CS1113
Shortest Paths in Graphs

Lecturer:

Professor Barry O'Sullivan

Office: 2.65, Western Gateway Building

email: *b.osullivan@cs.ucc.ie*

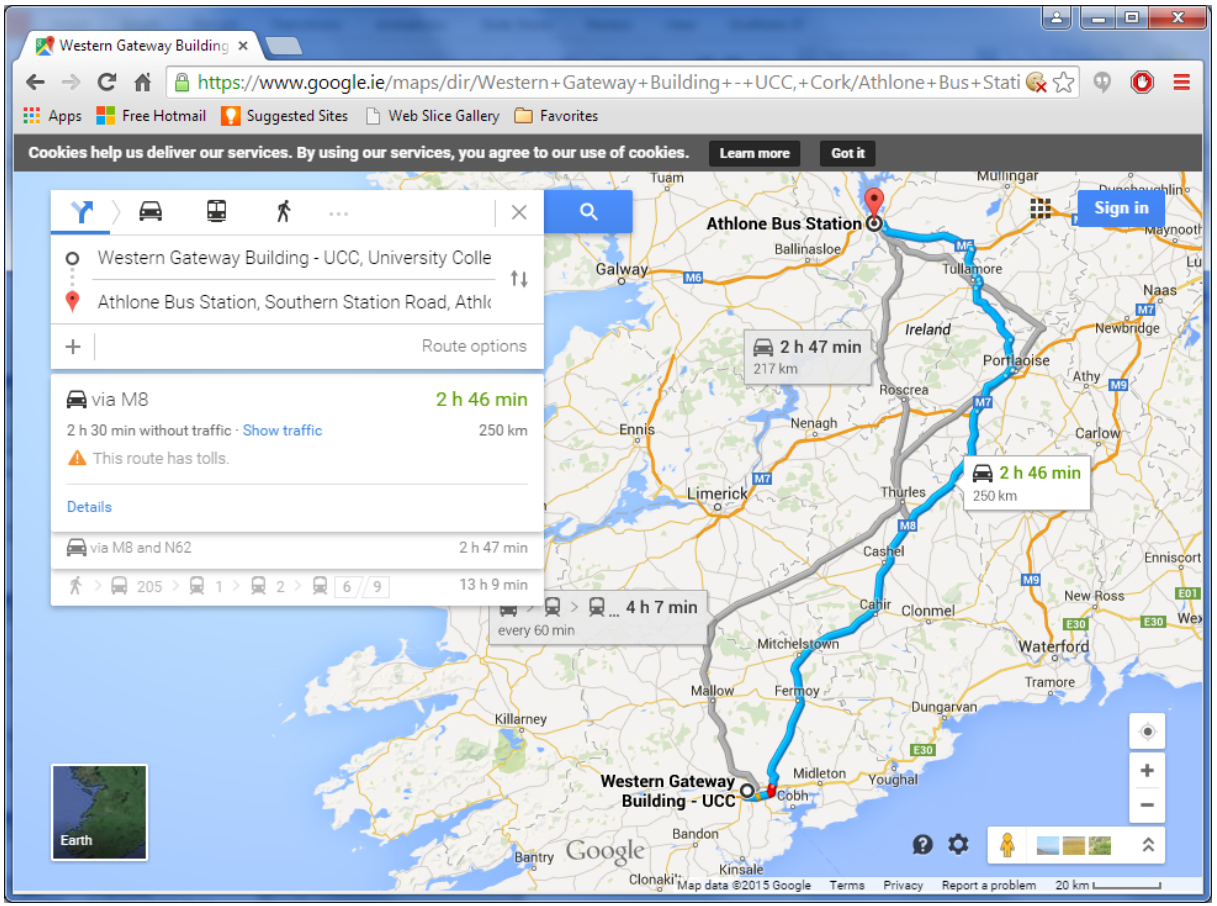
http://osullivan.ucc.ie/teaching/cs1113/

