oct-20

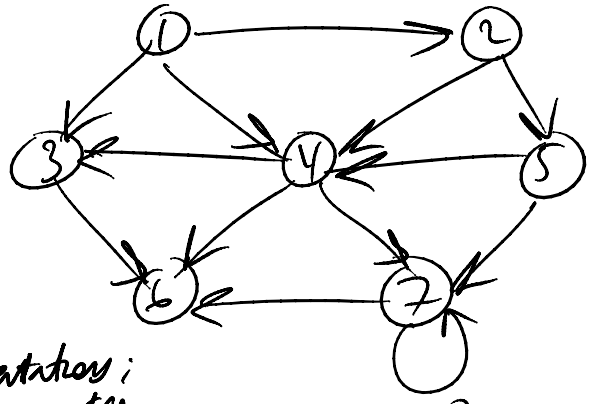
Tuesday, October 20, 2009
2:05 PM

Chp. 9 — Graph Algorithms

- Ass. 5: Dijkstra's & Prim's Algorithms
  with C++ STL maps & priority_queue
  not like the text

- Definitions:
  $G = (V, E)$ Graph G with vertices V & edges E
- Each edge is a pair $(v, w) \mid v, w \in V$
- If the pair is ordered i.e., $(v, w) \neq (w, v)$
  (or I.o.w. we have one of (v, w) or (w, v) but not necessarily both)
  the graph is directed or a digraph
- An edge expresses idea of adjacency,
  i.e. $(v, w) \Rightarrow$ vertex w is adjacent to v
  (vertex w is adjacent to v iff $(v, w) \in E$)
  $\Rightarrow$ this suggests a way to represent (store) the graph info

e.g. a directed graph:



an edge between every pair of vertices
↑
(e.g. Complete)

representations:
- matrix — good for dense graphs
- list of lists — good for sparse graphs

	v_1	v_2	v_3	v_4	v_5	v_6	v_7
v_1	0	1	1	1	∞	∞	∞
v_2	∞	0	∞	1	1	∞	∞
v_3	∞	∞	0	∞	∞	1	∞
v_4	∞	∞	∞	0	∞	1	1
v_5	∞	∞	∞	∞	0	∞	1
v_6	∞	∞	∞	∞	∞	0	∞
v_7	∞	∞	∞	∞	∞	1	0

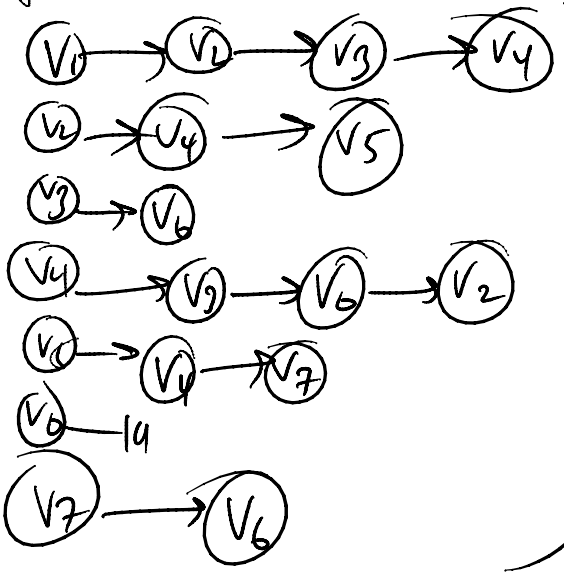
- edge weight (or cost): virtually assigned unit (1) cost if it exists, but more often there is an explicit cost assigned

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

- cycle: is a path from some vertex w_i s.t. it leads back to w_i, i.e. a path of length of at least 1 s.t. $w_1 = w_n$

of length of at least 1 s.t. $v_1 = v_n$

- a directed, acyclic graph 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)
- a matrix is a good representation of complete graphs
- for sparse graphs, list of lists



The STL provides the map<> container, an associative array where the Node's string keys are used as indices

