Optimization
ENGS 104, W2010
Lecture 17

George Cybenko

Agenda:

- Complete Approximation Algorithm for ΔTSP (triangular TSP)
- Heuristic/approximation methods for hard problems – overview

Last lecture:

- can make any TSP into ATSP by computing shortest paths between nodes
- Bellman-Ford algorithm for shortest paths
- General TSP can have no ε-approximation algorithm
- Minimal Spanning Tree Problem & Algorithm

Bridges of Konigsburg Problem

→ Can you end where you start and traverse <u>every</u> edge in a graph? (exactly <u>once</u>)

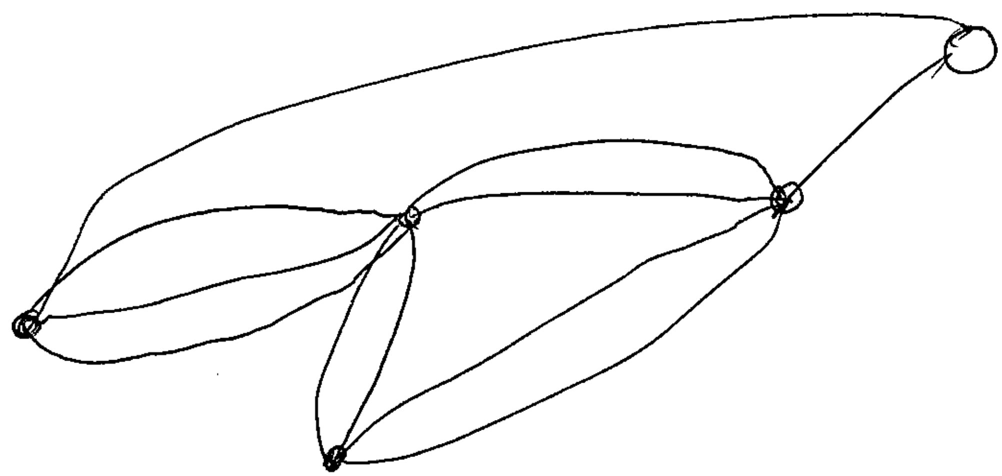
→ Eulerian graph ≡ graph in which this is possible

→ Who was Euler?

Eng 104, L17

We are now considering "multigraphs" where there may be multiple edges between two nodes.

Eg:



Euler's Theorem

A graph is Eulerian $\Longleftrightarrow$

a) graph is connected

b) "degree" of each node is even.

(degree = # edges incident on the node).

Proof: $\Rightarrow$ graph must be connected & every node is entered & exited by a unique edge

Proof: $\Leftarrow$ Assume $G=(V,E)$ is connected & each node has even degree.

$\Rightarrow$ Induction on $|E|$ = # ~~nodes~~ edges

Basis Graph with 0 edges and connected  Ok.

Two edges  OK.

General case: $|E| = 2m$ some m.

Start at any node & make a path (without repeating edges) until you return to start. Remove those edges. Resulting graph has:

Resulting graph has:

- fewer edges and;
- possibly disconnected components.

Each disconnected component has:

- even degree nodes (why?)

$\Rightarrow$ Eulerian path by induction

Now take your walk and piece together the Eulerian paths.

This is a nice "recursive" algorithm to write.

HW3 !

Back to ATSP.

(1) Construct a Minimal Spanning Tree for the graph in the ATSP, call it T.

Cost of Minimal Spanning Tree T

$$= C(T) = \sum_{\substack{\text{edges} \\ \text{in } T}} \text{edge costs}$$

$\le$ Cost of Minimal Tour for TSP

why??

Now make the MST, T, an Eulerian graph by doubling each edge ... each node has even degree & graph is connected.

"Cost" of the Eulerian path

$$= 2C(T) \ \$ \geq \min \Delta\text{TSP solution}$$

Why? (Eulerian path visits all nodes and so use Δ inequality)

<u>So</u>

$$\min \Delta TSP \geq C(T) \geq \frac{1}{2} \min \Delta TSP$$

So the tour produced by MST algorithm is a 1-approximation for the ΔTSP

$$\frac{|C(T) - \min \Delta TSP|}{|\min \Delta TSP|} \leq 2 = 1 + \epsilon$$

$\epsilon = 1$ here.

Followup: More work on solving TSP

- better approximation algorithms for special cases
  - eg Δ or Euclidean
- randomized algorithms.

