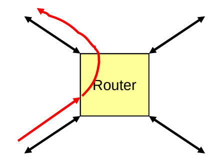
15-441 Computer Networking

Lecture 10: Intra-Domain Routing

RIP (Routing Information Protocol) &
OSPF (Open Shortest Path First)

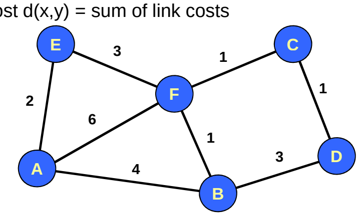
IP Forwarding

- The Story So Far…
  - IP addresses are structure to reflect Internet structure
  - IP packet headers carry these addresses
  - When Packet Arrives at Router
    - Examine header to determine intended destination
    - *Look up in table to determine next hop in path*
    - Send packet out appropriate port
- This/next lecture
  - How to generate the forwarding table



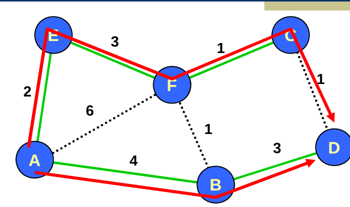
Graph Model

- Represent each router as node
- Direct link between routers represented by edge
  - Symmetric links ⇒ undirected graph
- Edge "cost" c(x,y) denotes measure of difficulty of using link
  - delay, $ cost, or congestion level
- Task
  - Determine least cost path from every node to every other node
    - Path cost d(x,y) = sum of link costs



Routes from Node A

Forwarding Table for A		
Dest	Cost	Next Hop
A	0	A
B	4	B
C	6	E
D	7	B
E	2	E
F	5	E



- Properties
  - Some set of shortest paths forms tree
    - Shortest path spanning tree
  - Solution not unique
    - E.g., A-E-F-C-D also has cost 7

Ways to Compute Shortest Paths

- Centralized
  - Collect graph structure in one place
  - Use standard graph algorithm
  - Disseminate routing tables
- Distance-vector
  - No one has copy of graph
  - Nodes construct their own tables iteratively
  - Each sends information about its table to neighbors
- Link-state
  - Every node collects complete graph structure
  - Each computes shortest paths from it
  - Each generates own routing table

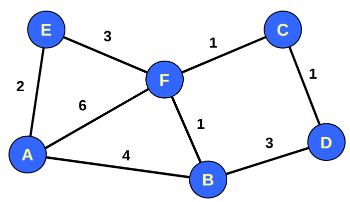
Outline

- Distance Vector
- Link State
- Routing Hierarchy

Distance-Vector Method

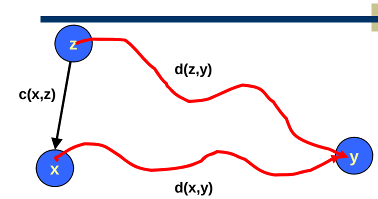
Initial Table for A

Dest	Cost	Next Hop
A	0	A
B	4	B
C	∞	–
D	∞	–
E	2	E
F	6	F



- Idea
  - At any time, have cost/next hop of best known path to destination
  - Use cost ∞ when no path known
- Initially
  - Only have entries for directly connected nodes

Distance-Vector Update



- Update(x,y,z)

  d ← c(x,z) + d(z,y) # Cost of path from x to y with first hop z

  if d < d(x,y)

    # Found better path

    return d,z # Updated cost / next hop

  else

    return d(x,y), nexthop(x,y) # Existing cost / next hop

Algorithm

- Bellman-Ford algorithm
- Repeat

  For every node x

    For every neighbor z

      For every destination y

        d(x,y) ← Update(x,y,z)
- Until converge

Start

Optimum 1-hop paths

Table for A			Table for B		
Dst	Cst	Hop	Dst	Cst	Hop
A	0	A	A	4	A
B	4	B	B	0	B
C	∞	–	C	∞	–
D	∞	–	D	3	D
E	2	E	E	∞	–
F	6	F	F	1	F

