Search Fundamentals

Radu Nicolescu
Department of Computer Science
University of Auckland

12 Aug 2020

1. Distributed DFS and BFS
2. ClassicalDFS
3. CidonDFS §1
4. CidonDFS §2
5. Bellman-Ford algorithm
6. Maximal Independent Set

Distributed DFS

- Despite its sequential appearance, DFS can work faster on distributed networks!
- Classical DFS traverses each edge (tree and frond) twice, thus time complexity = message complexity $= 2|E|$

- Like other distributed versions, Cidon DFS traverses each tree edge twice (needed), but avoids all or most fronds, thus time complexity $= 2|V| - 2$, but at the possible expense of increased message complexity $= 3|E|$

  - Small error in Tel's text: message complexity is not $4|E|$...
- Cidon DFS was designed for async networks, thus also works for sync networks (even better).

Distributed BFS

- For sync networks, Echo (aka SyncBFS) finds a BFS spanning tree: time complexity $= 2D + 1$, message complexity $= 2|E|$
- Despite its parallel appearance, BFS is harder to implement on async networks and does not get all the expected benefits (bad time *or* bad message complexity).
- In fact, on async networks, BFS looks like a "milder" version of Bellman-Ford: shows similar issues, but less severe
- AsyncBFS [Lynch]: messages $= O(D|E|)$; time−pileups $= O(D)$; time+pileups $= O(D|V|)$
- LayeredBFS [Lynch]: messages $= O(D|V| + |E|)$; time−pileups $= O(D^2)$
- We do not further follow these issues here...

Cidon DFS

- Cidon DFS uses two types of tokens:
  - One single classical **tok** token, sent on tree edges and, *occasionally*, on fronds
  - New **vis** notification tokens, used to mark possible fronds
- A **vis** token can be sent:
  - ahead of **tok**
  - together with **tok** (can be combined)
  - alone (on fronds only)

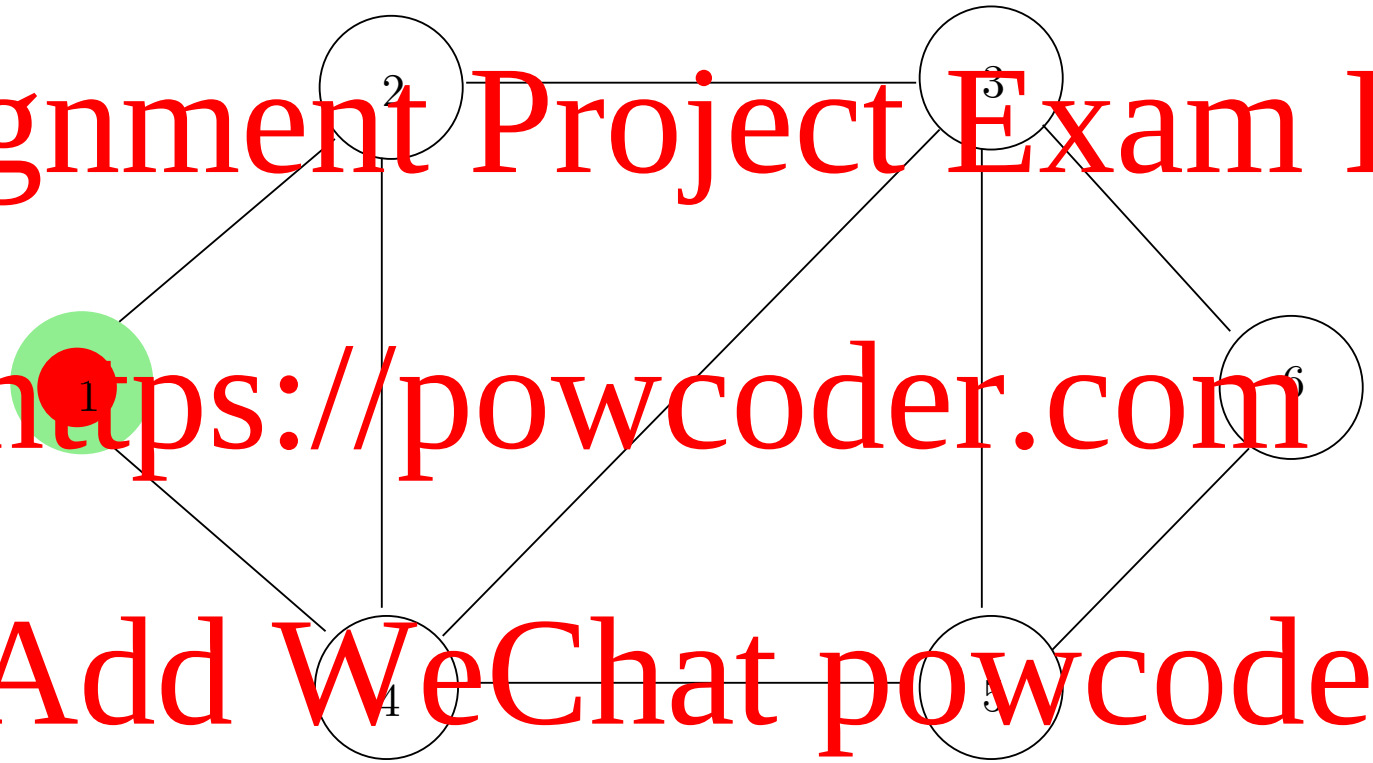
Read more: Tel §6.4, or Cidon's original paper:
Yet Another Distributed Depth-First-Search Algorithm

`http://cidon.eew.technion.ac.il/files/var/448324-cidon_dfs_87.pdf`

Classical vs Cidon DFS – Examples

- In all cases, there is one single **tok** token, thus the time complexity sums the delays in this token's delivery
- In classical DFS, frond edge $\{2, 4\}$ is sequentially twice traversed by the **tok** token, which adds 2 time intervals
- In Cidon DFS §1, frond edge $\{2, 4\}$ is twice traversed by **vis** tokens, in parallel with the progress of the **tok** token, which now saves 2 time intervals (while keeping messages count)
- In Cidon DFS §2, frond edge $\{2, 4\}$ is overlappingly traversed
  - in one direction, by the **vis** token, in parallel with the progress of the **tok** token,
  - in the reverse direction, by a pair of **tok+vis** tokens (**vis** is not strictly necessary)
  - this still saves 1 time interval (but increases messages count)