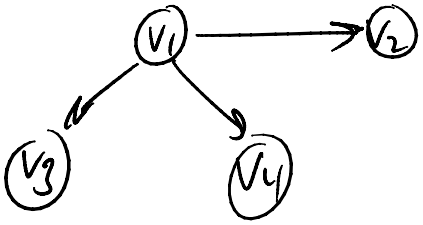
our graph

```
map<string, map<string, int> > g;
```

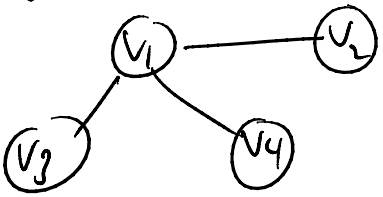
each g[string] entry is an associative array, e.g.

```
g["v1"]["v2"] = 1;
g["v1"]["v3"] = 1;
g["v1"]["v4"] = 1;
```

no need to store the ∞ or inaccessible nodes from v1 in the adjacency list

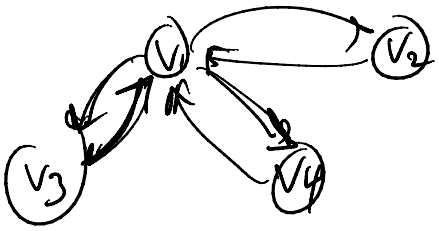


if the graph was undirected,



undirected

, need to record each edge as two directed edges



g["v1"]["v2"] = 1 g["v3"]["v1"] = 1
g["v1"]["v3"] = 1 g["v4"]["v1"] = 1
g["v1"]["v4"] = 1 g["v2"]["v1"] = 1

- in the text, they give pseudo-code for what we want in Asg 5.

But be careful — they rely on previous idea of priority queue (minHeap) & decKey()

They also use a diff't representation of vertices

Tuesday, October 20, 2009
4:04 PM

- in practice, we want to also express/store graph nodes (vertices), e.g.

