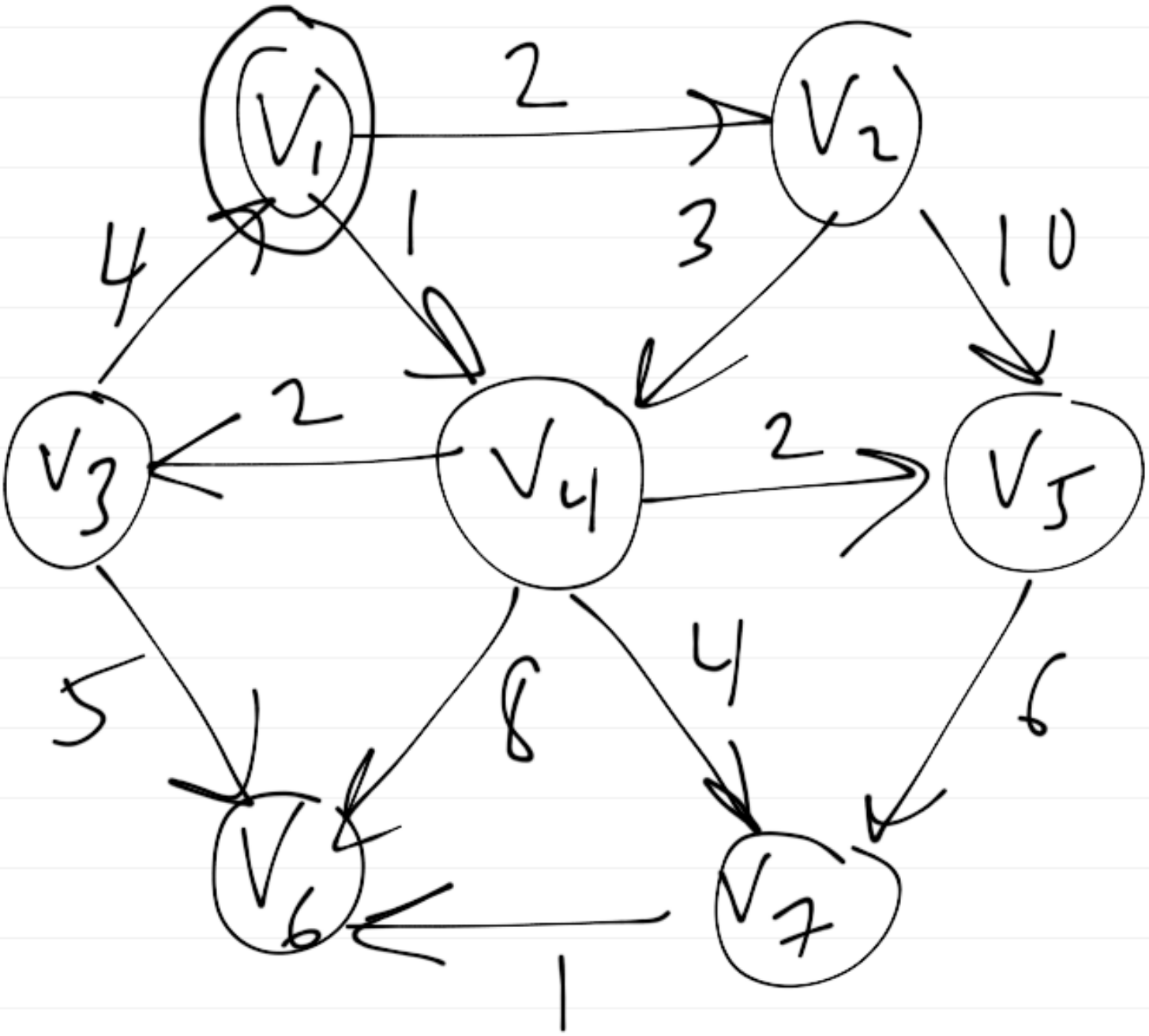


p. 354

- Dijkstra's shortest path:

  - each vertex maintains its own cost — initially these get updated as adj. proceeds

(in our STL implementation a separate map<> is used)

- algorithm is "greedy"
- initially in text, costs are ∞, get adjusted (as vertices are "visited")
- in our case, it's better to write down cost once vertex has been "visited" (cost map stores this info)
- cost map: if vertex is not in map, not yet "visited"

Step 1 — add source vertex to PQ

V	cost	path	priority_queue
v_1			$(v_1, \emptyset)$
v_2			
v_3			
v_4			
v_5			
v_6			
v_7			

source vertex

last

how alg. starts

while priority_queue is not empty {

- pop off topmost vertex (node)
  (least cost — min heap)
- if vertex not yet visited
  (i.e., costs.count(node.id) == 0)
  - add this node's cost to costs map
  - for each of node's adjacent nodes
    - push adj. node onto priority_queue

}

- pushing node onto priority-queue:

pq.push(node_t(node.id, adjacent → node.id, node.cost + adjacent → node.cost))

(predecessor node id)

node_t constructor

take it's out for Prim's

step 2

yet another map< >

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

queue

~~$(v_1, \emptyset)$~~

↓

$(v_2, \emptyset + 2)$

$(v_4, \emptyset + 1)$

queue

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

Step 3

map map

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

queue

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

$(v_3, 1+2)_{v_4}$

$(v_5, 1+2)_{v_4}$

$(v_6, 1+8)_{v_4}$

$(v_7, 1+4)_{v_4}$

queue:

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

v_1

$(v_6, 9)_{v_4}$

Step 4

V	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 ← $(V_3, 3)$ V_4 ← ....