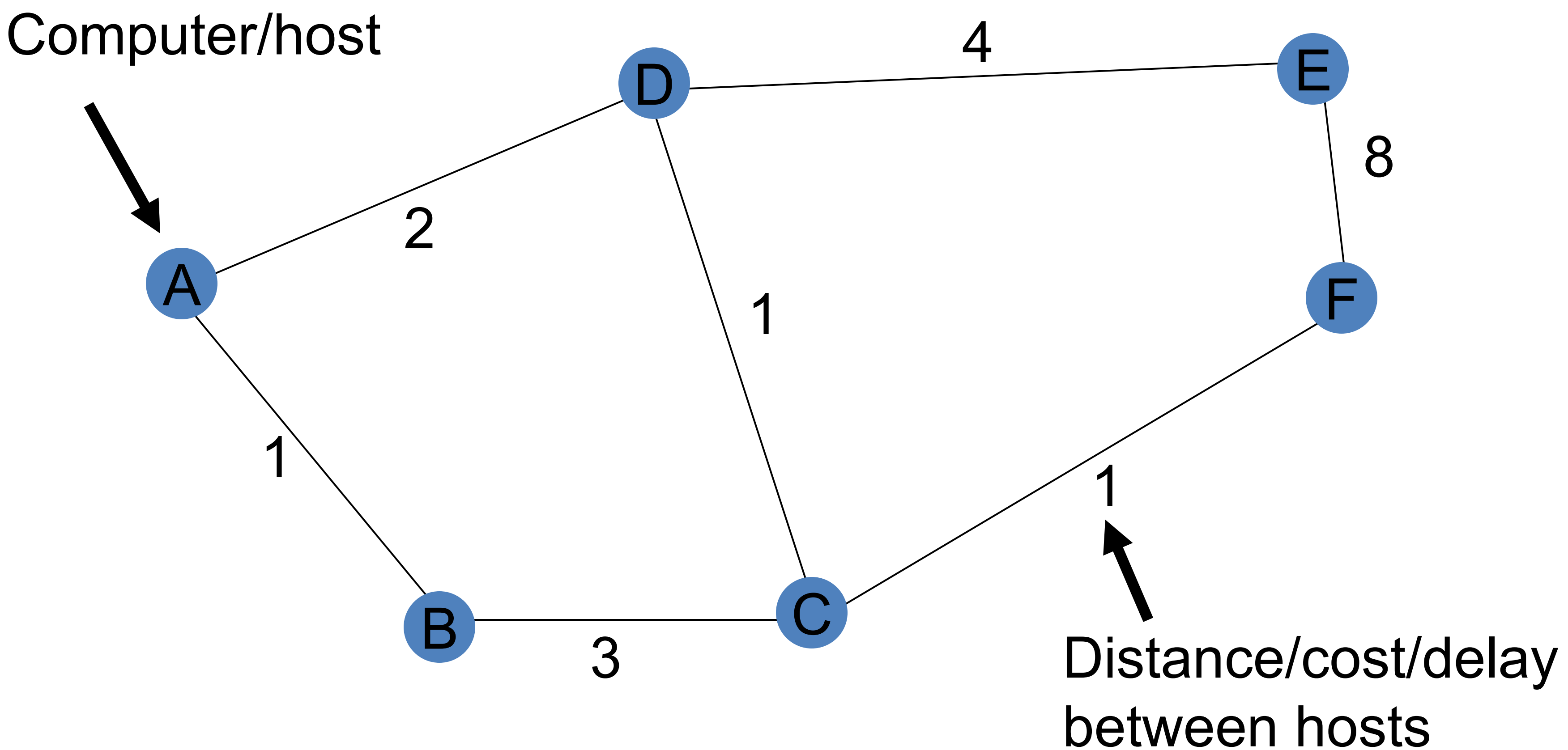
Minimal Spanning Tree Problem

- Definition
  - Tree
  - Spanning Tree
  - Cost of a tree
- A Greedy Algorithm (Kruskal)
- “Best” algorithms
- Compare with “Steiner’s Problem”