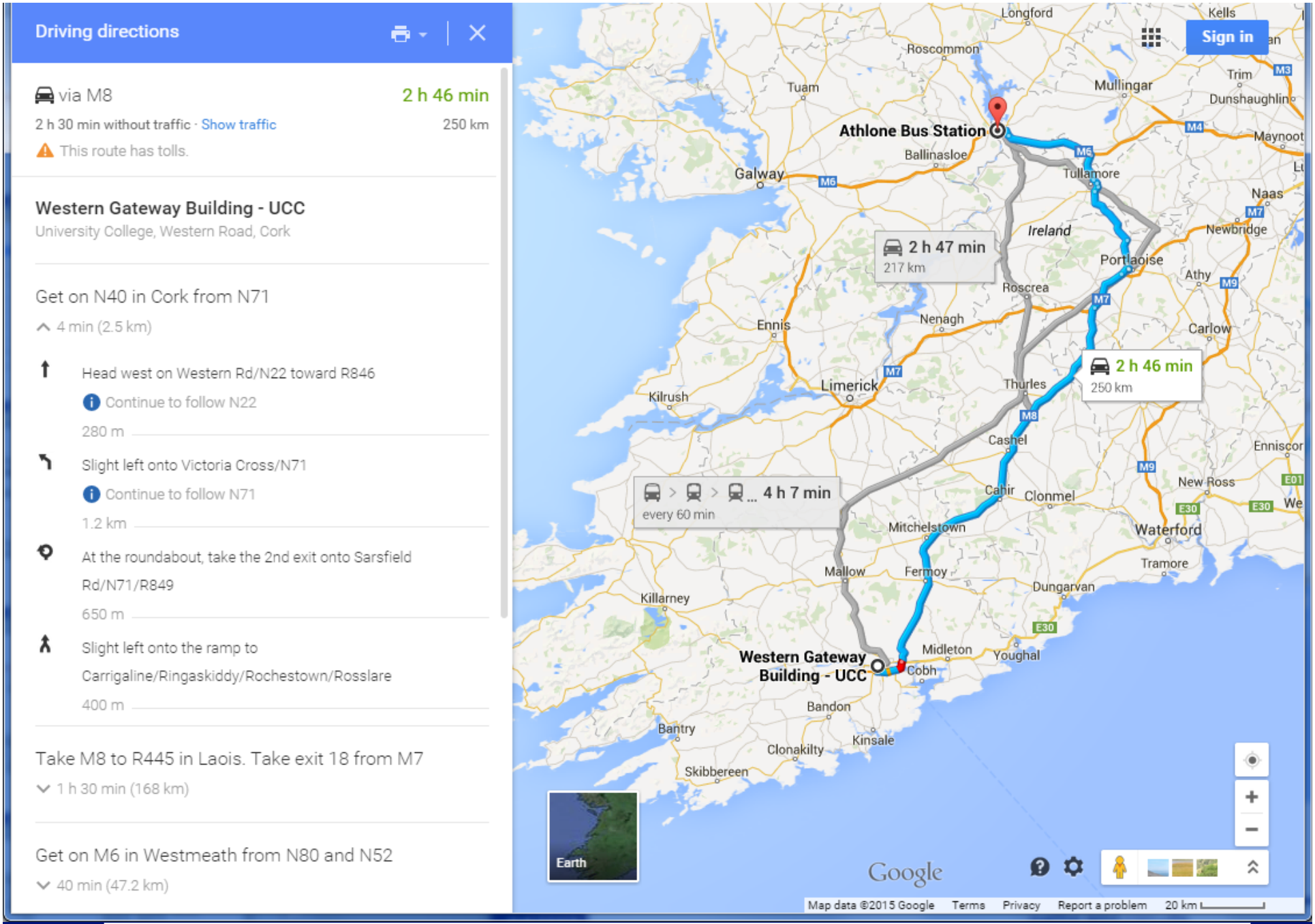
Fastest Paths

Graphs with edge weights

Fastest paths

Dijkstra's Algorithm




Driving directions
via M8
2 h 46 min
2 h 30 min without traffic · Show traffic
250 km
This route has tolls.
Western Gateway Building - UCC
University College, Western Road, Cork
Get on N40 in Cork from N71
4 min (2.5 km)
Head west on Western Rd/N22 toward R846
Continue to follow N22
280 m
Slight left onto Victoria Cross/N71
Continue to follow N71
1.2 km
At the roundabout, take the 2nd exit onto Sarsfield Rd/N71/R849
650 m
Slight left onto the ramp to Carrigaline/Ringaskiddy/Rochestown/Rosslare
400 m
Take M8 to R445 in Laois. Take exit 18 from M7
1 h 30 min (168 km)
Get on M6 in Westmeath from N80 and N52
40 min (47.2 km)
Sign in
Athlone Bus Station
2 h 47 min
217 km
2 h 46 min
250 km
4 h 7 min
every 60 min
Western Gateway Building - UCC
Ireland
Earth
Google