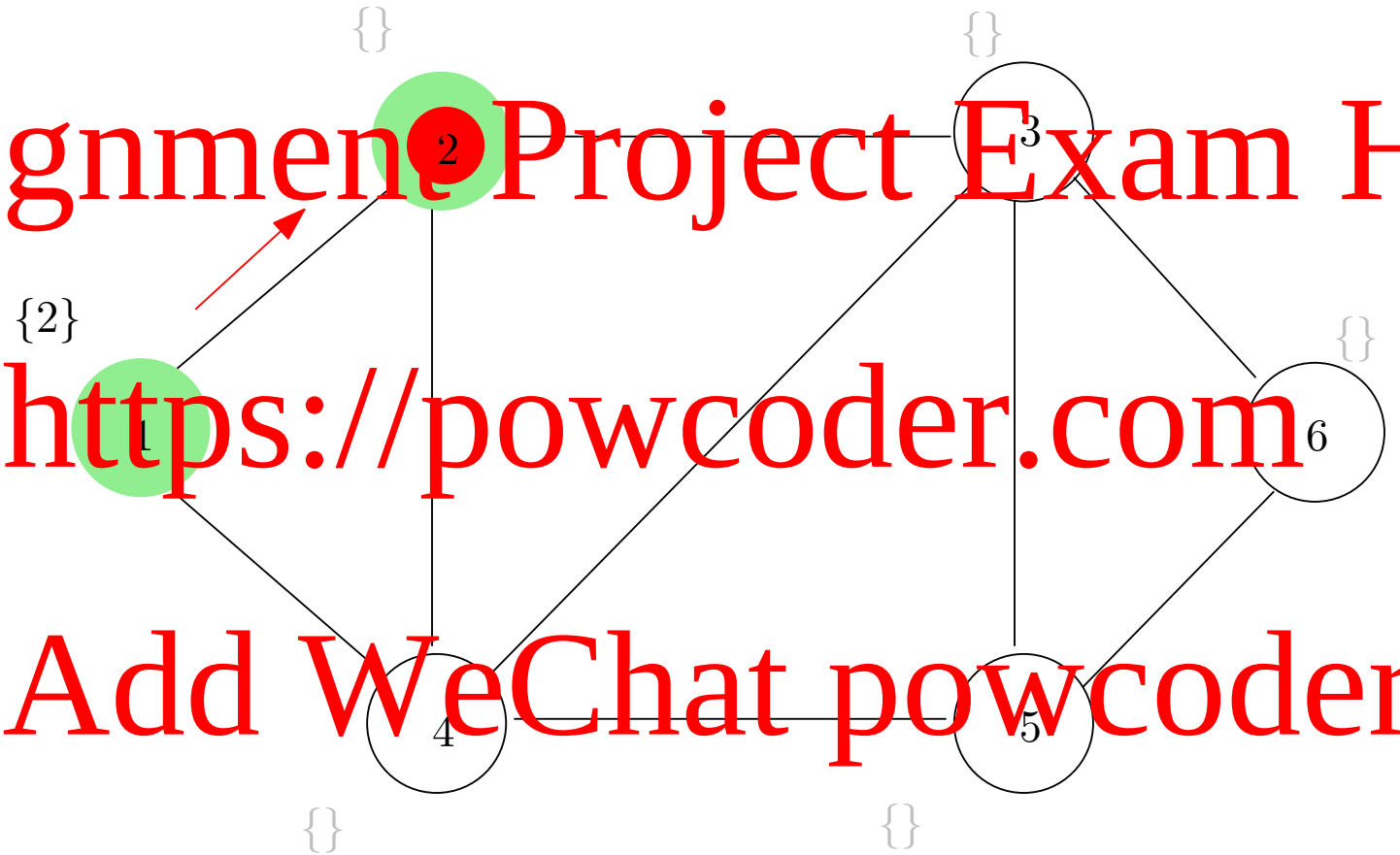
Classical DFS



Time Units = 0

Messages = 0

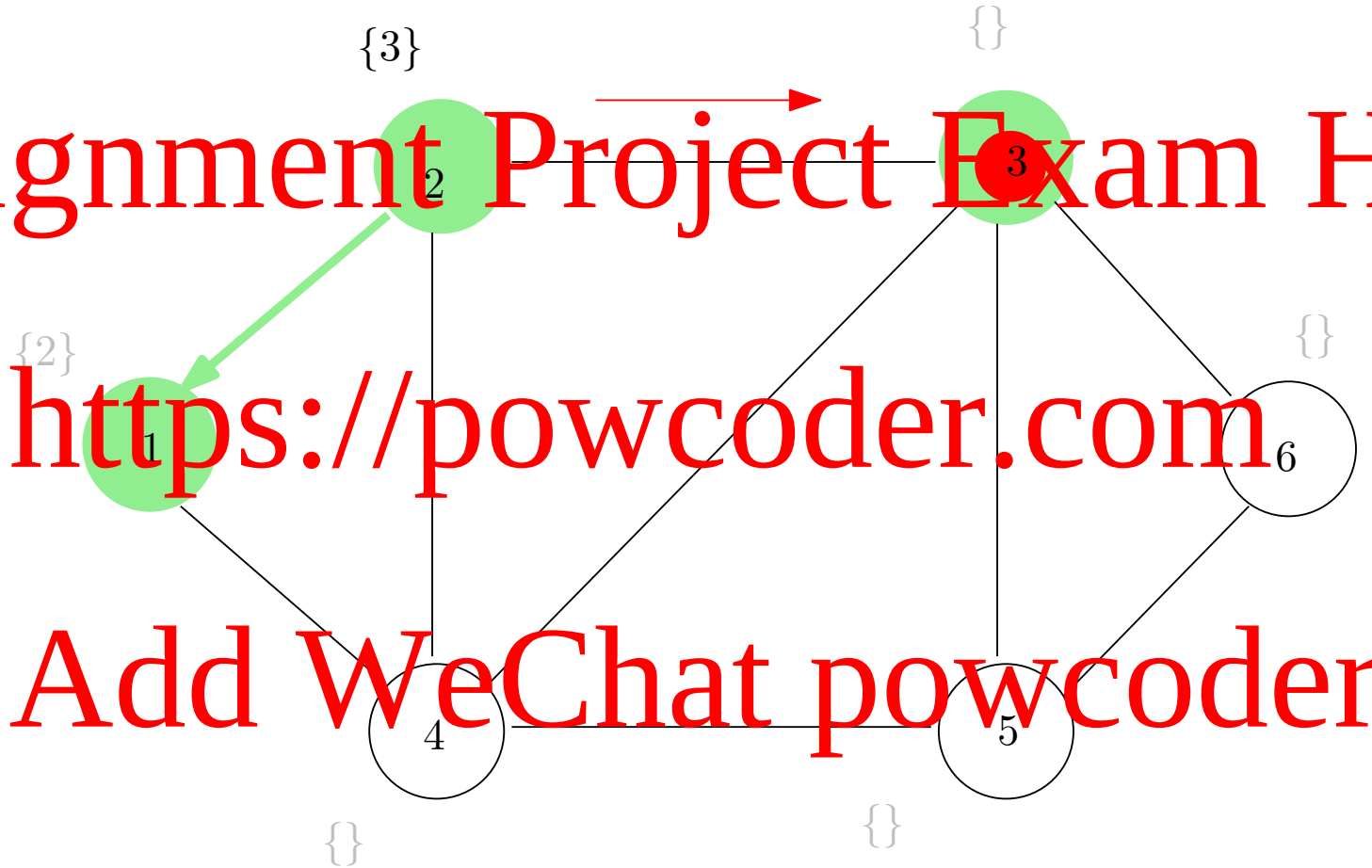
Classical DFS



Time Units = 1

Messages = 1

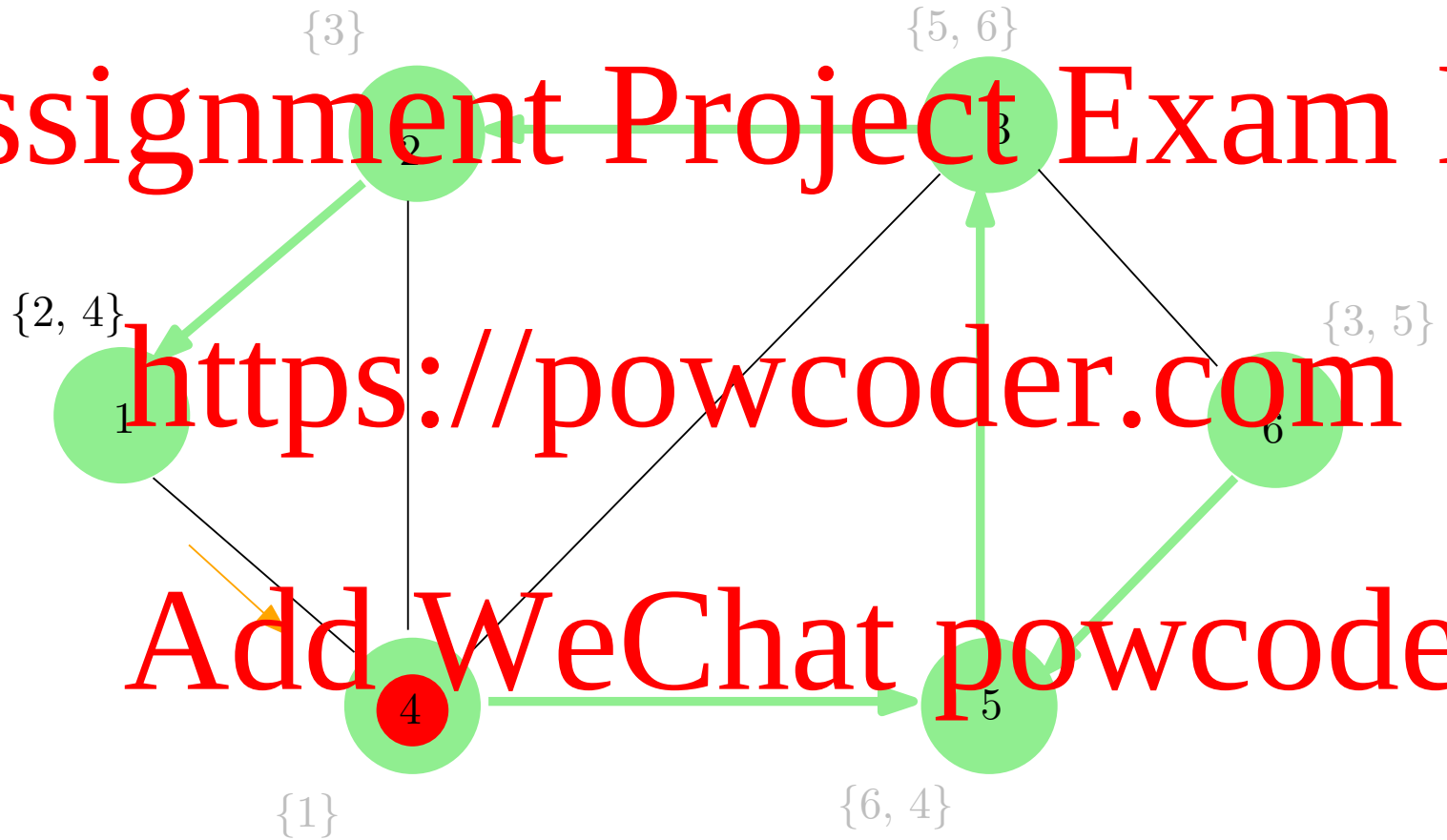
Classical DFS



