Least-Cost Paths

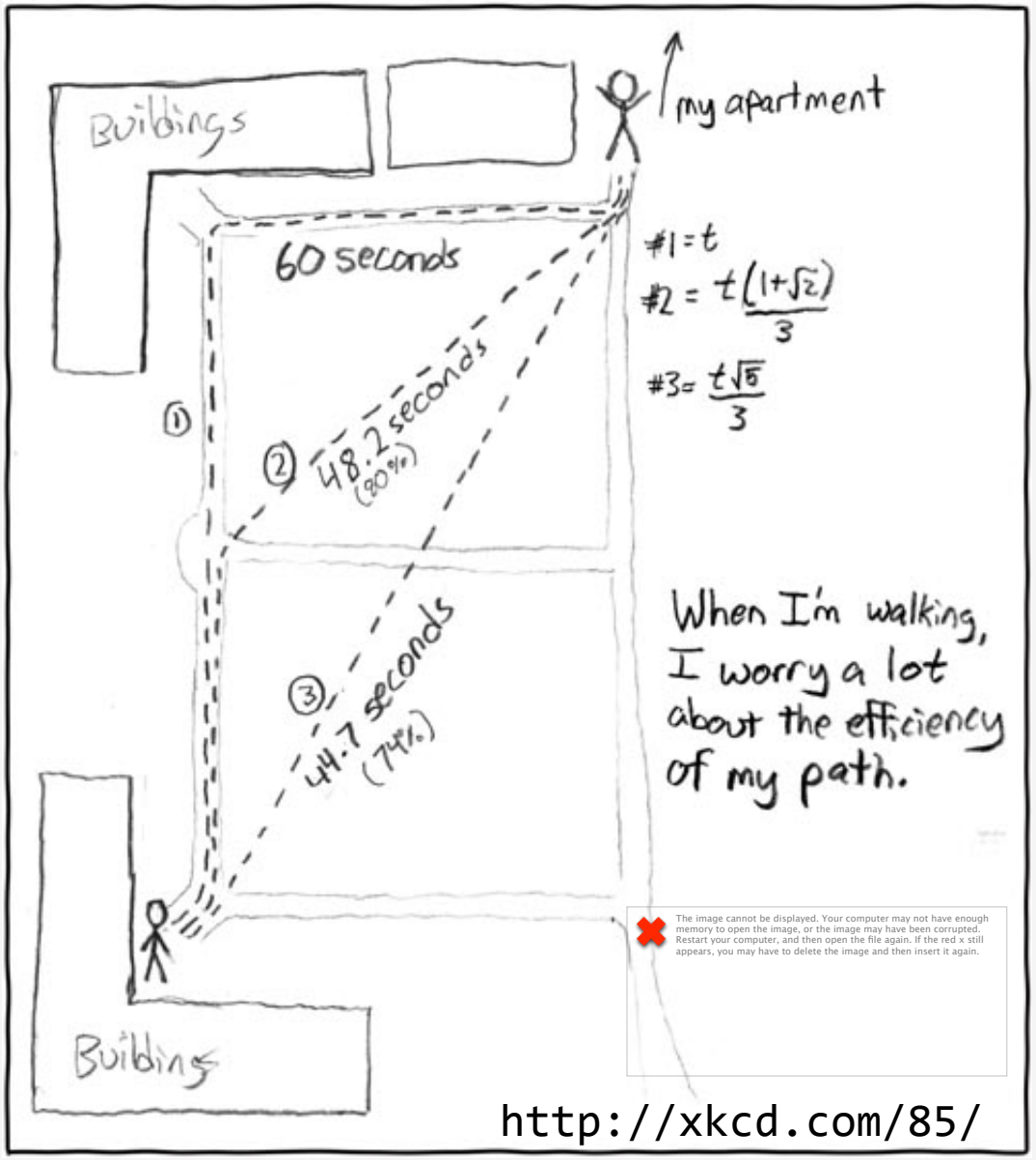
COMP 2210 – Dr. Hendrix

AUBURN

UNIVERSITY

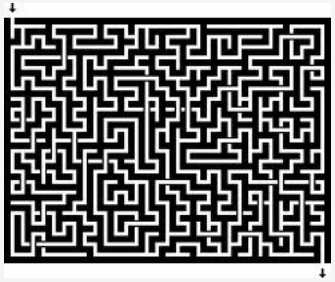
SAMUEL GINN
COLLEGE OF ENGINEERING

Efficient paths



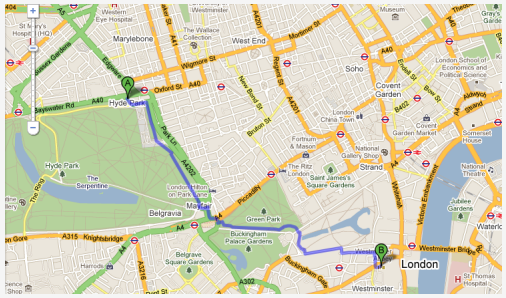
Shortest Paths