Map data ©2015 Google
Terms
Privacy
Report a problem
20 km

The cost of a path

The algorithm in the previous lecture finds a path with the least number of edges.

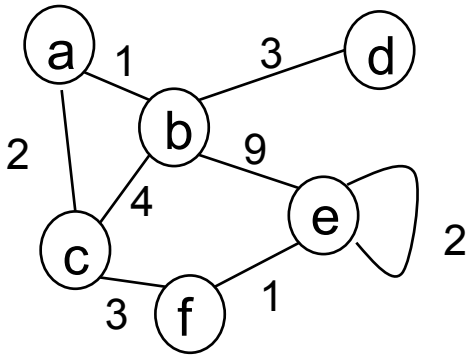
But what if each edge has different properties?

- e.g. distances, travel times, monetary cost, number of enemies who will attack you, energy use, ...

The cost of a path will be the sum of the costs of all the individual edges in the path.

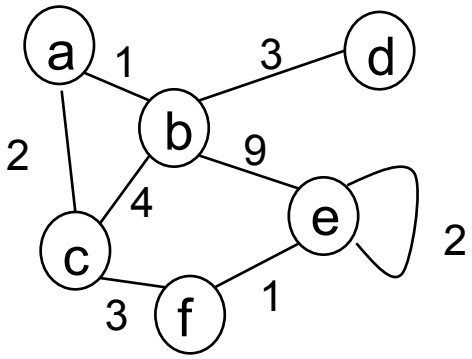
Weighted Graphs

A weighted simple graph G is a pair (V,E), where
V is a set of vertices (representing the objects)
E is a set of weighted edges, where each weighted edge in E is an ordered pair, consisting of an edge (a set of 1 or 2 vertices) followed by a numerical weight.



G = (V,E), where
V = {a,b,c,d,e,f}
E = {({a,b},1), ({a,c},2), ({b,c},4),
({b,d},3), ({b,e},9), ({c,f},3),
({e,e},2), ({e,f},1)}