Time Units = 2

Message = 2

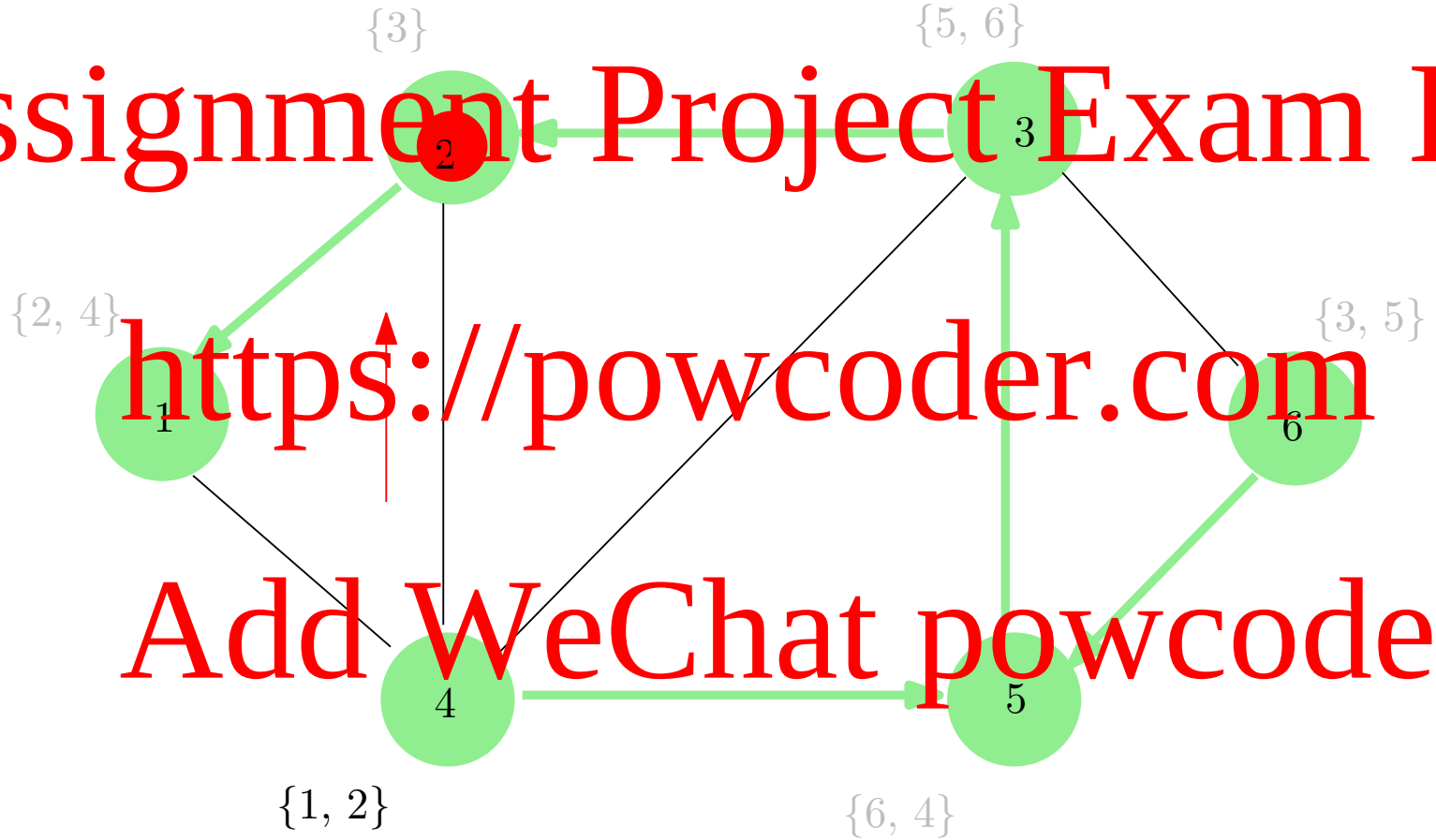
Classical DFS



Time Units = 10

Message = 10

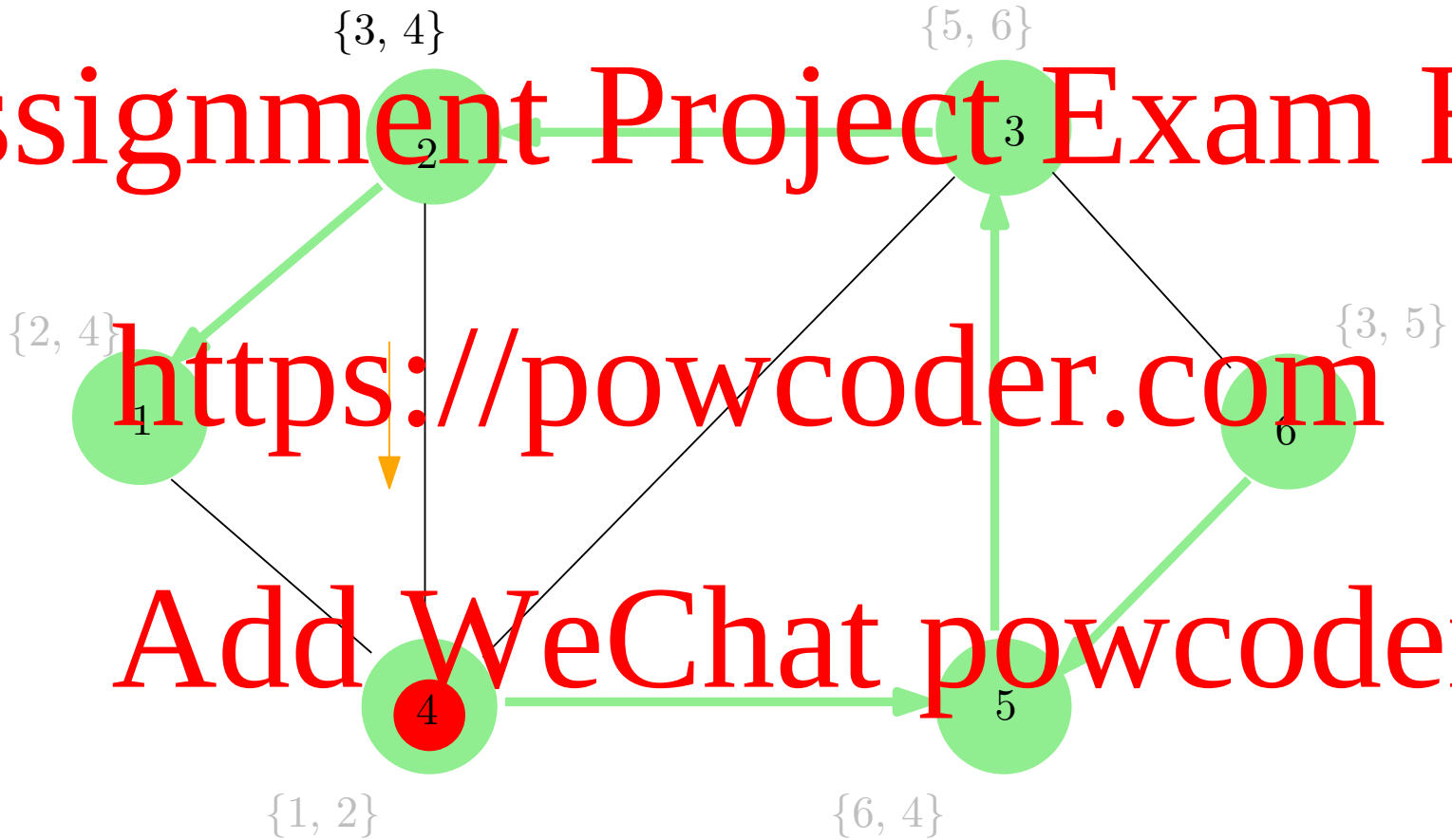
Classical DFS : 4 → **tok** → 2



Time Units = 11

Message = 11