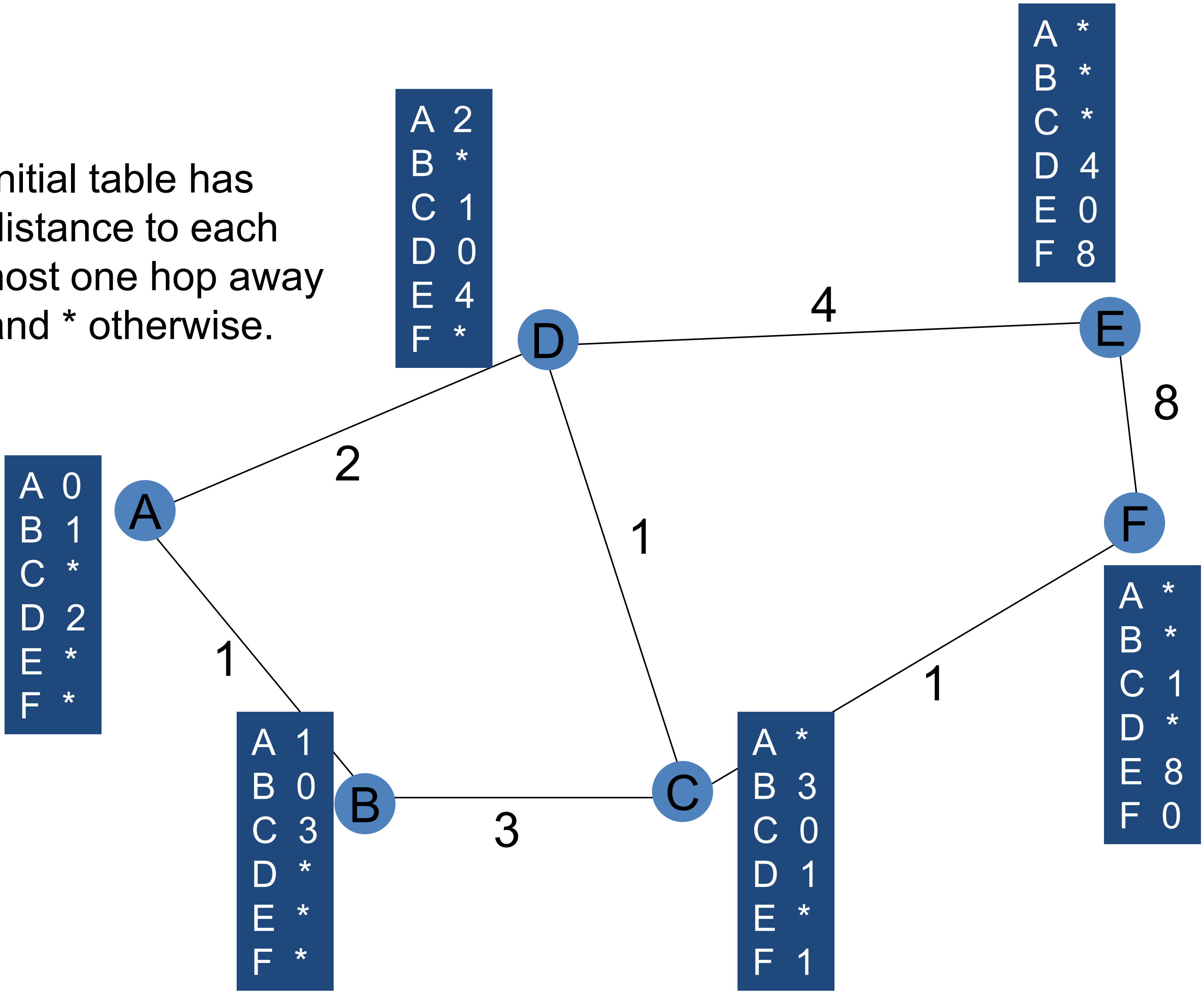
Shortest Path Problem

- Definition
- Basic Property (basis of dynamic programming)
- Some algorithms