Minimum number of edges

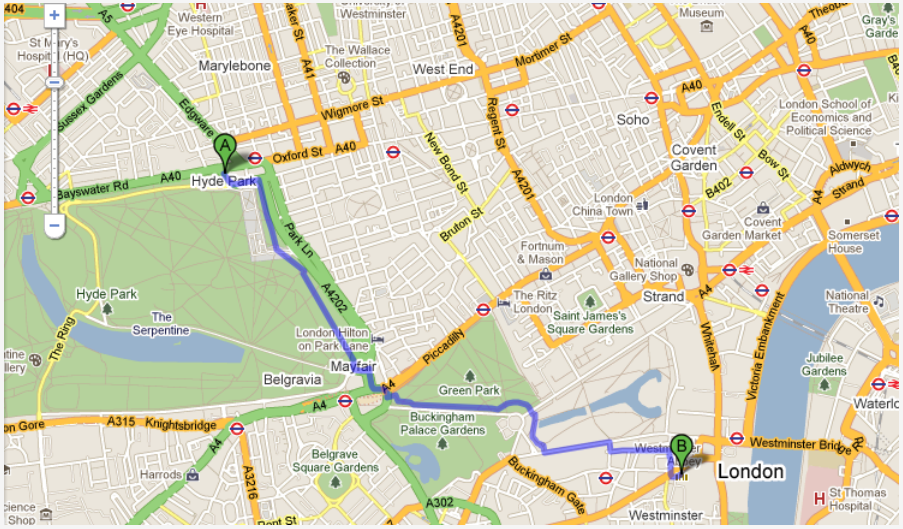


Least-Cost Paths

Minimum cumulative cost of edges

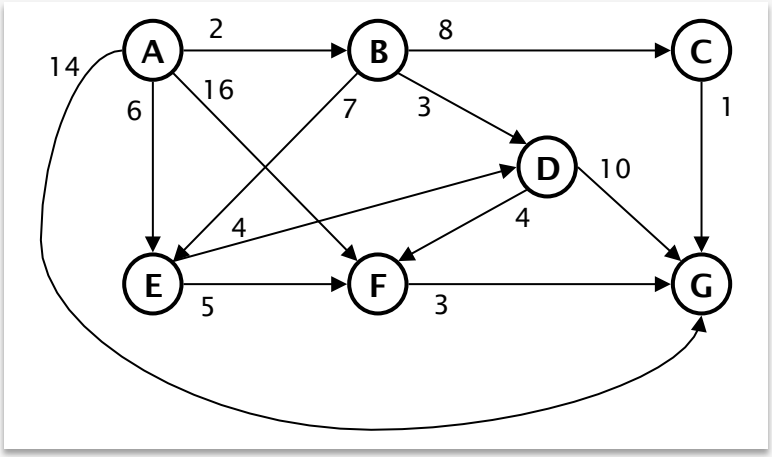


Least-cost path problems



What's the quickest way to walk from Hyde Park to Westminster Abbey?