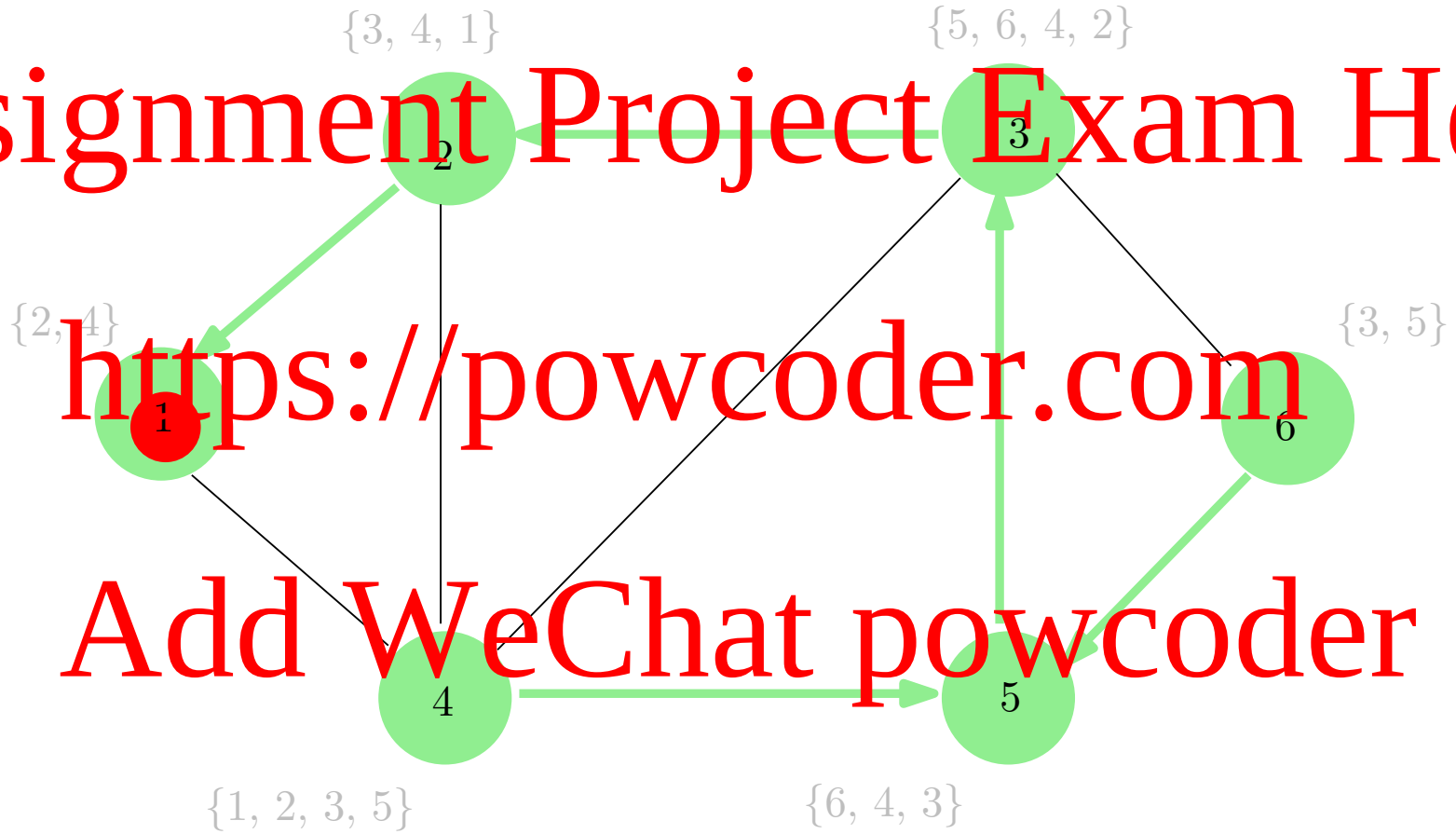
Classical DFS : 2 → **tok** → 4



Time Units = 12

Message = 12

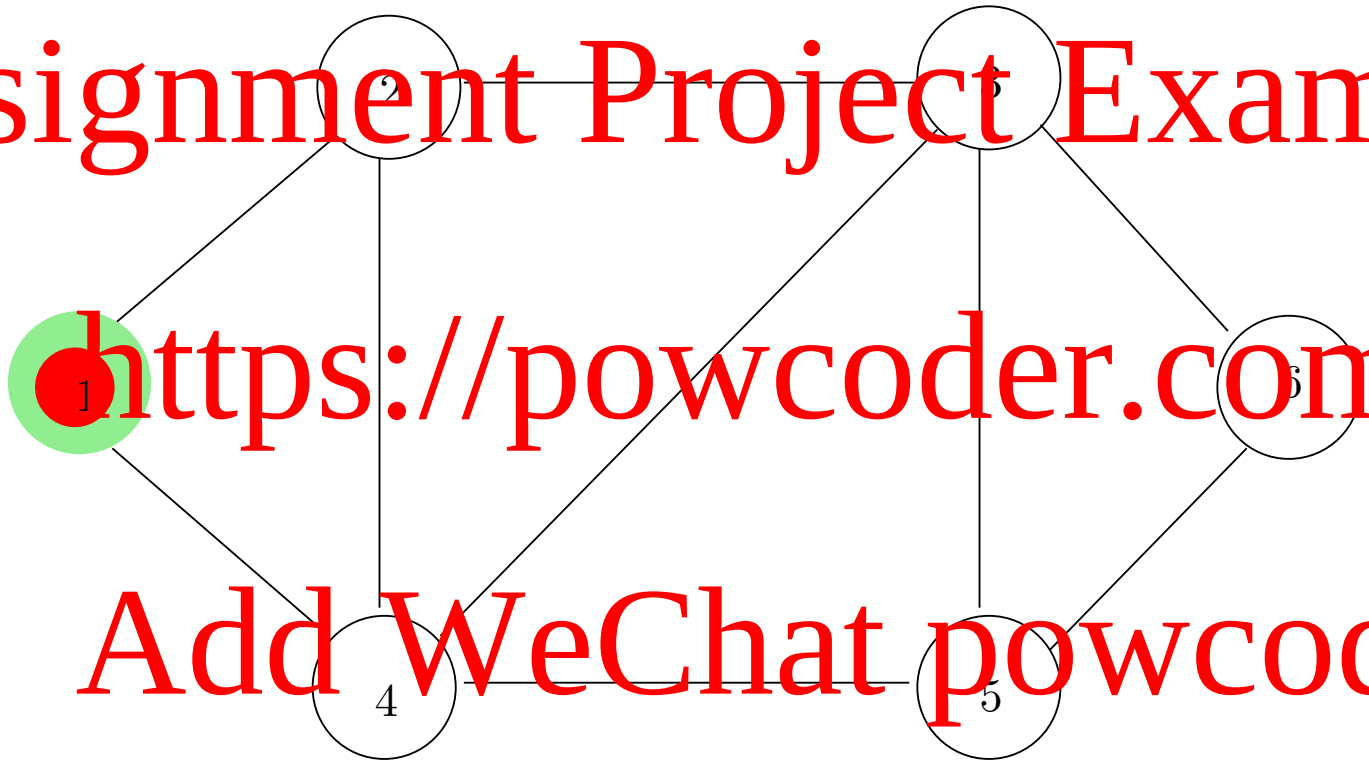
Classical DFS



Time Units = 18 = 2M

Message = 18 = 2M

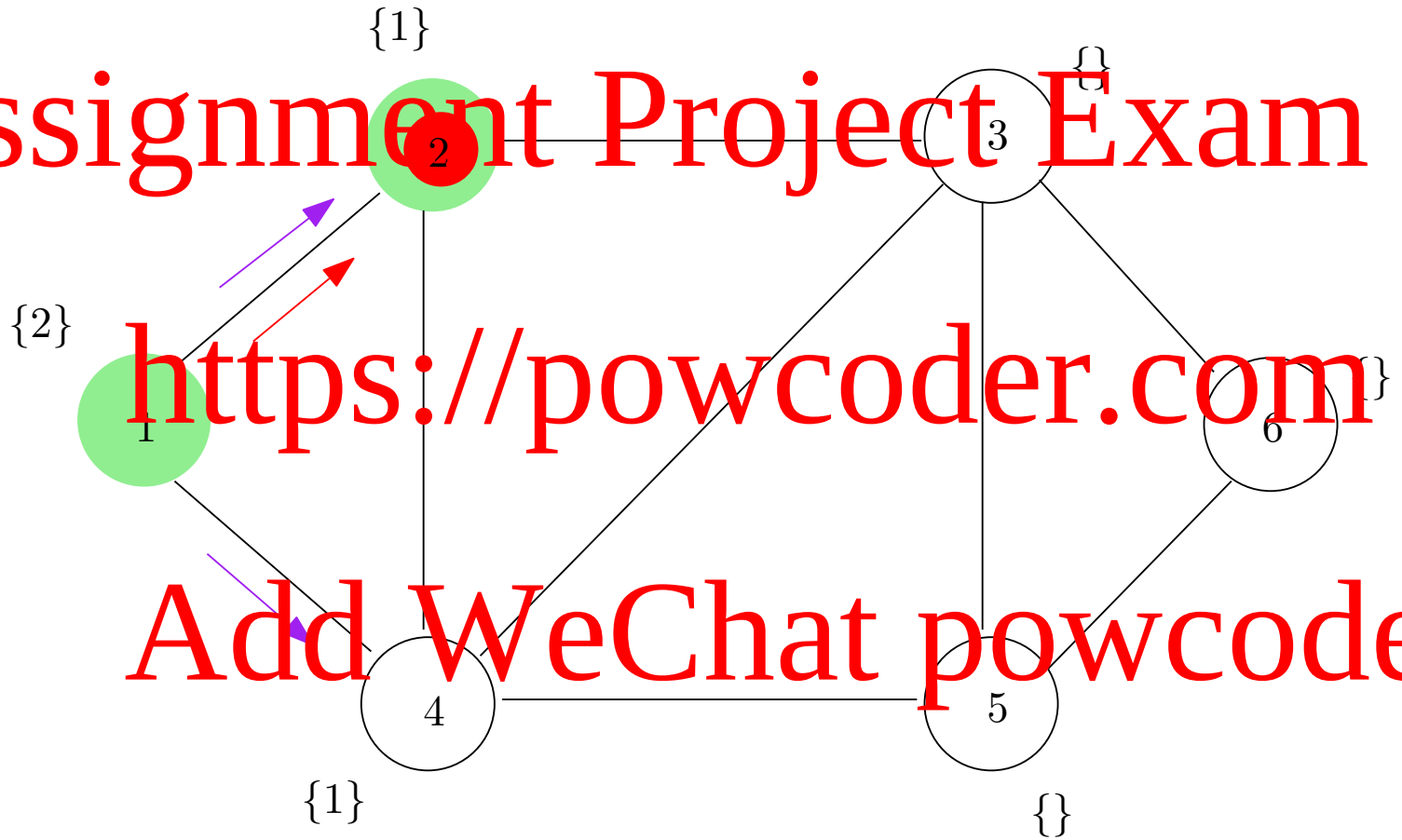