Path costs

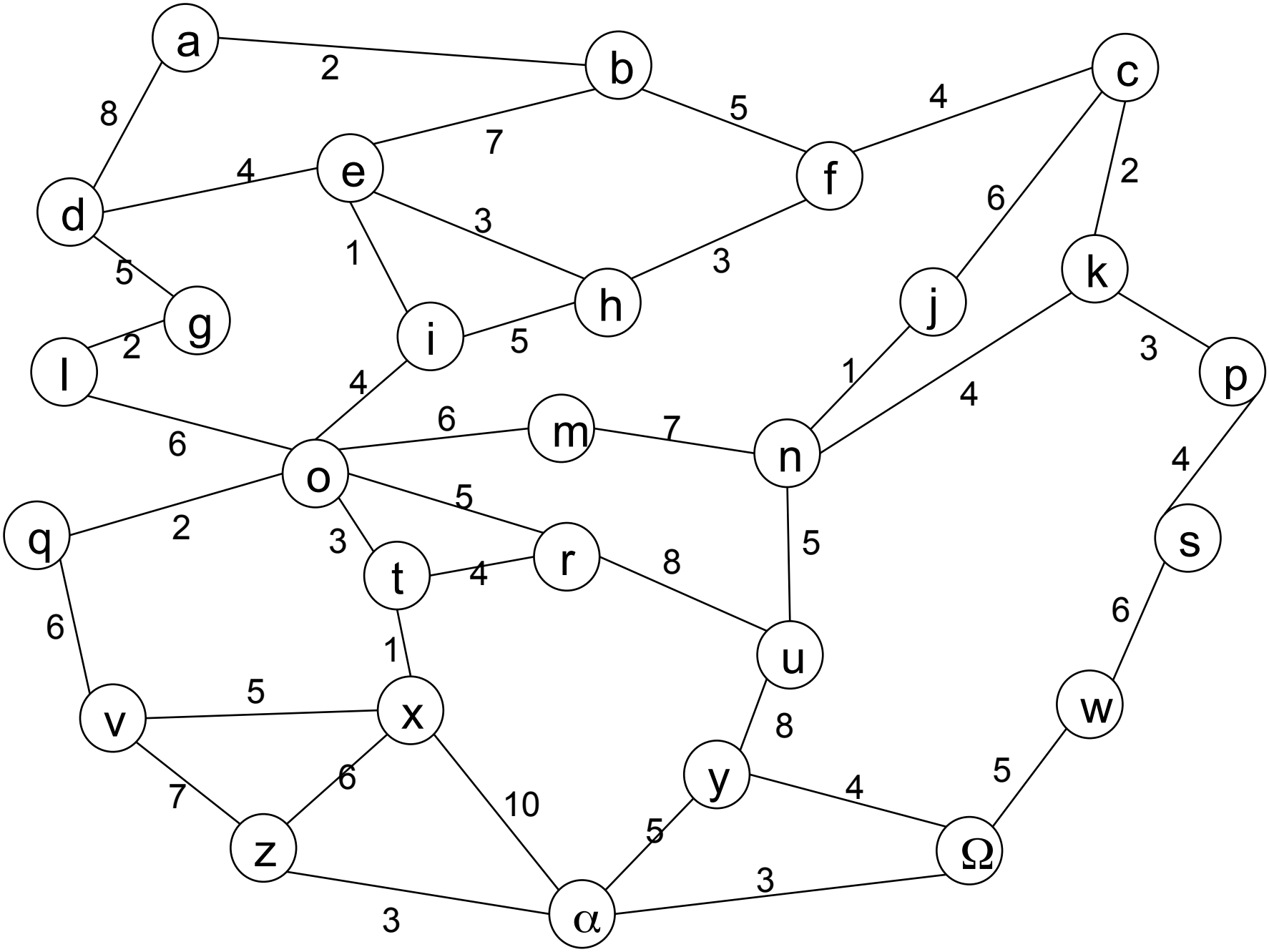


G = (V,E), where
V = {a,b,c,d,e,f}
E = {({a,b},1), ({a,c},2), ({b,c},4),
({b,d},3), ({b,e},9), ({c,f},3),
({e,e},2), ({e,f},1)}

Path <a,b,c,f>:	<(a,b),	(b,c),	(c,f)>	Total cost
Edge costs:	1	4	3	8

What is the cost of <f,e,e,b>?

Can we write an algorithm that computes the cheapest path between any pair of vertices?


a
b
c
d
e
f
g
h
i
j
k
l
m
n
o
p
q
r
s
t
u
v
w
x
y
z
Ω
α
2
8
7
4
5
4
6
2
3
1
3
5
2
5
1
3
4
6
7
4
6
2
5
3
4
8
5
6
1
5
8
6
7
6
10
5
4
5
3
3