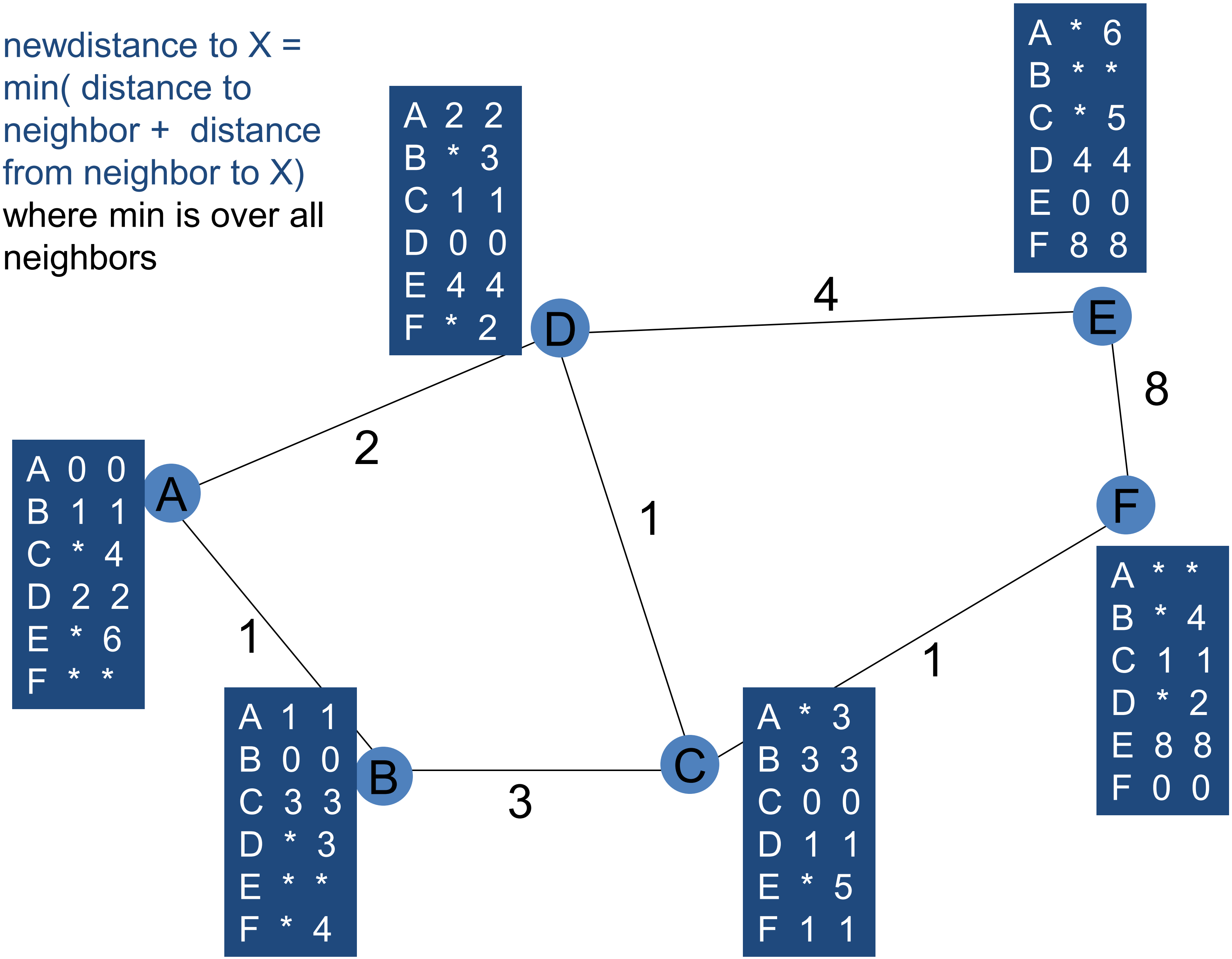
Bellman-Ford Routing

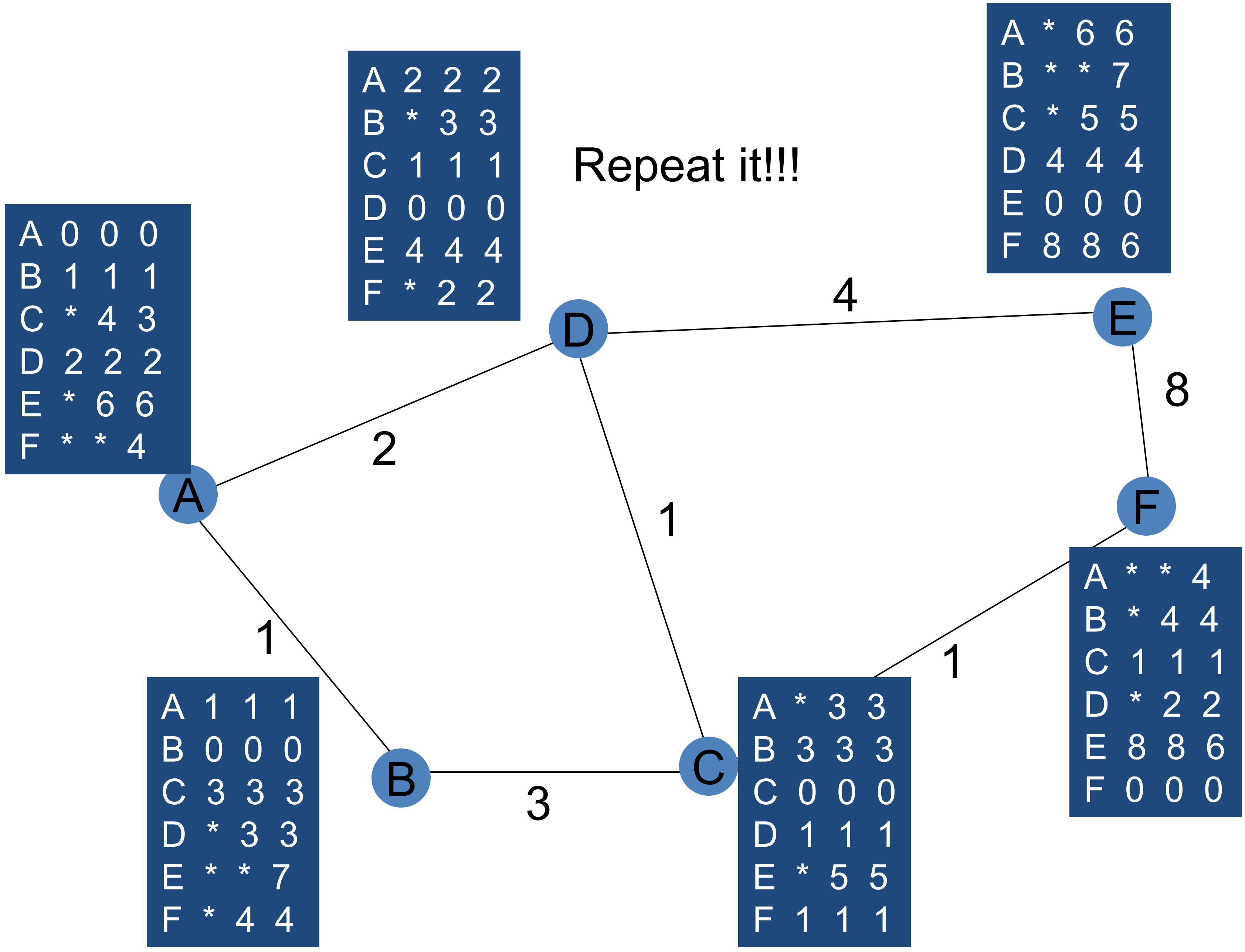


Initial table has distance to each host one hop away and * otherwise.