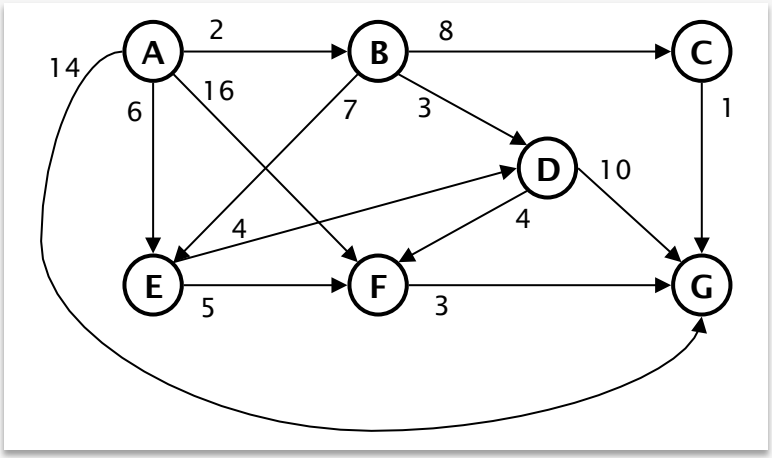
A **least-cost path** in a *weighted, directed graph with non-negative edge weights* is a path from vertex A to vertex B such that the cumulative cost of the edge weights is at least as small as the cumulative cost of any other path from A to B.