Cidon DFS §1



Time Units = 0

Messages = 0

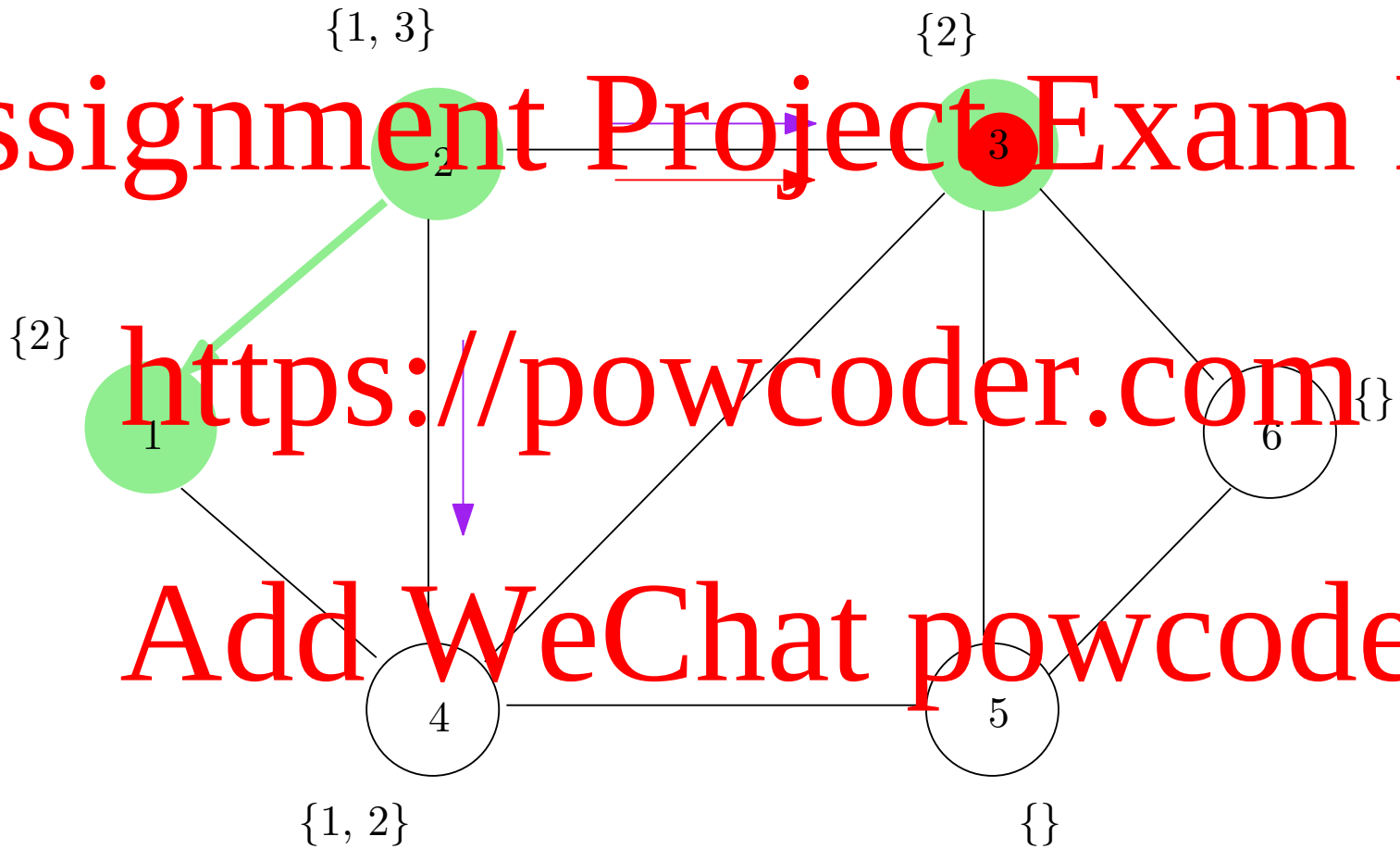
Cidon DFS §1



Time Units = 1

Messages = 3

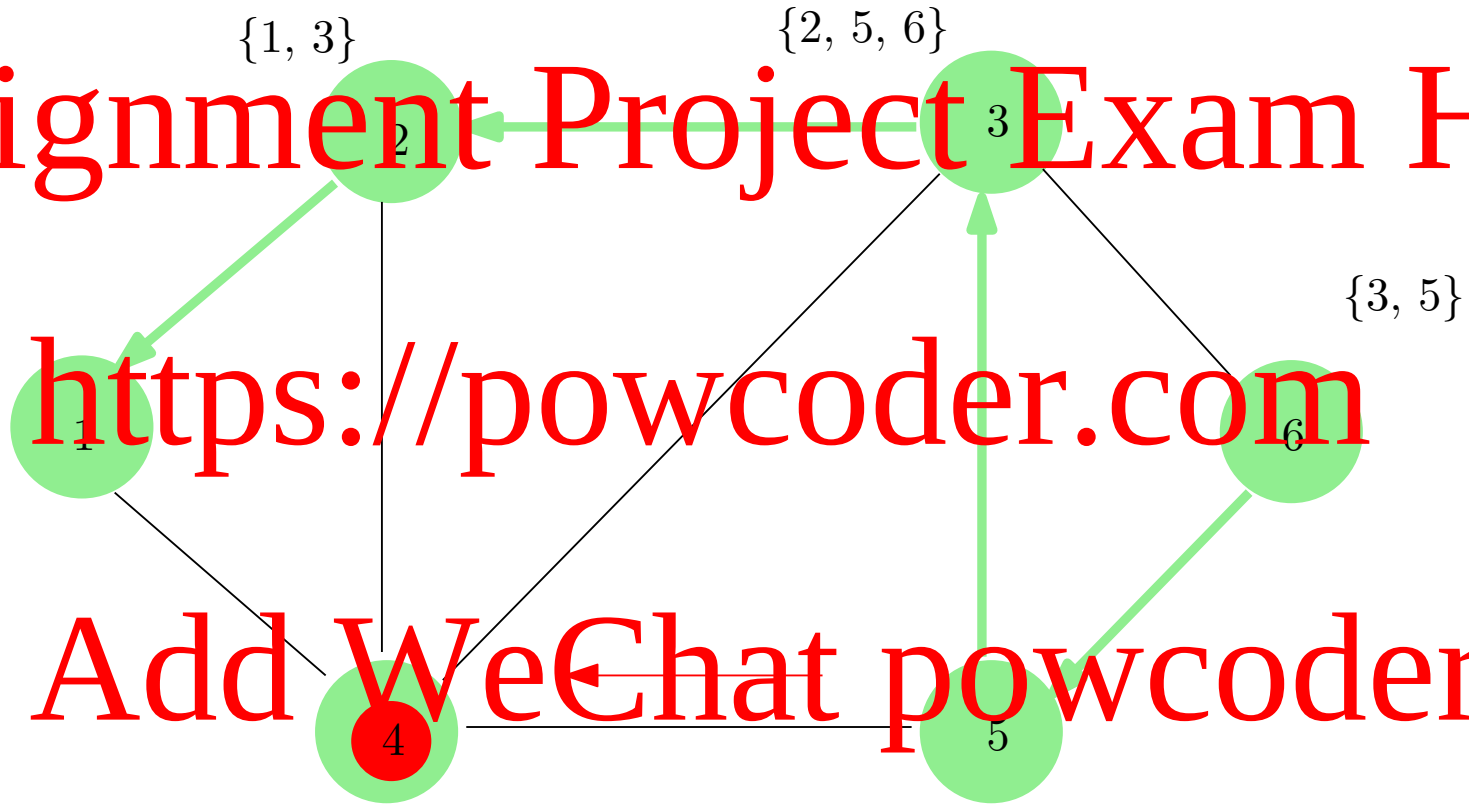
Cidon DFS §1 : 2 → **vis** → 4



Time Units = 2