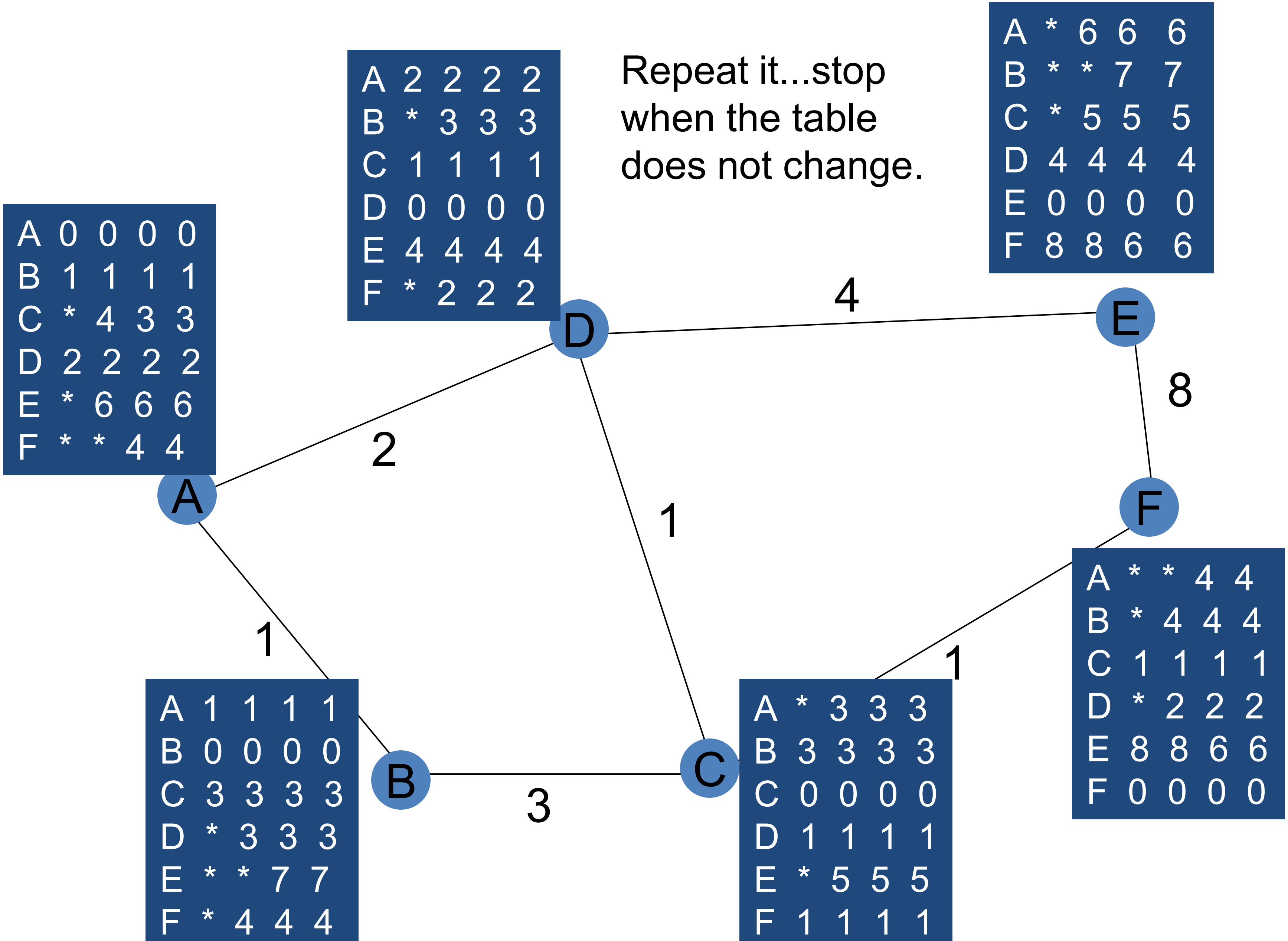
newdistance to X = min(distance to neighbor + distance from neighbor to X) where min is over all neighbors