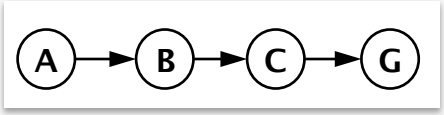
LCP A:D A → B → D Cost = 5

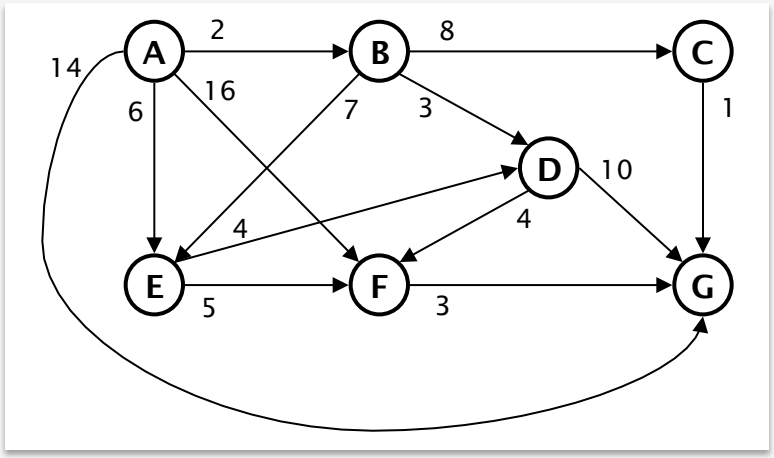
LCP A:G A → B → C → G Cost = 11

Variations on the least-cost path problem

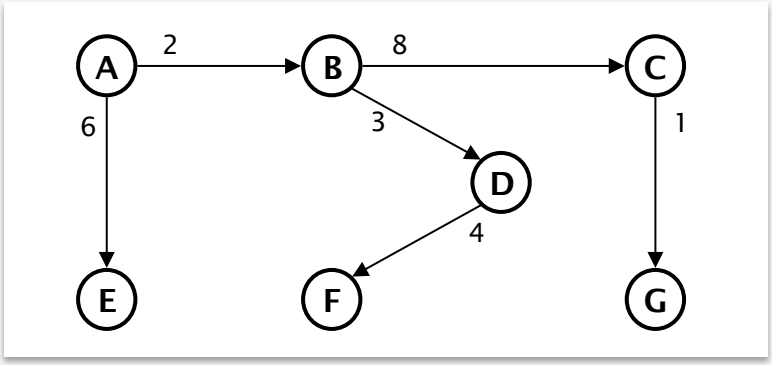


Single source, single destination





Single source, all destinations

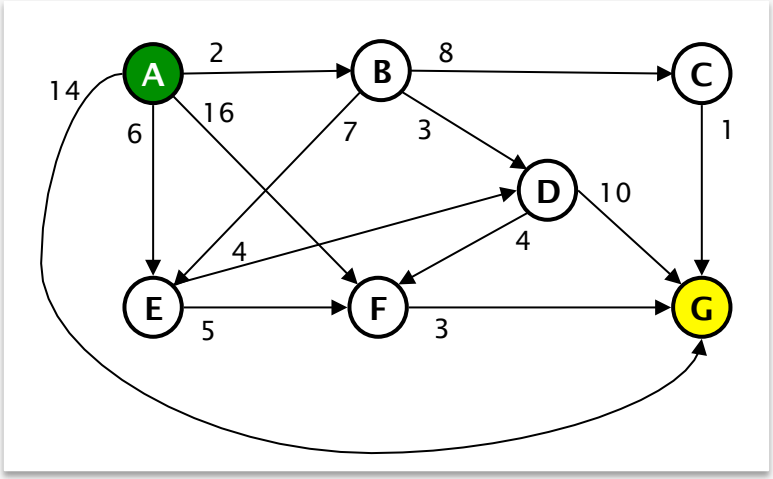


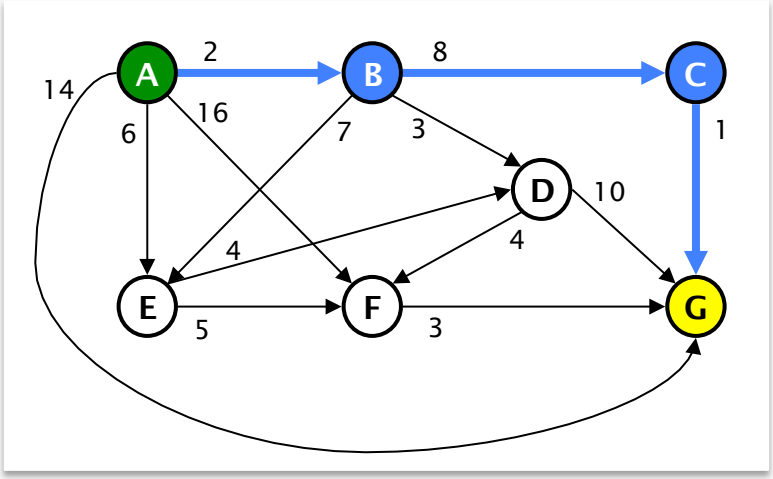
All pairs LCP …

How to construct a least-cost path?

Brute force

Always fun, but rarely practical.





Apply DFS with the greedy heuristic …

At each node, choose the cheapest edge to a new node..

Continue this process, backtracking as needed, until destination is reached.

Our greedy strategy's result:

A → B → D → F → G

2 3 4 3 **12**

The real LCP:

A → B → C → G

2 8 1 **11**

Dijkstra's Algorithm

Developed by Edsger Dijkstra in 1956.

Solves the **single source, all destinations** LCP problem for a directed graph with **non-negative weights**.

Applies the **greedy heuristic** and **relaxation**.

Greedy choice: Rather than selecting the cheapest neighbor on some path, this algorithm iteratively selects the *cheapest neighbor to the source* and thereby discovers a new LCP.

Relaxation: Iteratively refine estimates of the LCP from the source to all other vertices using adjacency and known LCPs.

Besides making significant contributions to computing, Dijkstra also made good quotes …

"Do only what you can do."

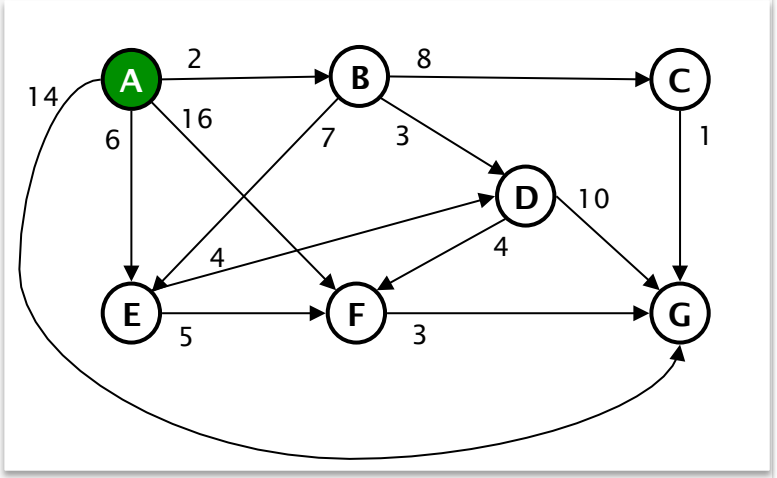
"In their capacity as a tool, computers will be but a ripple on the surface of our culture. In their capacity as an intellectual challenge, they are without precedent in the cultural history of mankind."