$(V_5, 2+10)$ V_2

$(V_4, 2+3)$ V_2

get added to queue

steps

V	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		

queue

$(V_3, 3)_{V_4} \leftarrow (V_5, 7)_{V_4}$

$(V_7, 5)_{V_4} \leftarrow (V_4, 5)_{V_2}$

$(V_6, 9)_{V_4} \leftarrow (V_5, 12)_{V_2}$

$(V_1, 3+4)_{V_3}$

$(V_6, 3+5)_{V_3}$

step 8

V	cost	path
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

queue

$(V_6, 6)$ V_7

$(V_6, 8)$

$(V_6, 9)$

$(V_7, 9)$

$(V_5, 12)$

no adj. verts

step 9

$(V_2, 8)$ gets popped off

but costs ["V_6"]

already exists (non-zero)

so nothing happens

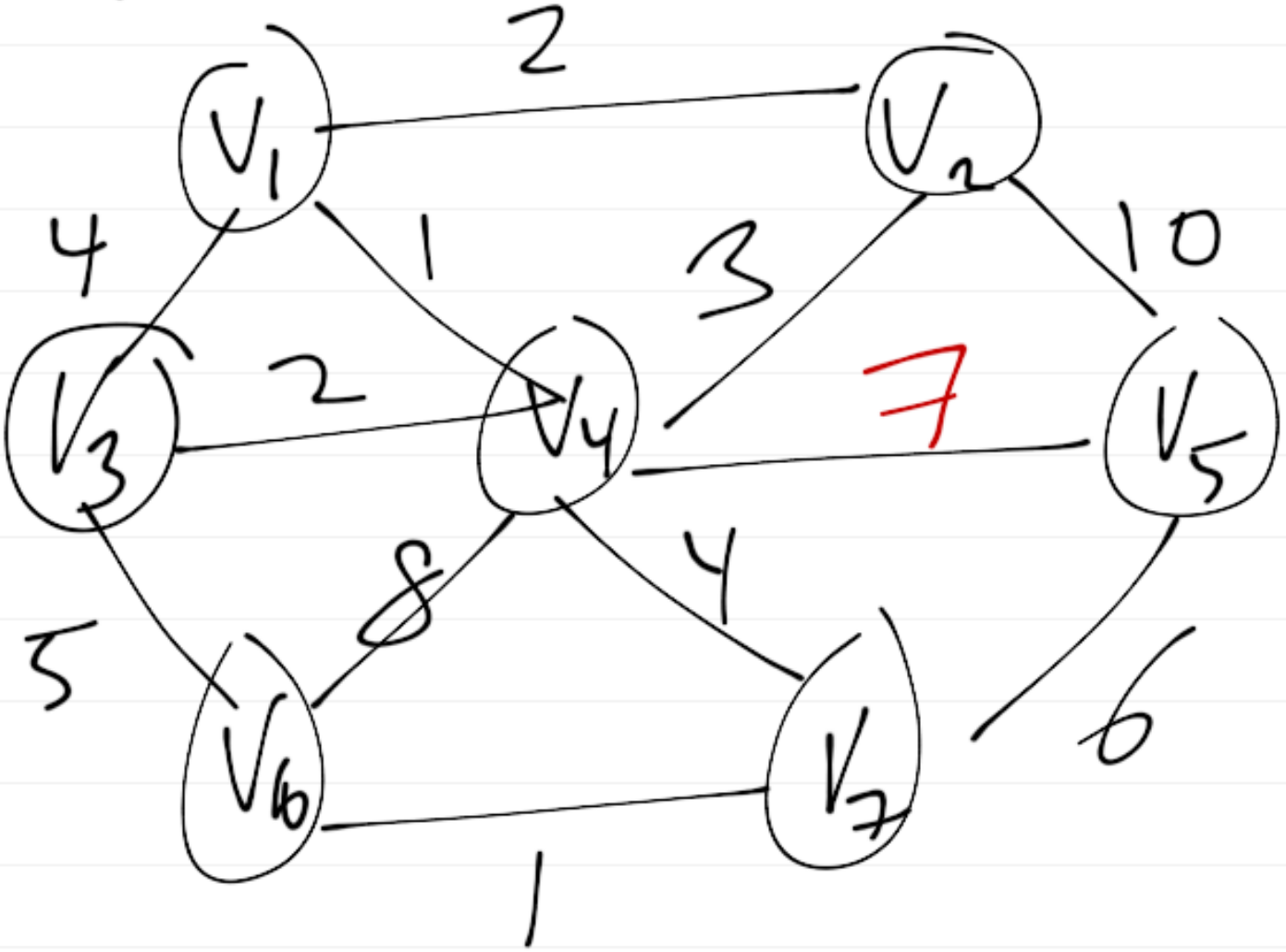
step 10-12

remaining nodes get popped off

Prim's Alg.: MST

Minimum Spanning Tree

(for undirected graphs)