Messages = 6

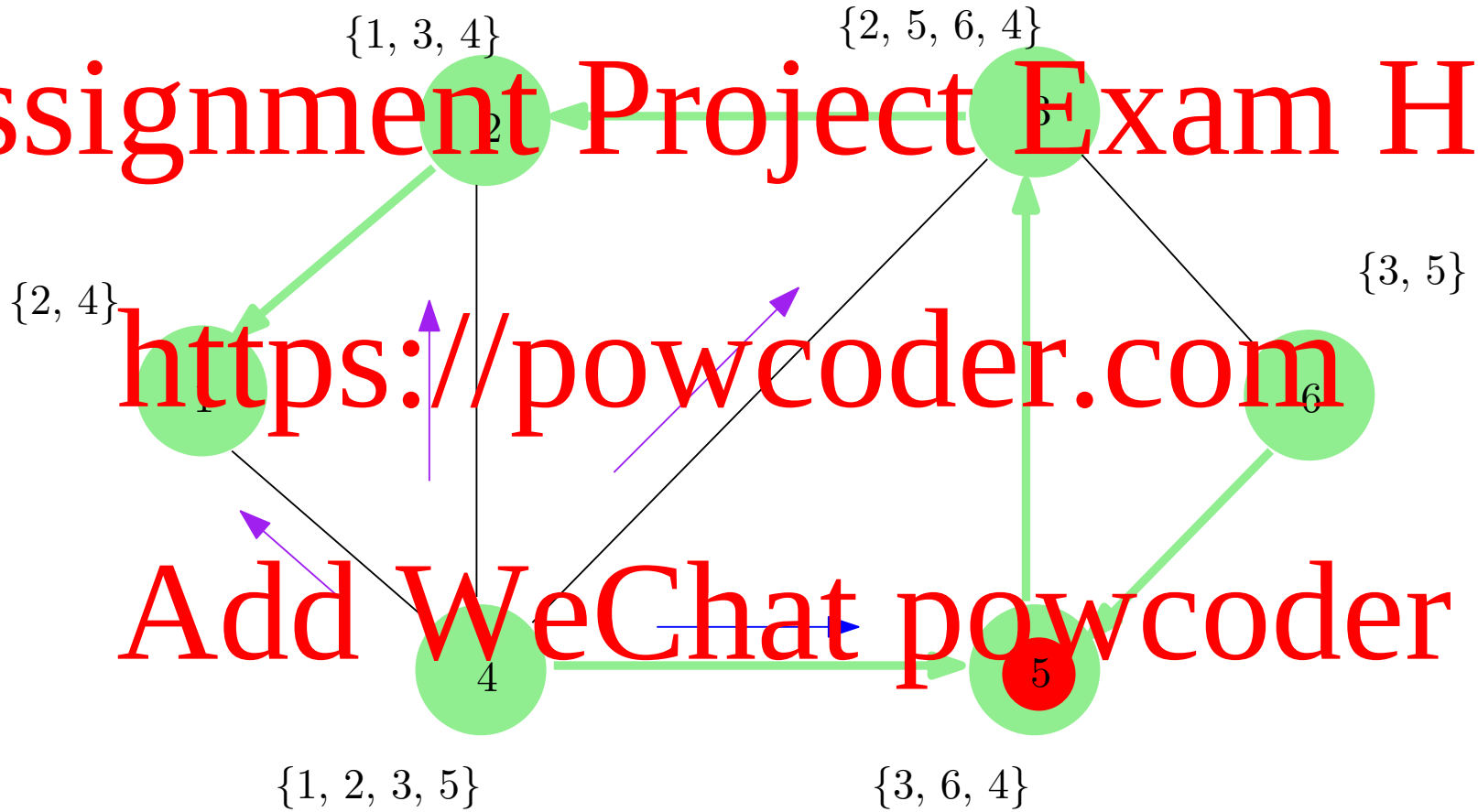
Cidon DFS §1



Time Units = 6

Messages = 16

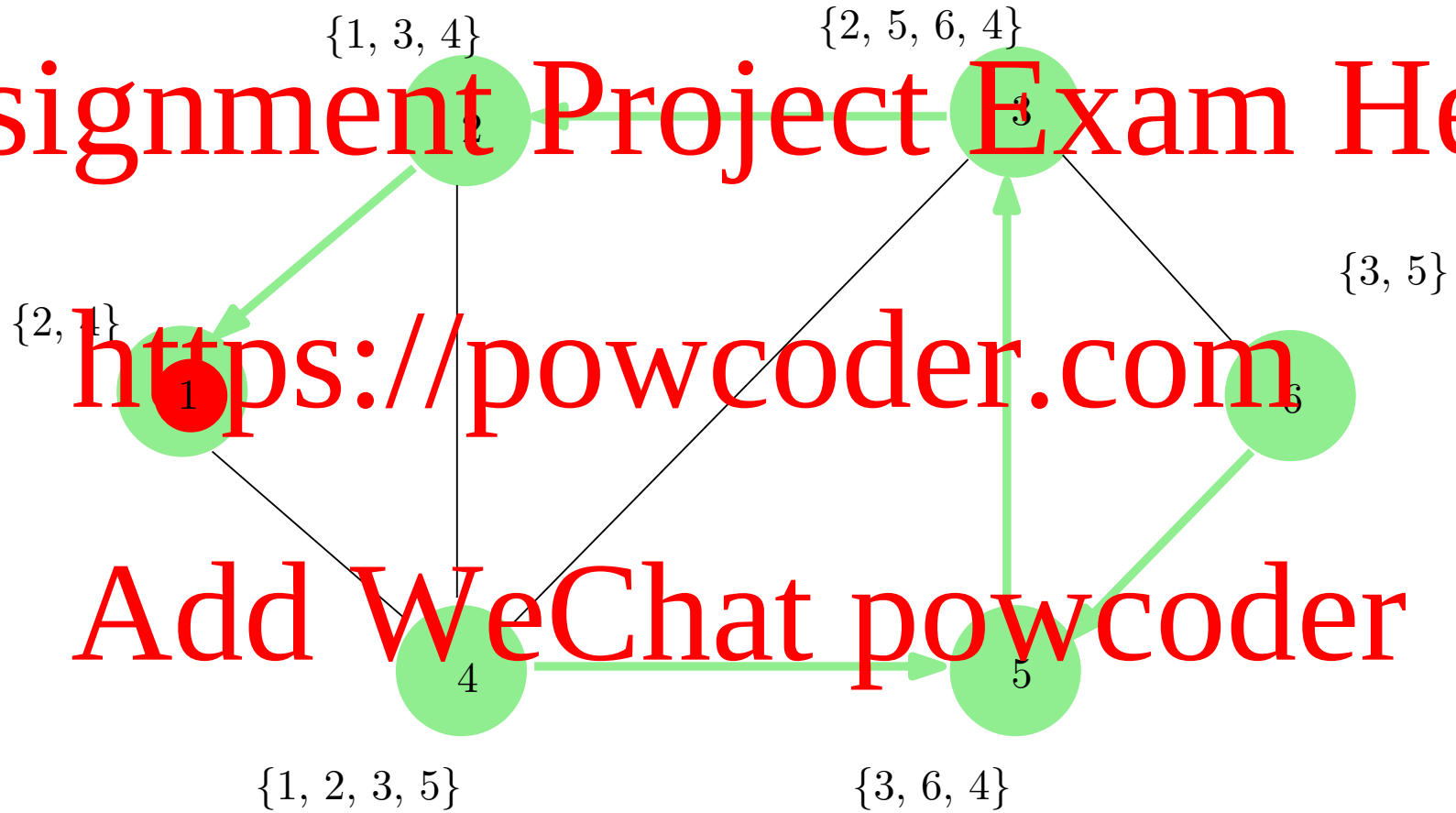
Cidon DFS §1 : 4 → **vis** → 2



Time Units = 7

Messages = 20

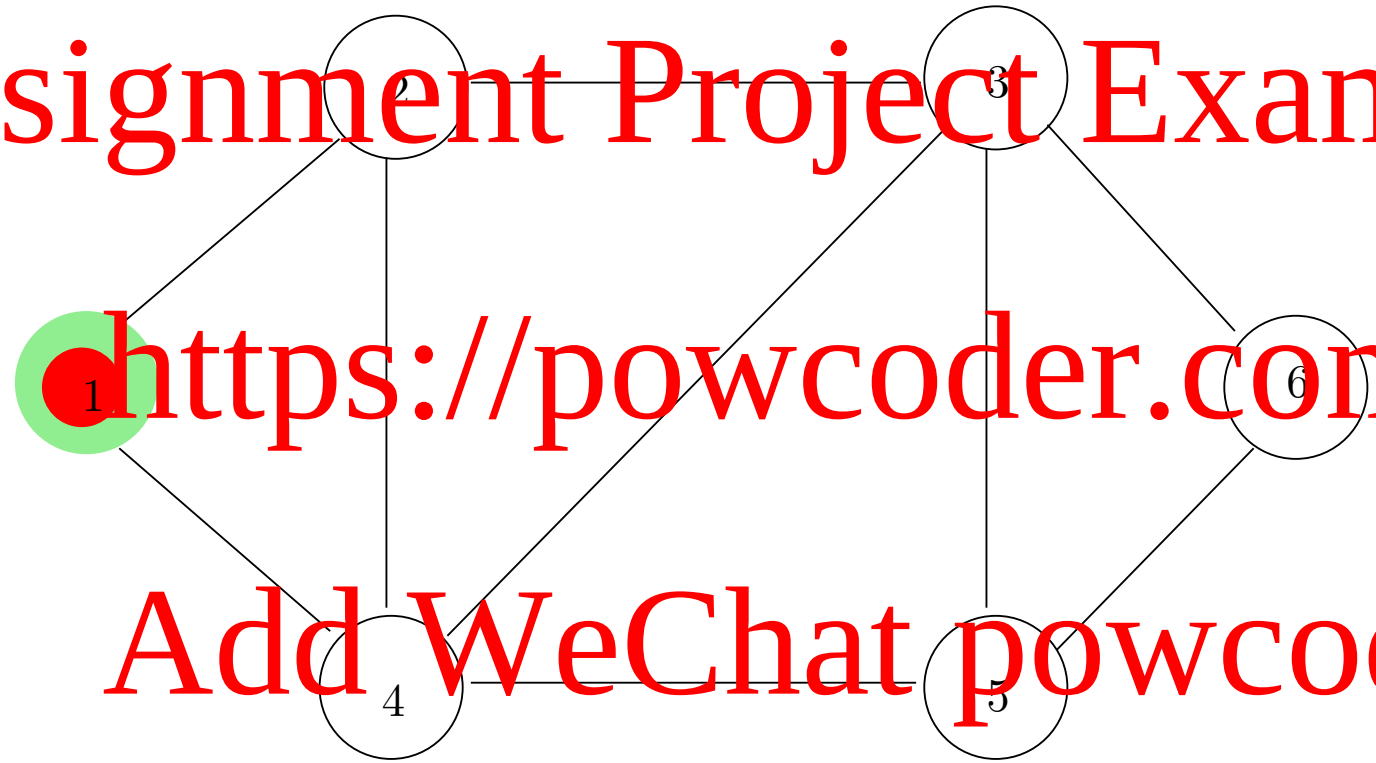