"It is practically impossible to teach good programming to students that have had prior exposure to BASIC: as potential programmers they are mentally mutilated beyond hope of regeneration."

"Object-oriented programming is an exceptionally bad idea which could have only originated in California."

Dijkstra's Algorithm

Repeatedly refine estimates of the LCP from the source node to all other nodes, using only adjacency and known LCPs.



Step 0 Estimate the min cost from the source node to all nodes in the graph.

Step 1 Select the node *i* with the current minimum cost estimate.

This node is a destination on a LCP from the source.

Step 2 Consider all one-edge extensions to this new LCP and update cost estimates.

Step 3 If all nodes have been selected, terminate; else go to Step 1.

cost

A	B	C	D	E	F	G
0	2	~~*~~ 10	~~*~~ 5	6	16	14

i	*j*	*cost[j] >? cost[i] + (i,j)*				
B	C	*	>	2	+	8
B	D	*	>	2	+	3
B	E	6	<	2	+	7

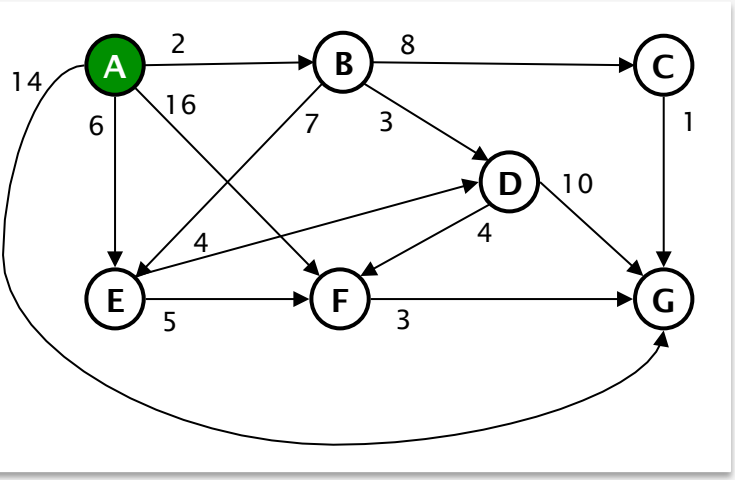
Step 0

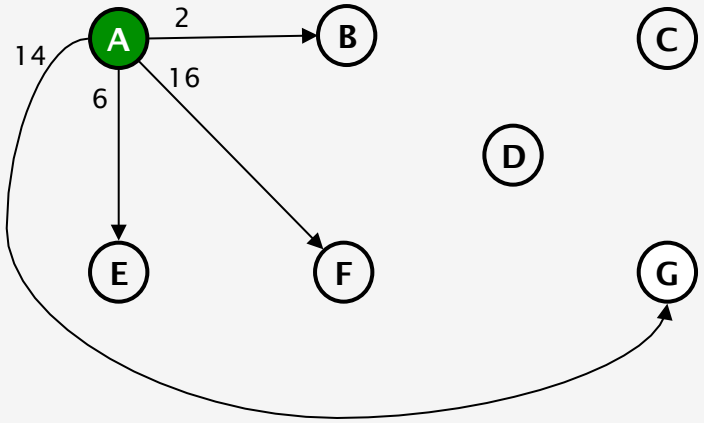
Estimate the min cost from the source node to all nodes in the graph.

using only adjacency and known LCPs.