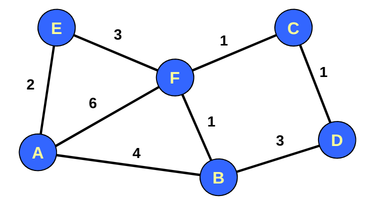


Table for C			Table for D			Table for E			Table for F		
Dst	Cst	Hop	Dst	Cst	Hop	Dst	Cst	Hop	Dst	Cst	Hop
A	∞	–	A	∞	–	A	2	A	A	6	A
B	∞	–	B	3	B	B	∞	–	B	1	B
C	0	C	C	1	C	C	∞	–	C	1	C
D	1	D	D	0	D	D	∞	–	D	∞	–
E	∞	–	E	∞	–	E	0	E	E	3	E
F	1	F	F	∞	–	F	3	F	F	0	F

Iteration #1

Optimum 2-hop paths

Table for A			Table for B		
Dst	Cst	Hop	Dst	Cst	Hop
A	0	A	A	4	A
B	4	B	B	0	B
C	7	F	C	2	F
D	7	B	D	3	D
E	2	E	E	4	F
F	5	E	F	1	F

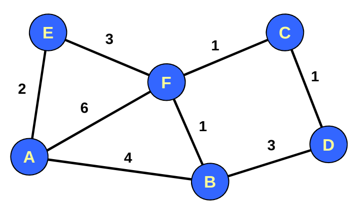


Table for C			Table for D			Table for E			Table for F		
Dst	Cst	Hop	Dst	Cst	Hop	Dst	Cst	Hop	Dst	Cst	Hop
A	7	F	A	7	B	A	2	A	A	5	B
B	2	F	B	3	B	B	4	F	B	1	B
C	0	C	C	1	C	C	4	F	C	1	C
D	1	D	D	0	D	D	∞	–	D	2	C
E	4	F	E	∞	–	E	0	E	E	3	E
F	1	F	F	2	C	F	3	F	F	0	F

Iteration #2

Optimum 3-hop paths

Table for A			Table for B		
Dst	Cst	Hop	Dst	Cst	Hop
A	0	A	A	4	A
B	4	B	B	0	B
C	6	E	C	2	F
D	7	B	D	3	D
E	2	E	E	4	F
F	5	E	F	1	F

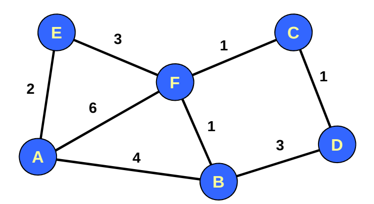
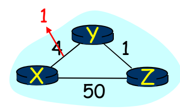


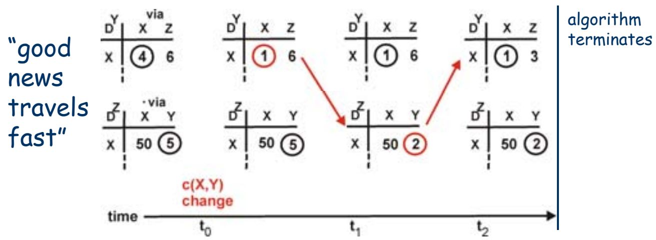
Table for C			Table for D			Table for E			Table for F		
Dst	Cst	Hop	Dst	Cst	Hop	Dst	Cst	Hop	Dst	Cst	Hop
A	6	F	A	7	B	A	2	A	A	5	B
B	2	F	B	3	B	B	4	F	B	1	B
C	0	C	C	1	C	C	4	F	C	1	C
D	1	D	D	0	D	D	5	F	D	2	C
E	4	F	E	5	C	E	0	E	E	3	E
F	1	F	F	2	C	F	3	F	F	0	F

Distance Vector: Link Cost Changes

Link cost changes:

- Node detects local link cost change
- Updates distance table
- If cost change in least cost path, notify neighbors

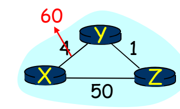


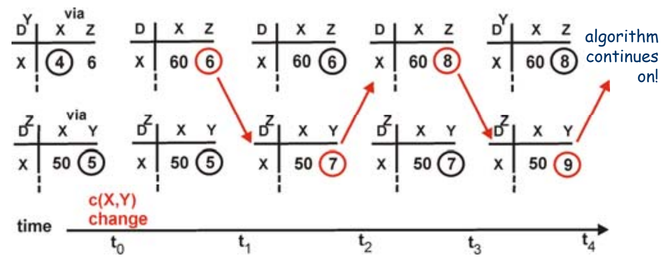


Distance Vector: Link Cost Changes

Link cost changes:

- Good news travels fast
- Bad news travels slow - "count to infinity" problem!