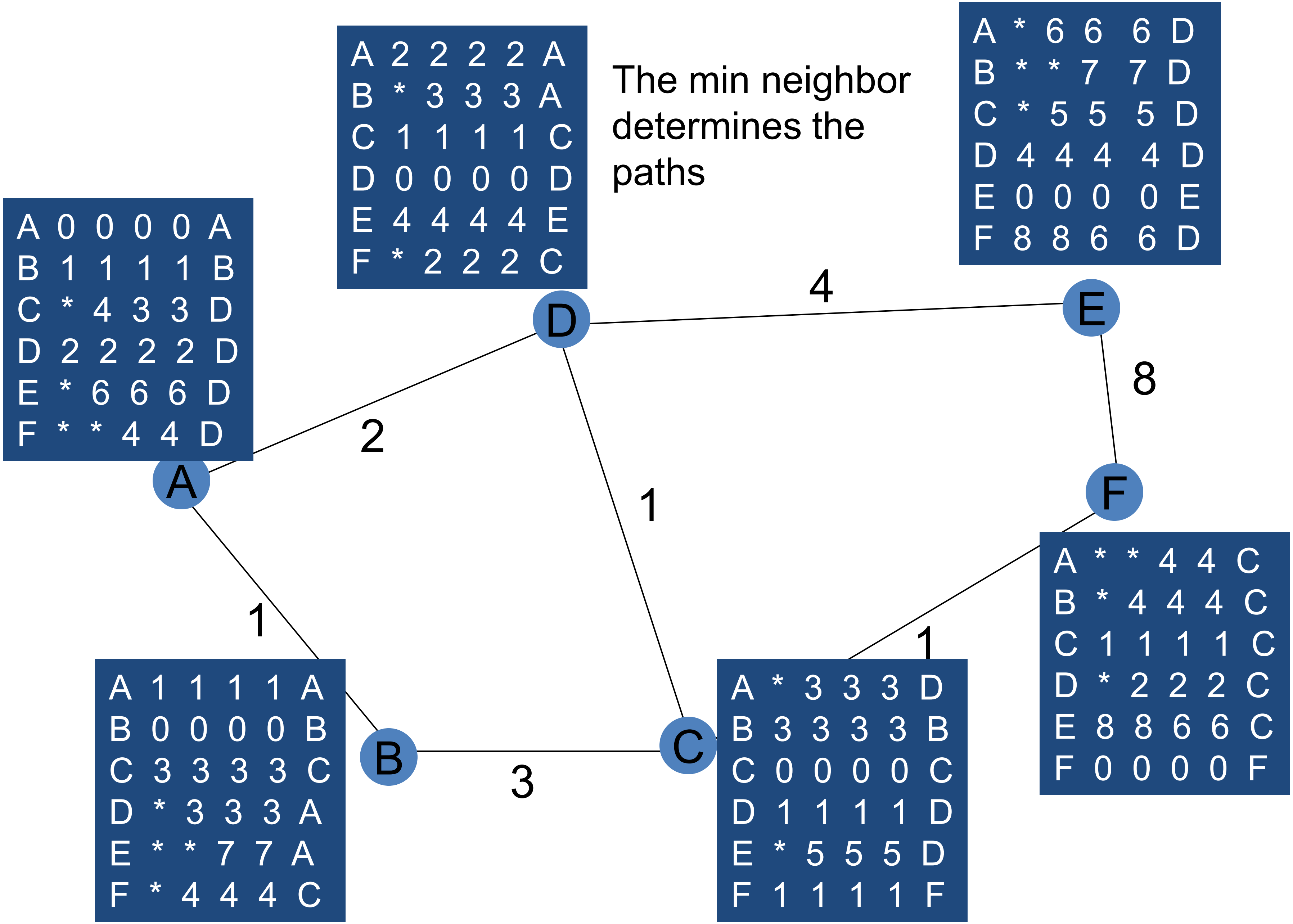


Repeat it!!!
A 0 0 0
B 1 1 1
C * 4 3
D 2 2 2
E * 6 6
F * * 4
A 2 2 2
B * 3 3
C 1 1 1
D 0 0 0
E 4 4 4
F * 2 2
A * 6 6
B * * 7
C * 5 5
D 4 4 4
E 0 0 0
F 8 8 6
A 1 1 1
B 0 0 0
C 3 3 3
D * 3 3
E * * 7
F * 4 4
A * 3 3
B 3 3 3
C 0 0 0
D 1 1 1
E * 5 5
F 1 1 1
A * * 4
B * 4 4
C 1 1 1
D * 2 2
E 8 8 6
F 0 0 0
A
B
C
D
E
F
2
1
3
1
4
8
1