B, E, F, G

A::A





cost

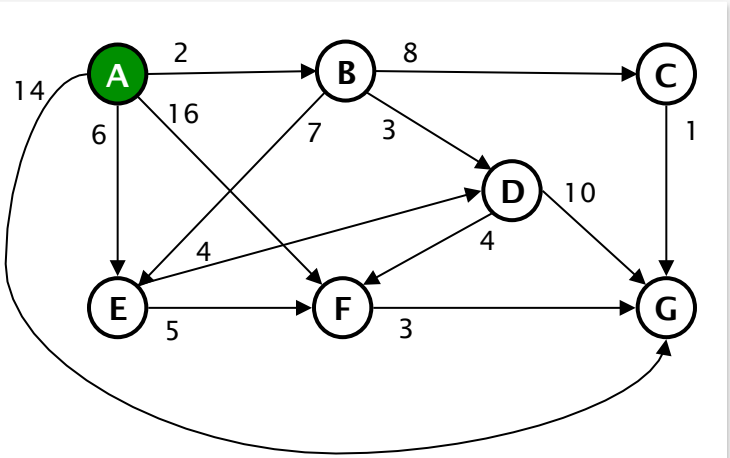
A	B	C	D	E	F	G
0	2	*	*	6	16	14

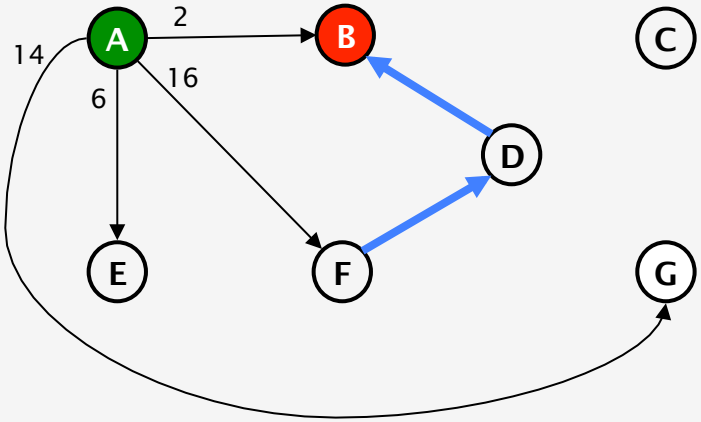
As the algorithm runs, this basic constraint will be "relaxed" by discovering new LCPs and, thus, new adjacent nodes.

Step 1

Select the node *i* with the current minimum cost estimate.

This node is a destination on a LCP from the source.





No negative weights!

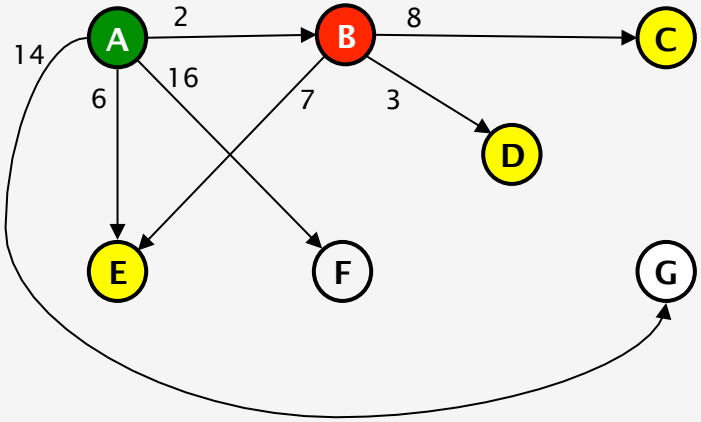
cost

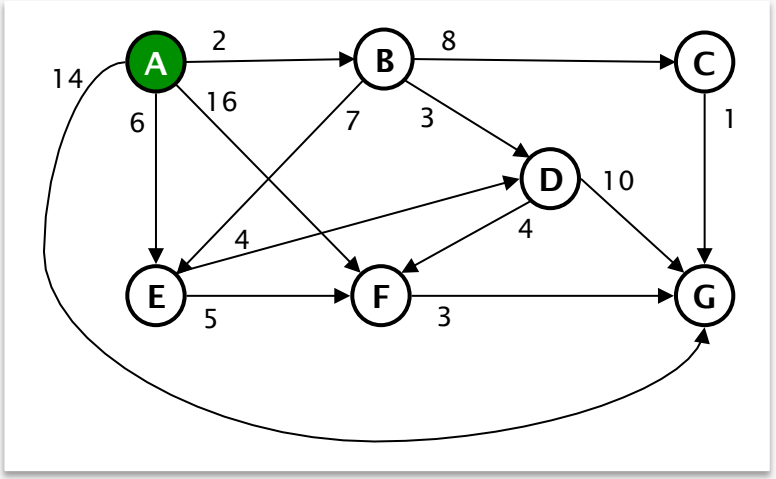
A	B	C	D	E	F	G
0	2	*	*	6	16	14

Known LCPs:
A::A = 0
A::B = 2

Step 2

Consider all one-edge extensions to this new LCP and update cost estimates.