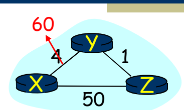


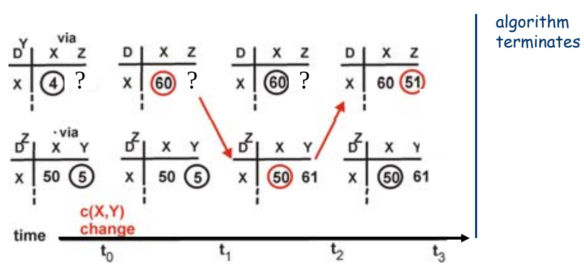


Distance Vector: Split Horizon

If Z routes through Y to get to X :

- Z does not advertise its route to X back to Y

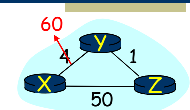


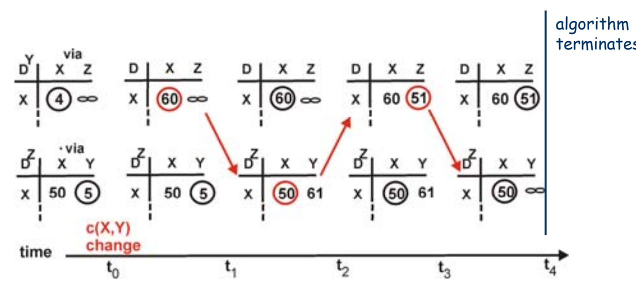


Distance Vector: Poison Reverse

If Z routes through Y to get to X :

- Z tells Y its (Z's) distance to X is infinite (so Y won't route to X via Z)
- Eliminates some possible timeouts with split horizon
- Will this completely solve count to infinity problem?





Poison Reverse Failures

Table for A			Table for B			Table for D			Table for F		
Dst	Cst	Hop	Dst	Cst	Hop	Dst	Cst	Hop	Dst	Cst	Hop
C	7	F	C	8	A	C	9	B	C	1	C

Table for A (Forced Update)

Dst	Cst	Hop
C	∞	–

Table for F (Forced Update)

Dst	Cst	Hop
C	∞	–

Table for A (Better Route)

Dst	Cst	Hop
C	13	D

Table for B (Forced Update)

Dst	Cst	Hop
C	14	A

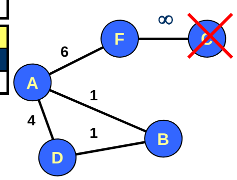
Table for D (Forced Update)

Dst	Cst	Hop
C	15	B

Table for A (Forced Update)

Dst	Cst	Hop
C	19	D

•
•
•



- Iterations don't converge
- "Count to infinity"
- Solution
  - Make "infinity" smaller
- What is upper bound on maximum path length?

Routing Information Protocol (RIP)

- Earliest IP routing protocol (1982 BSD)
  - Current standard is version 2 (RFC 1723) which includes CIDR
- Features
  - Every link has cost 1
  - "Infinity" = 16
    - Limits to networks where everything reachable within 15 hops
- Sending Updates
  - Every router listens for updates on UDP port 520
  - RIP message can contain entries for up to 25 table entries

RIP Updates

- Initial
  - When router first starts, asks for copy of table for every neighbor
  - Uses it to iteratively generate own table
- Periodic
  - Every 30 seconds, router sends copy of its table to each neighbor
  - Neighbors use to iteratively update their tables
- Triggered
  - When every entry changes, send copy of entry to neighbors
    - Except for one causing update (split horizon rule)
  - Neighbors use to update their tables

RIP Staleness / Oscillation Control

- Small Infinity
  - Count to infinity doesn't take very long
- Route Timer
  - Every route has timeout limit of 180 seconds
    - Reached when haven't received update from next hop for 6 periods
  - If not updated, set to infinity
  - Soft-state refresh → important concept!!!
- Behavior
  - When router or link fails, can take minutes to stabilize

Outline

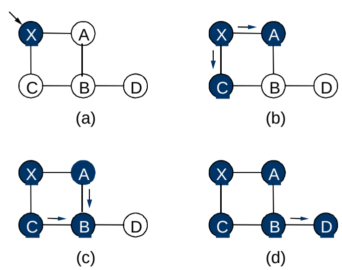
- Distance Vector
- Link State
- Routing Hierarchy