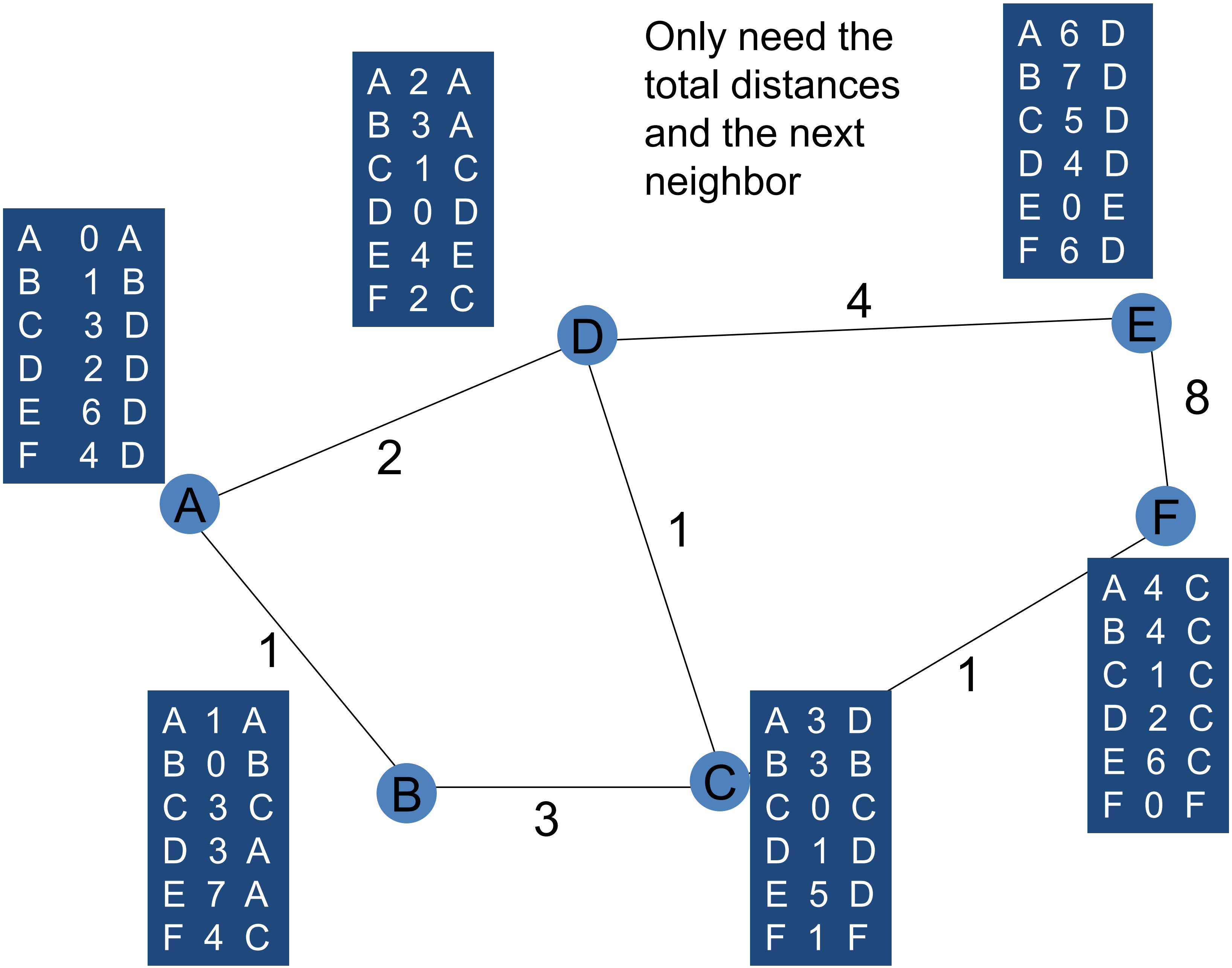
Repeat it...stop when the table does not change.