Cidon DFS §1



Time Units = 10 = 2N - 2

Messages = 23 $\leq$ 3M

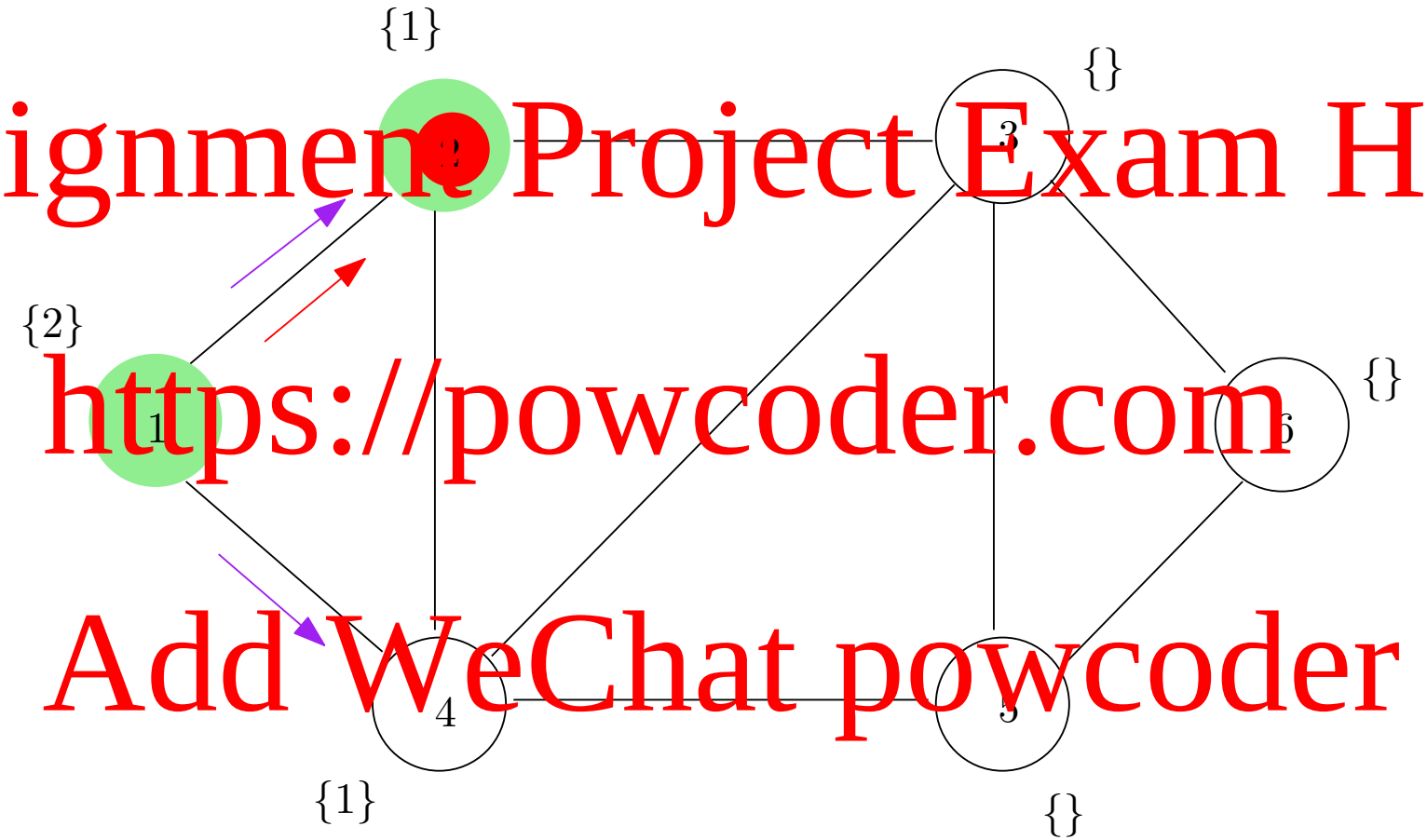
Cidon DFS §2



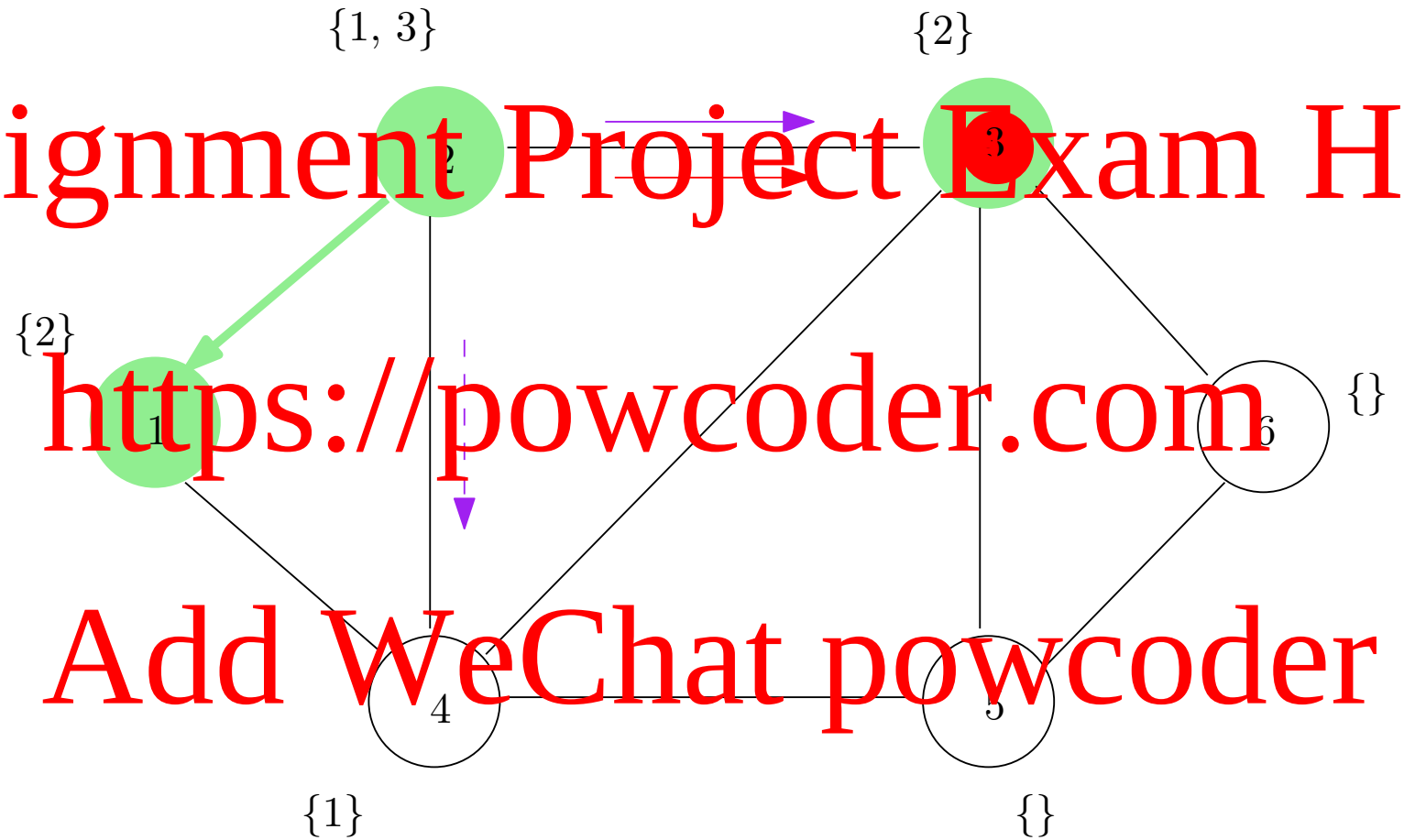
Time Units = 0

Messages = 0

Cidon DFS §2