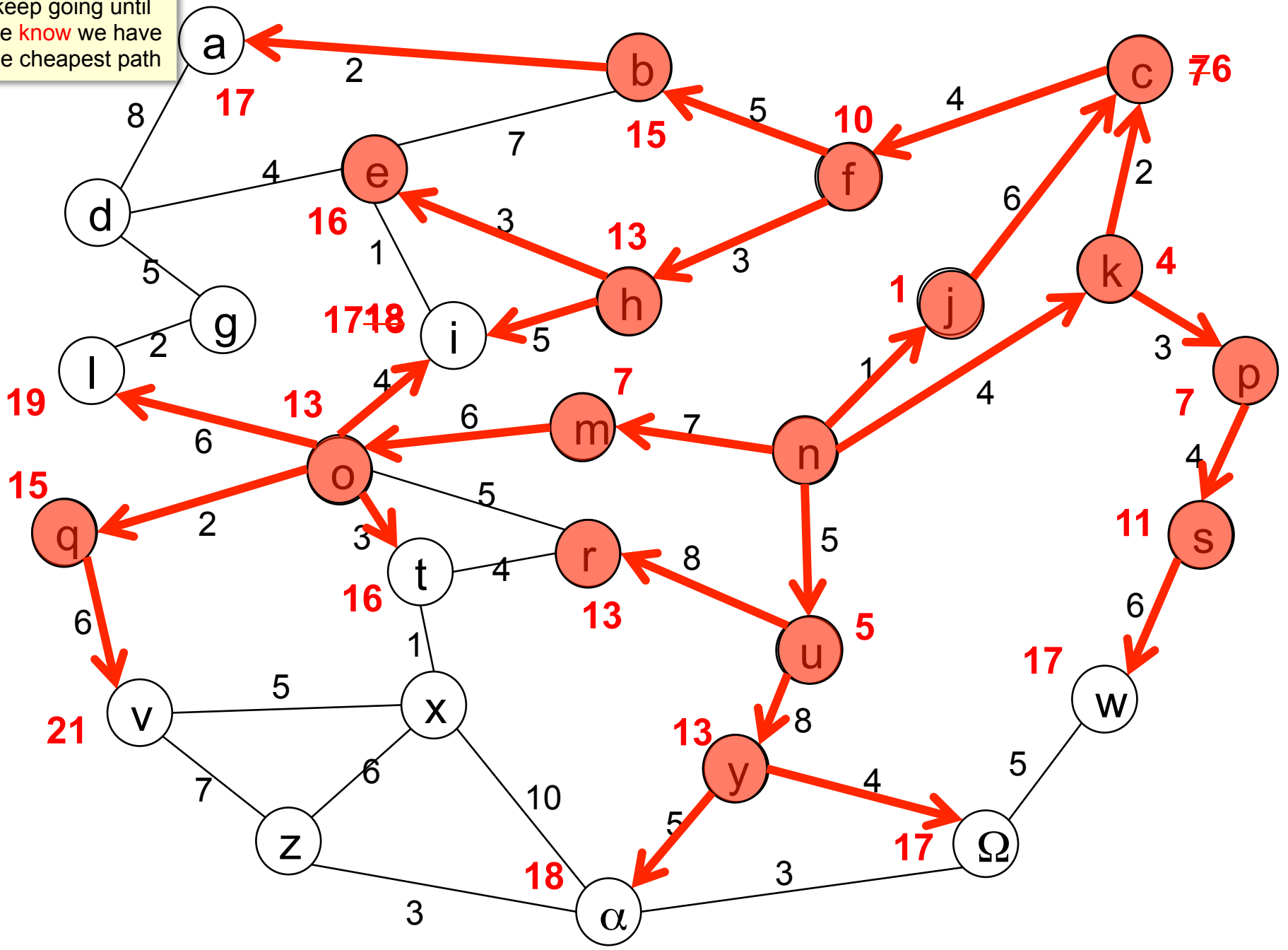
G = (V,E), where

V= {a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s,t,u,v,w,x,y,z,α,Ω}

E = { ({a,b},2),({a,d},8),({b,e},7),({b,f},5),({c,f},5),({c,k},5),
({d,e},4),({d,g},5),({e,h},3),({e,i},1),({f,h},3),
({g,l},2),({h,i},5),({i,o},4),({j,n},1),({k,n},4),({k,p},3),
({l,o},6),({m,n},7),({m,o},6),({n,u},5),({o,q},2),
({o,r},5),({o,t},3),({p,s},4),({q,v},6),({r,t},4),({r,u},8),
({s,w},6),({t,x},1),({u,y},8),({v,x},5),({v,z},7),
({w,Ω},5),({x,z},6),({x,α},10),({y,α},5),({y,Ω},4),
({z,α},3),({α,Ω},3)}