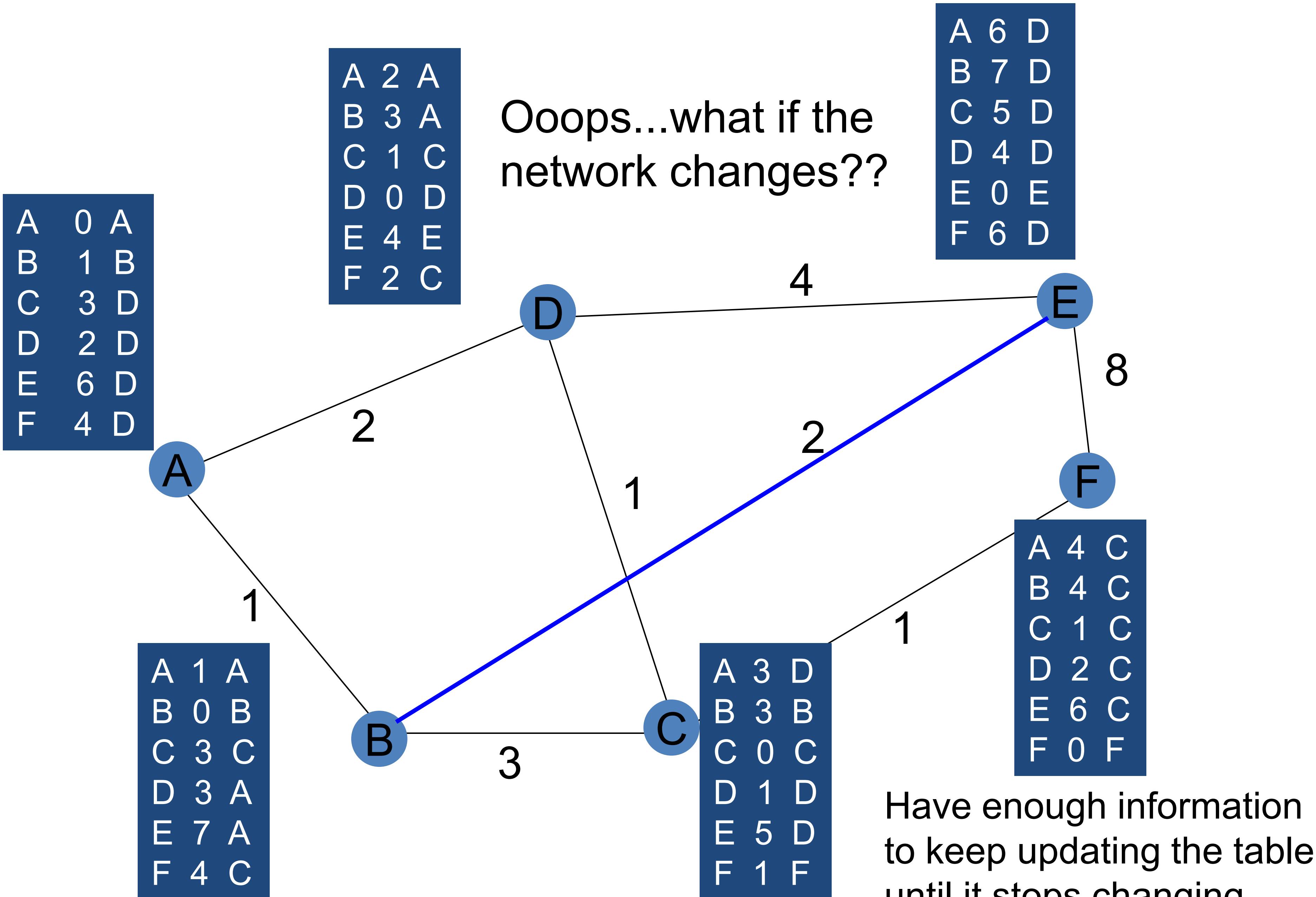


The min neighbor determines the paths



Only need the total distances and the next neighbor




Ooops...what if the network changes??
A 0 A
B 1 B
C 3 D
D 2 D
E 6 D
F 4 D
A 2 A
B 3 A
C 1 C
D 0 D
E 4 E
F 2 C
A 6 D
B 7 D
C 5 D
D 4 D
E 0 E
F 6 D
A 1 A
B 0 B
C 3 C
D 3 A
E 7 A
F 4 C
A 3 D
B 3 B
C 0 C
D 1 D
E 5 D
F 1 F
A 4 C
B 4 C
C 1 C
D 2 C
E 6 C
F 0 F
A
B
C
D
E
F
2
1
1
3
4
2
8
1
Have enough information to keep updating the table until it stops changing

Random/stochastic Algorithms for "hard" combinatorial problems.

- Key ingredients
  a) notion of neighboring candidate solution, ie if x is a candidate, ~~do we~~ how do we get a neighboring candidate?
  b) when to "accept" a candidate?

Let's consider some techniques for generating neighboring solutions

eg. ~~BM~~ Partition Problem

n items $1, \ldots, n$

costs $c_1, \ldots, c_n$

Candidate partition/solution: a subset of $\{1, \ldots, n\}$ ie an n-vector of 0's & 1's

Candidate solution

$$X = (x_1, \ldots, x_n) \qquad x_j = 0 \text{ or } 1$$

$$\text{Cost}(X) = \sum_{i=1}^{n} x_i c_i$$

Goal: $\text{cost}(X) = \frac{1}{2} \sum_{i=1}^{n} c_i$

A "neighbor" of x is a vector y with $x = y$ except for one coordinate. IE, flip one bit in x.

So each candidate x has n vectors in this sense of "neighborhood".

If we allow 2 bits to be changed, neighborhood has $\approx \frac{n^2}{2}$ solutions, etc.

Algorithm: Simple neighborhood search

- Start with a candidate solution
- Find a neighbor which is a better solution
- If no neighbors are better, stop

Sometimes this works!

Eg Sort a list of numbers.

Neighborhood: adjacent positions are swapped

Algorithm reduces to "Bubblesort"

Optimal solution always found!

How about simple neighborhood search for partition or TSP?

Fact: "local" minima with respect to the notion of neighborhood exist if the "neighborhoods" are small

__So__ we need mechanisms for "leaving" a neighborhood sometimes

$\Rightarrow$ randomized searches

- simulated annealing
- genetic algorithms

__Next__