Link State Protocol Concept

- Every node gets complete copy of graph
  - Every node “floods” network with data about its outgoing links
- Every node computes routes to every other node
  - Using single-source, shortest-path algorithm
- Process performed whenever needed
  - When connections die / reappear

Sending Link States by Flooding

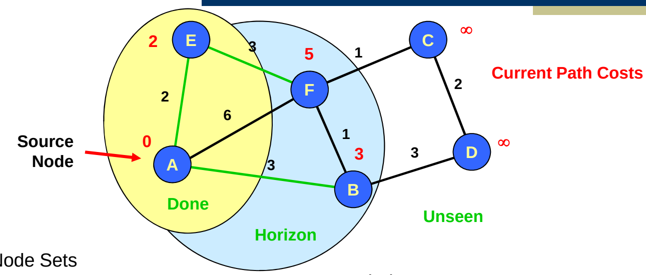
- X Wants to Send Information
  - Sends on all outgoing links
- When Node Y Receives Information from Z
  - Send on all links other than Z



Dijkstra’s Algorithm

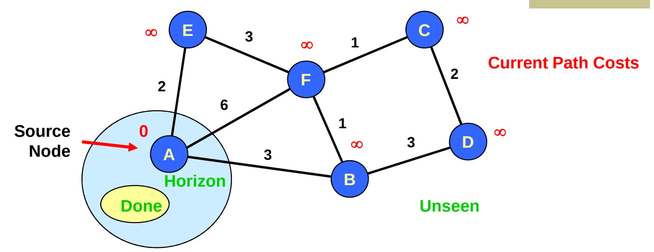
- Given
  - Graph with source node s and edge costs c(u,v)
  - Determine least cost path from s to every node v
- Shortest Path First Algorithm
  - Traverse graph in order of least cost from source

Dijkstra's Algorithm: Concept



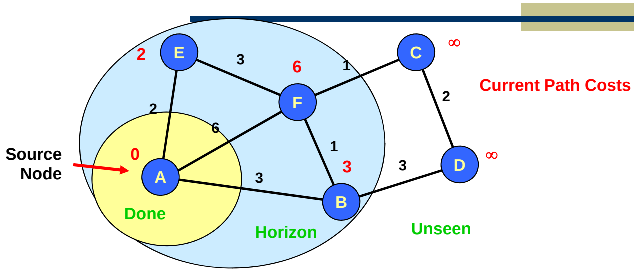
- Node Sets
  - Done
    - Already have least cost path to it
  - Horizon:
    - Reachable in 1 hop from node in Done
  - Unseen:
    - Cannot reach directly from node in Done
- Label
  - d(v) = path cost
    - From s to v
- Path
  - Keep track of last link in path

Dijkstra's Algorithm: Initially



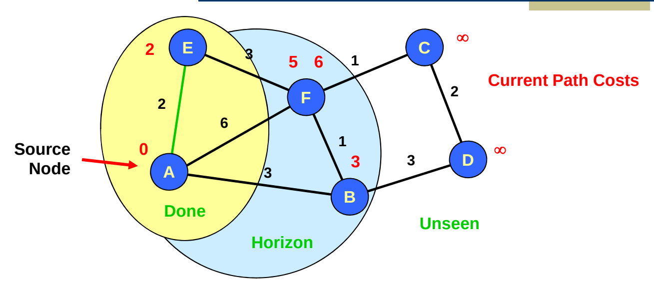
- No nodes done
- Source in horizon

Dijkstra's Algorithm: Initially



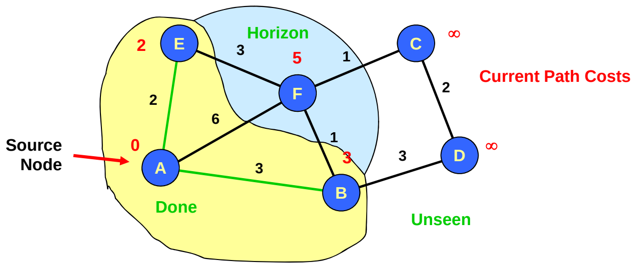
- d(v) to node A shown in red
  - Only consider links from done nodes