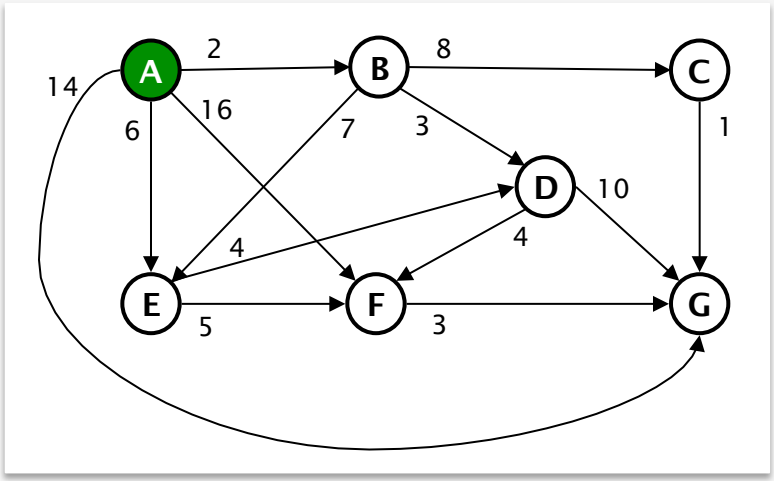
cost

A	B	C	D	E	F	G
0	2	~~*~~ 10	~~*~~ 5	6	16	14

Cost estimates are being repeatedly improved by using new LCPs to potentially decrease the cost to adjacent nodes.

i	*j*	*cost[j] >? cost[i] + (i,j)*				
B	C	*	>	2	+	8
B	D	*	>	2	+	3
B	E	6	<	2	+	7

Example



cost

A	B	C	D	E	F	G
0	2	*	*	6	16	14
		10	5		9	12
						11

path

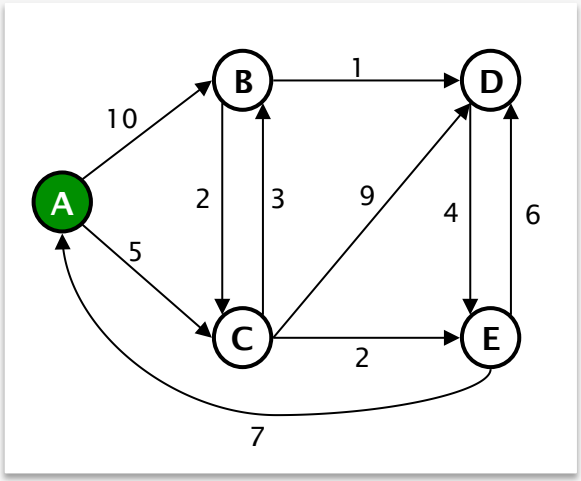
A	B	C	D	E	F	G
A	A	*	*	A	A	A
		B	B		D	F
						C

M	N	cost[N]	>?	cost[M]	+	a[M,N]
B	C	*	>	2	+	8
	D	*	>	2	+	3
	E	6	<	2	+	7
D	F	16	>	5	+	4
	G	14	<	5	+	10
E	F	9	<	6	+	5
F	G	14	>	9	+	3
C	G	12	>	10	+	1

known least-cost paths

A::A	0	A
A::B	2	A → B
A::D	5	A → B → D
A::E	6	A → E
A::F	9	A → B → D → F
A::C	10	A → B → C
A::G	11	A → B → C → G

Example



Initial estimates:

cost

A	B	C	D	E
0	10	5	∞	∞

path

A	B	C	D	E
A	A	A	•	•

Final values:

cost

A	B	C	D	E
0	8	5	9	7

path

A	B	C	D	E
A	C	A	B	C

					cost						path				
Step	**M**	**N**	**update?**		**A**	**B**	**C**	**D**	**E**		**A**	**B**	**C**	**D**	**E**
1	C	B	10 > 5+3		0	**8**	5	∞	∞		A	**C**	A	•	•
		D	∞ > 5+9		0	8	5	**14**	∞		A	C	A	**C**	•
		E	∞ > 5+2		0	8	5	14	**7**		A	C	A	C	**C**
2	E	D	14 > 7+6		0	8	5	**13**	7		A	C	A	**E**	C
3	B	D	13 > 8+1		0	8	5	**9**	7		A	C	A	**B**	C