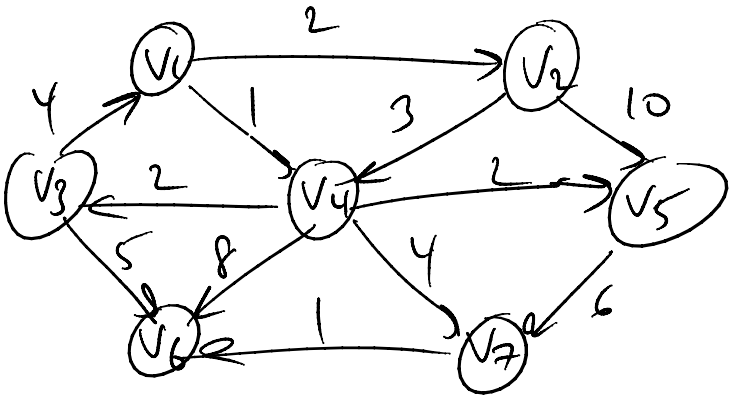
```
class Node {
    public:
        string id;
        int    cost;
    :
    // constructor
    // operator=
    bool operator<(const Node& rhs) const {
            return cost < rhs.cost; }
    bool operator>(...) ...
    bool operator==(...) const {
            return id == rhs.id; }

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

- you'll probably want to add in string for storing id of a "predecessor" — the node that comes before in a path.

Tuesday, October 20, 2009
4:09 PM

§9.3 Shortest Paths

- Dijkstra's algorithm
- each vertex maintains its own cost — this gets updated as the alg proceeds
- Note that this cost is separate from the costs initially set up & maintained untouched in the graph.
- Do ass. by hand (see p. 354)



- to start: put start vertex on priority_que (min-heap)

v	cost	path
v_1		
v_2		
v_3		
v_4		
v_5		
v_6		
v_7		

priority_que
$(v_1, 0)$

- only write down cost once vertex has been visited

hook:

use:

keep track of visited node via hashset

whether cost to node has been registered / saved (in a separate "costs" map)

- Pseudo-code:

while priority-queue is not empty {

- pop off topmost node (because we have a min-heap, node with shortest distance (lowest cost) gets popped off)
- if not yet visited (costs.count(node.id) == 0)
  - not yet visited, not yet in costs map, so add its cost to the costs map
  - for each of its adjacent vertices
    push(Node(node.id, adjacent_node.id, (new node's predecessor)

check if this is

node.cost + adjacent_node.cost)

Step 2

	cost	path
v_1	0	v_1
v_2		
v_3		
v_4		
v_5		
v_6		
v_7		

queue

~~$(v_1, 0)$~~

$(v_2, 0+2)_{v_1}$
$(v_4, 0+1)_{v_1}$ } get put on queue

queue

$(v_4, 1)_{v_1} \leftarrow (v_2, 2)_{v_1}$ (v_1: predecessor — who put me on the queue)

Step 3

	cost	path
v_1	0	v_1
v_2		
v_3		
v_4	1	v_1
v_5		
v_6		
v_7		

queue

~~$(v_4, 1)_{v_1}$~~ $\leftarrow (v_2, 2)_{v_1}$

$(v_3, 1+2)_{v_4}$
$(v_5, 1+2)_{v_4}$
$(v_6, 1+8)_{v_4}$
$(v_7, 1+4)_{v_4}$ } get put on queue

queue

$(v_2, 2)_{v_1} \leftarrow (v_3, 3)_{v_4} \leftarrow (v_5, 3)_{v_4}$
$\leftarrow (v_7, 5)_{v_4} \leftarrow (v_6, 9)_{v_4}$

Step 4

	cost	path
v_1	0	v_1
v_2	2	v_1
v_3		
v_4	1	v_1
v_5		
v_6		
v_7		

queue

~~$(v_2, 2)_{v_1}$~~ $\leftarrow (v_3, 3)_{v_4} \leftarrow (v_5, 3)_{v_4}$
$\leftarrow (v_7, 5)_{v_4} \leftarrow (v_6, 9)_{v_4}$

$(v_5, 2+10)_{v_2}$

v_1	0	v_1
v_2	2	v_1
v_3		
v_4	1	v_1
v_5		
v_6		
v_7		

$(v_7, 5)_{v_4} \leftarrow (v_6, 9)_{v_4}$

$(v_5, 2+1=0)_{v_2}$

Step 5

	cost	path
v_1	0	v_1
v_2	2	v_1
v_3	3	v_4
v_4	1	v_1
v_5		
v_6		
v_7		

$(v_3, 3)_{v_4} \leftarrow (v_5, 3)_{v_4}$

$(v_7, 5)_{v_4} \leftarrow (v_6, 9)_{v_4}$

$(v_5, 12)_{v_2}$

$(v_6, 3+5)_{v_3}$ — put on queue

Step 6

queue

v_1	0	v_1
v_2	2	v_1
v_3	3	v_4
v_4	1	v_1
v_5	3	v_4
v_6		
v_7		

$(v_5, 3)_{v_4} \leftarrow (v_7, 5)_{v_4} \leftarrow (v_6, 8)_{v_3}$

$(v_6, 9)_{v_4} \leftarrow (v_5, 12)_{v_2}$

$(v_7, 3+6)_{v_5}$

Step 7

$(v_7, 5)_{v_4} \leftarrow (v_6, 8)_{v_3} \leftarrow (v_6, 9)_{v_4}$

v_1	0	v_1
v_2	2	v_1

v_3	3	v_4
v_4	1	v_1
v_5	3	v_4
v_6		
v_7	5	v_4

$(v_7, 9)_{v_5} \leftarrow (v_5, 12)_{v_2}$

$(v_6, 5+1)_{v_7}$

---

step 8

v_1	0	v_1
v_2	2	v_1
v_3	3	v_4
v_4	1	v_1
v_5	3	v_4
v_6	6	v_7
v_7	5	v_4

$(v_6, 6)_{v_7} \leftarrow (v_6, 8)_{v_3} \leftarrow (v_7, 9)_{v_5}$

$(v_5, 12)_{v_2}$

have no adj. vert's.

---

step 9 : $(v_6, 8)_{v_3}$, $(v_7, 9)$, $(v_5, 12)$

already seen — " — " —

do nothing

NOTE : USE STL's priority_queue :

priority_queue<Node> pq; // default max heap

override the < to get min heap

priority_queue<Node, vector<Node>, greater<Node>> pq;

check this — google for

c++ STL programming guide