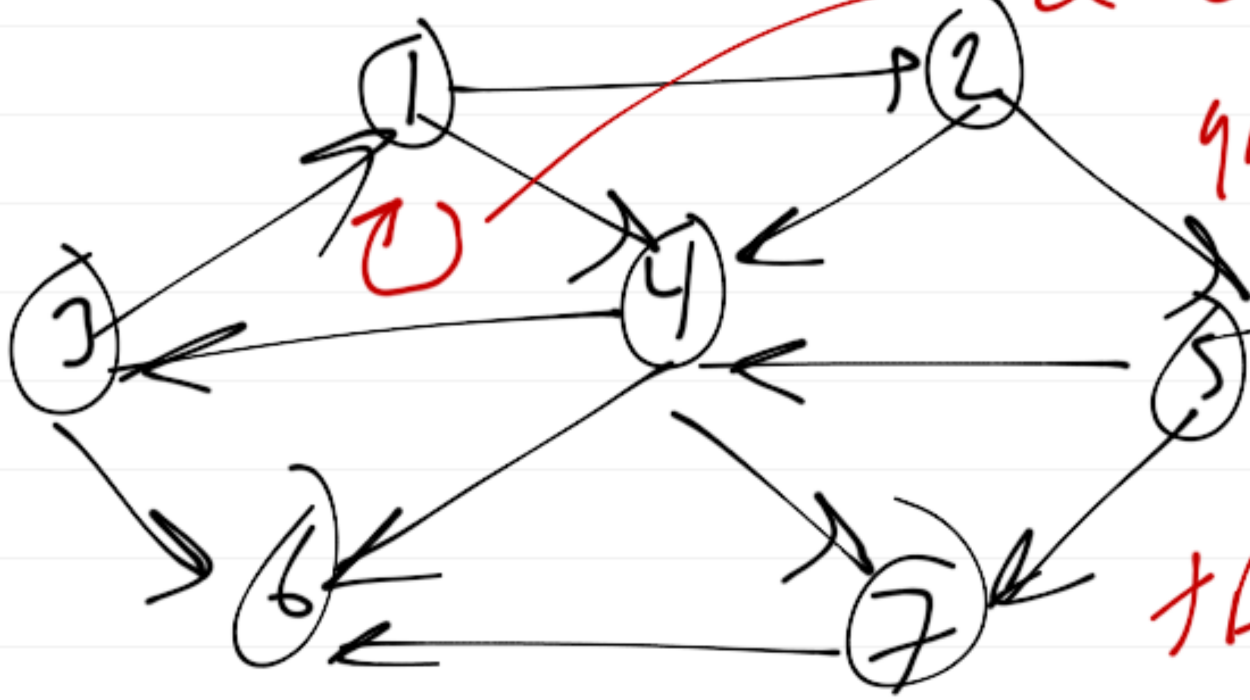
$(v, w) \neq (w, v)$

then graph is <u>directed</u>

- an edge expresses notion of adjacency

i.e., vertex w is adjacent to v iff $(v,w) \in E$

eg., a digraph (directed graph)



a cycle: graphs can have these trees cannot.

- datastructure:
  (representation): matrix

$$
\begin{array}{c|ccccccc}
 & v_1 & v_2 & v_3 & v_4 & v_5 & v_6 & v_7 \\
\hline
v_1 & 0 & 1 & \infty & 1 & \infty & \infty & \infty \\
v_2 & \infty & 0 & \infty & 1 & 1 & \infty & \infty \\
v_3 & 1 & \infty & 0 & \infty & \infty & 1 & \infty \\
v_4 & \infty & \infty & \infty & 0 & \infty & 1 & 1 \\
v_5 & \infty & \infty & \infty & \infty & 0 & \infty & 1 \\
v_6 & \infty & \infty & \infty & \infty & \infty & 0 & \infty \\
v_7 & \infty & \infty & \infty & \infty & \infty & 1 & 0
\end{array}
$$

- edge weight or cost:

  If edge exists, assign unit (1) cost by default (often edges will have cost > 1)

- path: a sequence of vertices $w_1, w_2, \ldots, w_n$ | such that $(w_i, w_{i+1}) \in E$, $1 \leq i \leq n$
  one