Basic idea: grow all paths out from the start one edge at a time, always expanding from the vertex with the lowest cost so far. When two paths join up at the same vertex, only keep the cheaper one. When we try to expand from the target vertex, we know we must have found the cheapest path.

keep going until we know we have the cheapest path

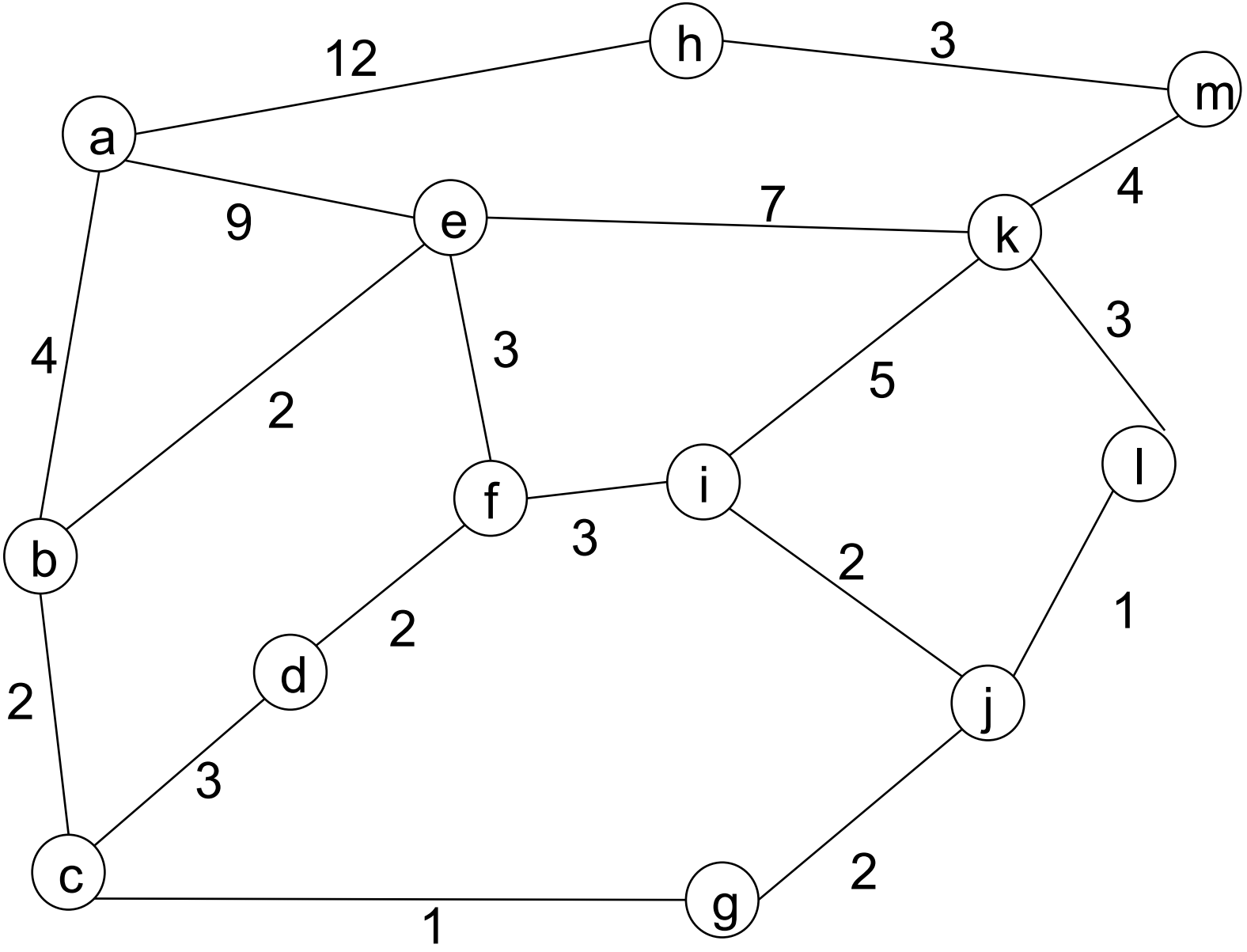


Finding the cheapest path (informal)