Dijkstra's Algorithm



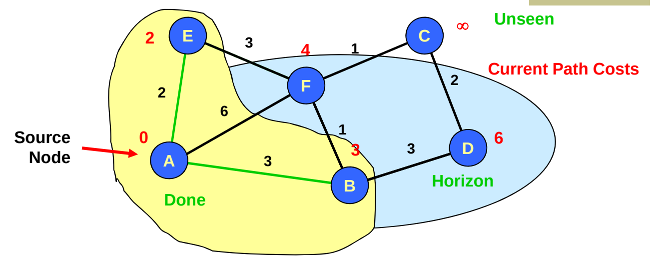
- Select node v in horizon with minimum d(v)
- Add link used to add node to shortest path tree
- Update d(v) information

Dijkstra's Algorithm



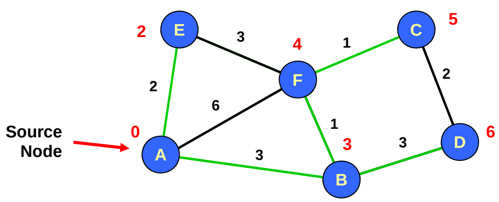
- Repeat…

Dijkstra's Algorithm



- Update d(v) values
  - Can cause addition of new nodes to horizon

Dijkstra's Algorithm

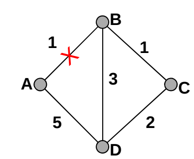


- Final tree shown in green

Link State Characteristics

- With consistent LSDBs*, all nodes compute consistent loop-free paths
- Can still have transient loops

*Link State Data Base



Packet from C→A may loop around BDC if B knows about failure and C & D do not

OSPF Routing Protocol

- Open
  - Open standard created by IETF
- Shortest-path first
  - Another name for Dijkstra’s algorithm
- More prevalent than RIP

OSPF Reliable Flooding

- Transmit link state advertisements (LSA)
  - Originating router
    - Typically, minimum IP address for router
  - Link ID
    - ID of router at other end of link
  - Metric
    - Cost of link
  - Link-state age
    - Incremented each second
    - Packet expires when reaches 3600
  - Sequence number
    - Incremented each time sending new link information

OSPF Flooding Operation

- Node X Receives LSA from Node Y
  - With Sequence Number q
  - Looks for entry with same origin/link ID
- Cases
  - No entry present
    - Add entry, propagate to all neighbors other than Y
  - Entry present with sequence number p < q
    - Update entry, propagate to all neighbors other than Y
  - Entry present with sequence number p > q
    - Send entry back to Y
    - To tell Y that it has out-of-date information
  - Entry present with sequence number p = q
    - Ignore it

Flooding Issues

- When should it be performed
  - Periodically
  - When status of link changes
    - Detected by connected node
- What happens when router goes down & back up
  - Sequence number reset to 0
    - Other routers may have entries with higher sequence numbers
  - Router will send out LSAs with number 0
  - Will get back LSAs with last valid sequence number p
  - Router sets sequence number to p+1 & resends

Adoption of OSPF

- RIP viewed as outmoded
  - Good when networks small and routers had limited memory & computational power
- OSPF Advantages
  - Fast convergence when configuration changes

Comparison of LS and DV Algorithms

Message complexity
- LS: with n nodes, E links, O(nE) messages
- DV: exchange between neighbors only O(E)

Speed of Convergence
- LS: Complex computation
  - But…can forward before computation
  - may have oscillations
- DV: convergence time varies
  - may be routing loops
  - count-to-infinity problem
  - (faster with triggered updates)

Space requirements:
- LS maintains entire topology
- DV maintains only neighbor state

Comparison of LS and DV Algorithms

Robustness: what happens if router malfunctions?

LS:
- node can advertise incorrect *link* cost
- each node computes only its *own* table

DV:
- DV node can advertise incorrect *path* cost
- each node's table used by others
  - errors propagate thru network

- Other tradeoffs
  - Making LSP flood reliable

Outline

- Distance Vector
- Link State
- Routing Hierarchy

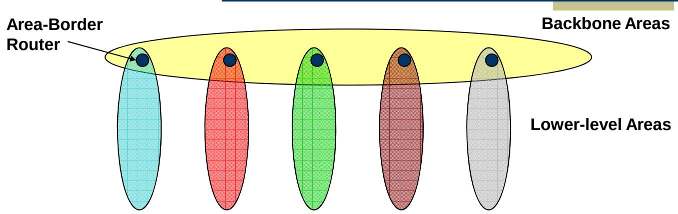
Routing Hierarchies

- Flat routing doesn't scale
  - Storage → Each node cannot be expected to store routes to every destination (or destination network)
  - Convergence times increase
  - Communication → Total message count increases
- Key observation
  - Need less information with increasing distance to destination
  - Need lower diameters networks
- Solution: area hierarchy