e.g., in graph, path from

1 to 7 can be

$1 \to 4 \to 7$

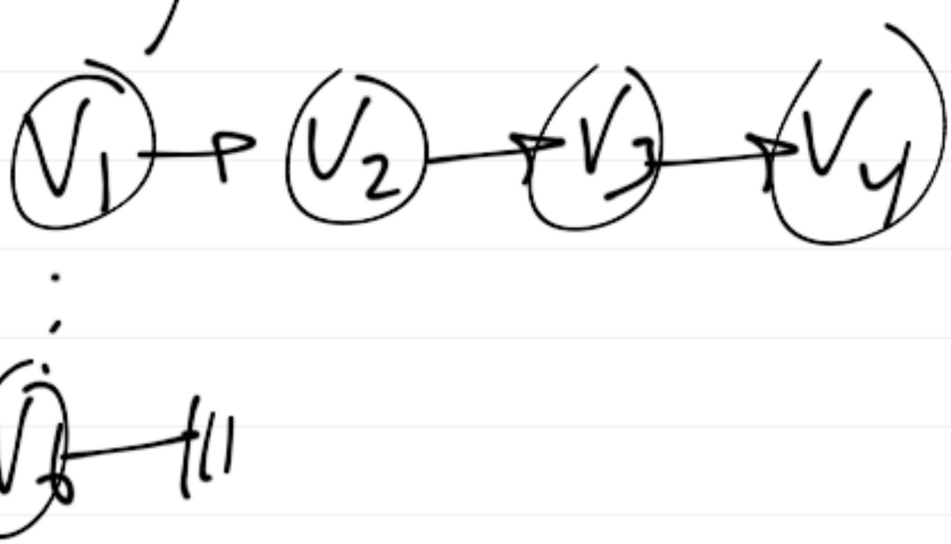
$(v_1, v_4), (v_4, v_7) \in E$

- cycle: path from some vertex w_i s.t. it leads back to w_i: a path of length $\geq 1 \mid w_1 = w_n$

- A directed, acyclic graph (one with no cycles) is known as a DAG
- a <u>complete</u> graph is one where there is an edge between every pair of vertices (matrix is full)

- matrix representation is generally preferred for <u>non-sparse</u> graphs (many. edge connections or complete graph)

- for sparse graphs, node-ptr. representation may be OK



- WARNING: text gives pseudo-code for algorithm in Lab 12 (e.g. Fig. 9.31)

BUT it uses previous idea of DecKey (minheap)
(Cf. Fig. 9.18 where DecKey appears to be absent)

- I will show you how to do Dijkstra's w/out DecKey (std::priority_queue)

- DecKey changes a node's weight (key) while on minheap (not a good idea)
- we use STL's priority_queue which has no DecKey equivalent

- STL solution relies on the use of an associative array where each node's (vertice's) string key is used as index

```
std::map<string, map<string,
                         int>>
              graph;
```

std::string

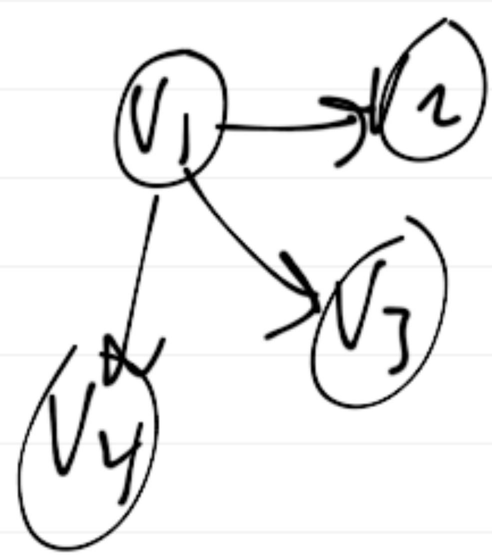
- each graph[string] is an associative array, ex,

std::map<string, map<string, int>> g;

g["v1"]["v2"] = 1;

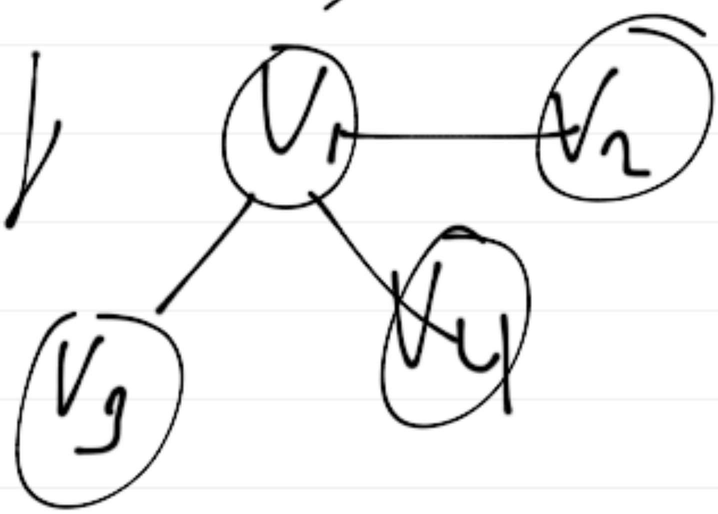
g["v1"]["v3"] = 1;

g["v1"]["v4"] = 1;



(think g["Greenville"]["Myrtle"] = miles)

- no need to store ∞ edges (edges in graph that don't exist)
- if



was an undirected graph,

g["v1"]["v2"] = 1   g["v2"]["v1"] = 1

g["v1"]["v4"] = 1   g["v4"]["v1"] = 1

g["v1"]["v3"] = 1   g["v3"]["v1"] = 1

- in practice, we want to express graph nodes (vertices) as node_t :

```
class node_t {
  public:
      string id;
      int cost;
  // constructor
  // operator=
  bool operator< (const node_t& rhs)
  {
      return cost < rhs.cost;
  }
```

```
bool operator > ( ... )
  { return cost > rhs.cost; }
bool operator == ( ..  rhs)
  { return id == rhs.id;
 }
```

std::string operator==

```
friend ostream & operator<<(..)
  {.. }
```

initialization:

g[node1.id][node2.id] = cost;