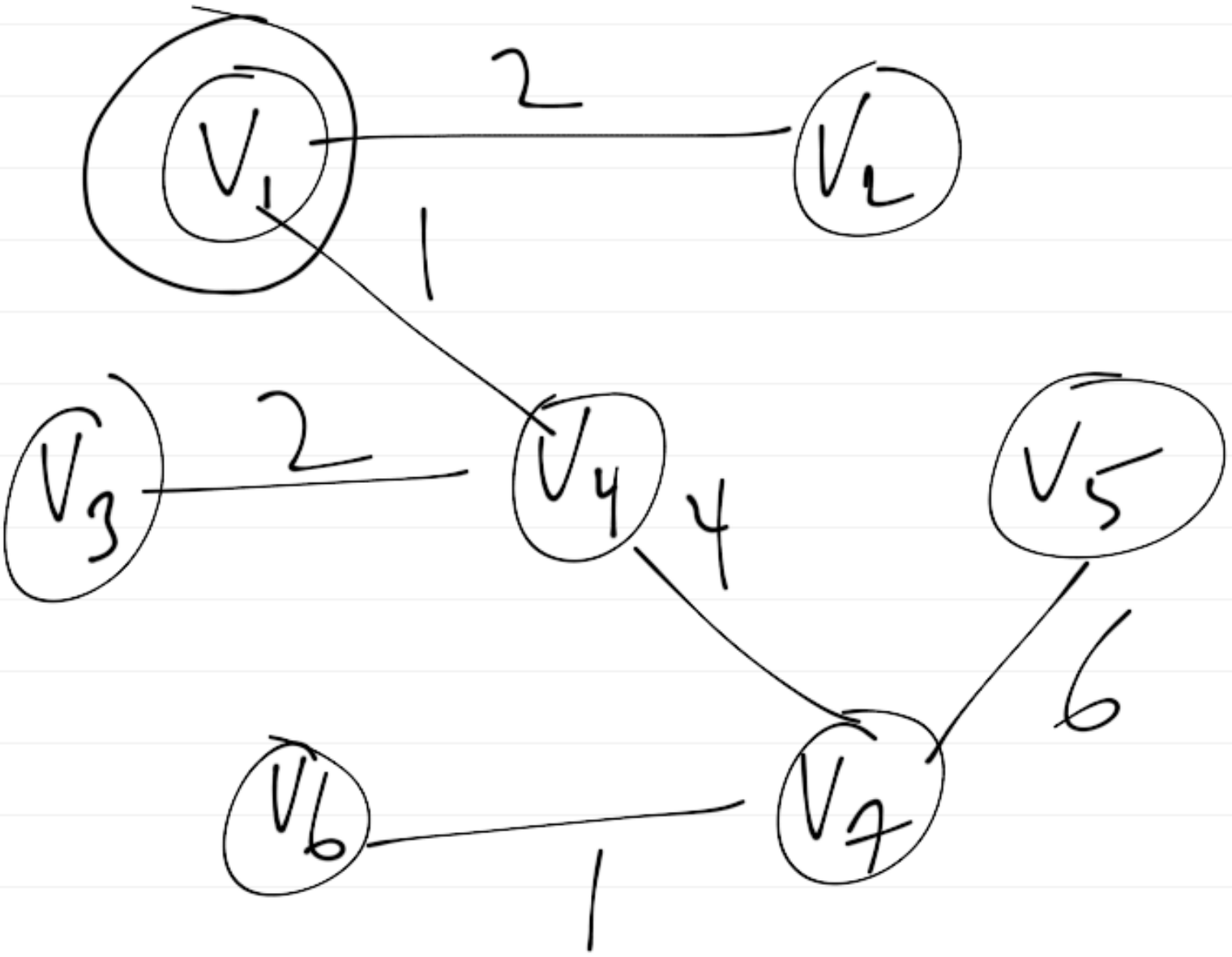
- note undirected graphs:

graph[v1.id][v2.id] = 2
graph[v2.id][v1.id] = 2

⋮

- Prim's is identical to Dijkstra's except for pq statement:

pq.push(node_t(node.id,
adjacent->node.id,
~~cost~~ + adjacent->node.cost));



MST of previous graph
(V_1 root)

– kd-trees (§12.6)

– spatial subdivision tree:

break up 2D (or 3D) space