Areas

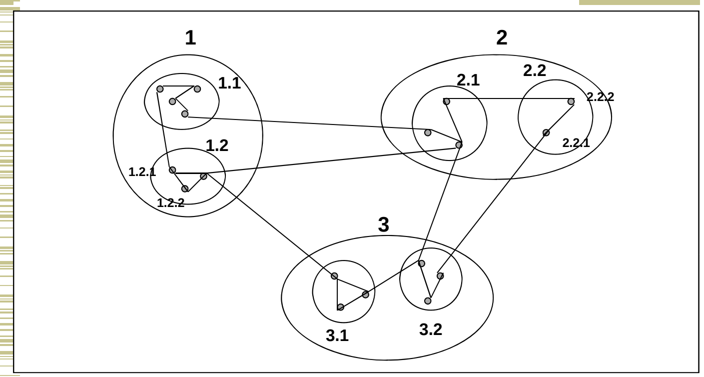
- Divide network into areas
  - Areas can have nested sub-areas
- Hierarchically address nodes in a network
  - Sequentially number top-level areas
  - Sub-areas of area are labeled relative to that area
  - Nodes are numbered relative to the smallest containing area

Routing Hierarchy



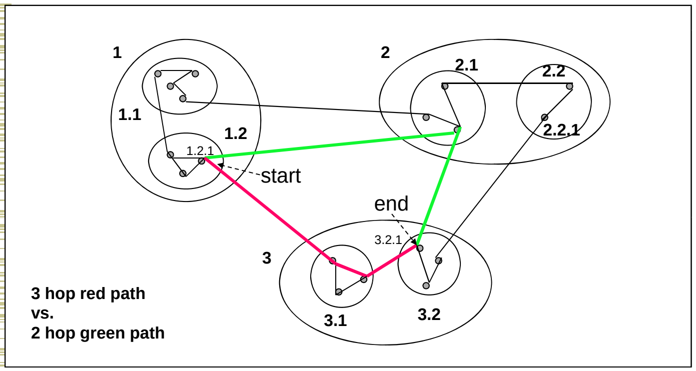
- Partition Network into "Areas"
  - Within area
    - Each node has routes to every other node
  - Outside area
    - Each node has routes for other top-level areas only
    - Inter-area packets are routed to nearest appropriate border router
- Constraint: no path between two sub-areas of an area can exit that area

Area Hierarchy Addressing



Path Sub-optimality

- Can result in sub-optimal paths



Next Lecture: BGP

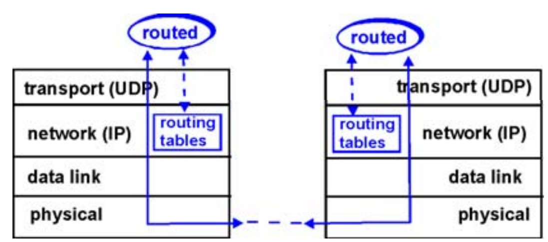
- How to connect together different ISPs

EXTRA SLIDES

The rest of the slides are FYI

RIP Table Processing

- RIP routing tables managed by **application-level** process called route-d (daemon)
- advertisements sent in UDP packets, periodically repeated



Dijsktra's Algorithm

```
Initialization:
  N = {A}
  for all nodes v
    if v adjacent to A
      then D(v) = c(A,v)
      else D(v) = infinity

Loop
  find w not in N such that D(w) is a minimum
  add w to N
  update D(v) for all v adjacent to w and not in N:
     D(v) = min( D(v), D(w) + c(w,v) )
  /* new cost to v is either old cost to v or known
   shortest path cost to w plus cost from w to v */
until all nodes in N
```


Dijkstra's algorithm: example

Step	start N	D(B),p(B)	D(C),p(C)	D(D),p(D)	D(E),p(E)	D(F),p(F)
0	A	2,A	5,A	1,A	infinity	infinity
1	AD	2,A	4,D		2,D	infinity
2	ADE	2,A	3,E			4,E
3	ADEB		3,E			4,E
4	ADEBC					4,E
5	ADEBCF