Start at the start vertex
Record that it is the start and costs 0 to get there
For each adjacent vertex, record the cost of getting there from the current vertex, and record how we got there
Mark the current vertex as being "done"

From all the vertices that are not done but that have some cost recorded, find the one with lowest cost, and make that the current vertex
If it is the target, we are finished and can report success; if not, we will try expanding from that vertex
For each adjacent vertex to the current one, if it is not "done", compute the cost of extending the path, and if it is a better cost or a new cost, update our records
Mark the current vertex as done
And repeat ...

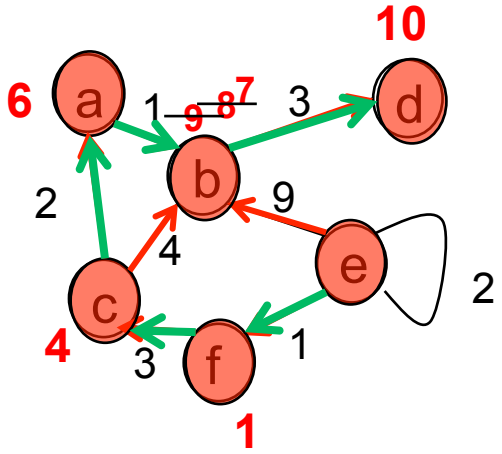
If we run out of vertices, stop and report failure


h
m
a
e
k
l
f
i
b
d
j
c
g
12
3
9
7
4
4
2
3
5
3
3
2
1
2
2
3
1

Recording the cost and the path

We use an array with 3 cells for each vertex in the graph.
The first cell says whether we have checked that vertex.
The second cell contains our current cost of getting there.
The third cells will tell us the previous vertex on that path.

start: e
end: d



vertex:	a	b	c	d	e	f
done?	1	1	1		1	1
cost:	6	~~9~~ ~~8~~ 7	4	10	0	1
previous:	c	~~c~~ ~~c~~ a	f	b	0	e

Dijkstra's algorithm for finding the cheapest path

```
Algorithm: cheapestpath1 (from Dijkstra)
Input: a weighted graph G=(V,E), where V ={1,2,...,n}
Input: a vertex x from V, the start
Input: a vertex y from V, the end
Output: an table representation of the path, or null if none

1.  L := a table with 3 rows for each of n vertices, all null
2.  v := x
3.  L[v][2] := 0                              //the cost of getting to v
4.  L[v][3] := 0                    //the previous vertex in the path to v
5.  while v is not null
6.     for each vertex j adjacent to v          //expand paths from v
7.        if L[j][1] =/= 1                          //if j not done
8.        then cost := L[v][2] + weight of edge {v,j}  //compute cost
9.             if L[j][2]== null OR cost < L[j][2]       //if better
10.            then L[j][2] := cost                 //update j's cost
11.                 L[j][3] := v                  //say how we got here
12.    L[v][1] := 1                                  //mark v as done
13.    v := vertex with smallest L[v][2] and with L[v][1]== null
            or null if there isn't one          //find cheapest vertex
14.    if v == y, return L     //stop - we've found the cheapest path
15. return null                 //we didn't reach the target by any path
```

	a	b	c	d	e	f	g	h	i	j	k	l	m
a		4			9			12					
b	4		2		2								
c		2		3			1						
d			3			2							
e	9	2				3					7		
f				2	3				3				
g			1							2			
h	12												3
i						3				2	5		
j							2		2			1	
k					7				5			3	4
l										1	3		
m								3			4		

	a	b	c	d	e	f	g	h	i	j	k	l	m
?													
cost													
path													

Next lecture ...

Representing Graphs
Connected Graphs
Circuits