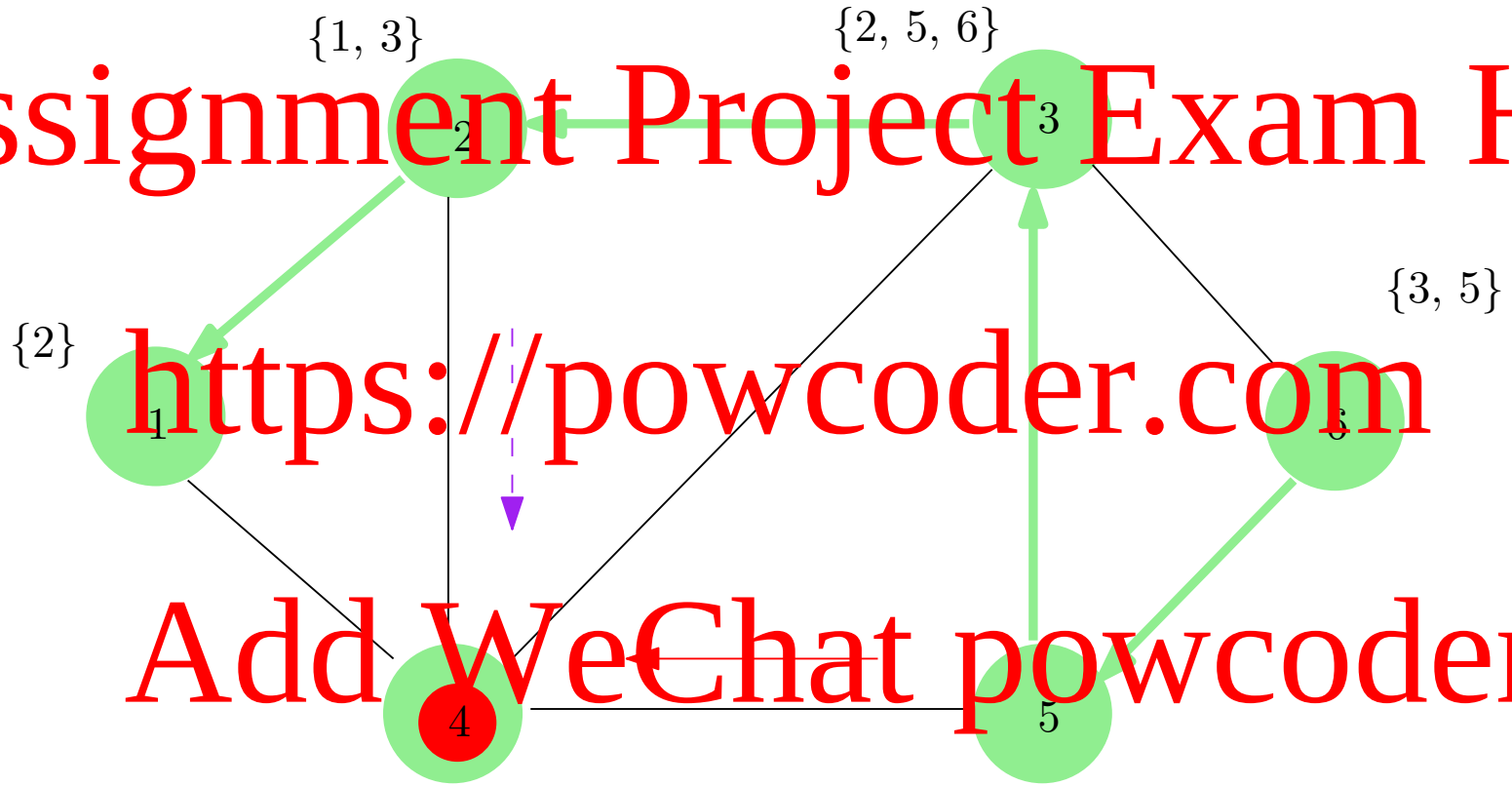
Cidon DFS §2 : 2 −−→ **vis** −−→ 4



Time Units $= 1 + \varepsilon$

Messages $= 6$

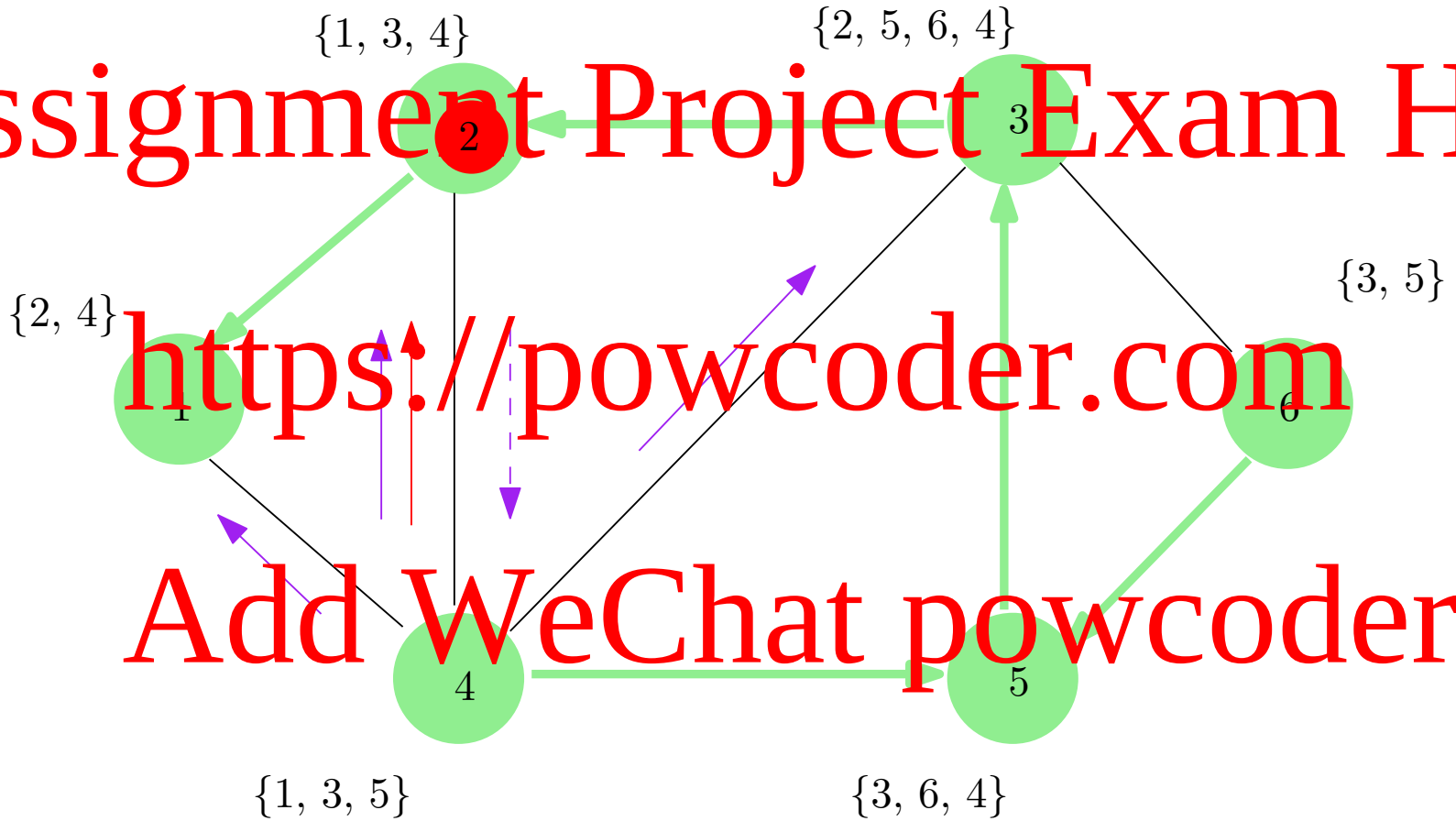
Cidon DFS §2



Time Units $= 1 + 5\varepsilon$

Messages $= 16$

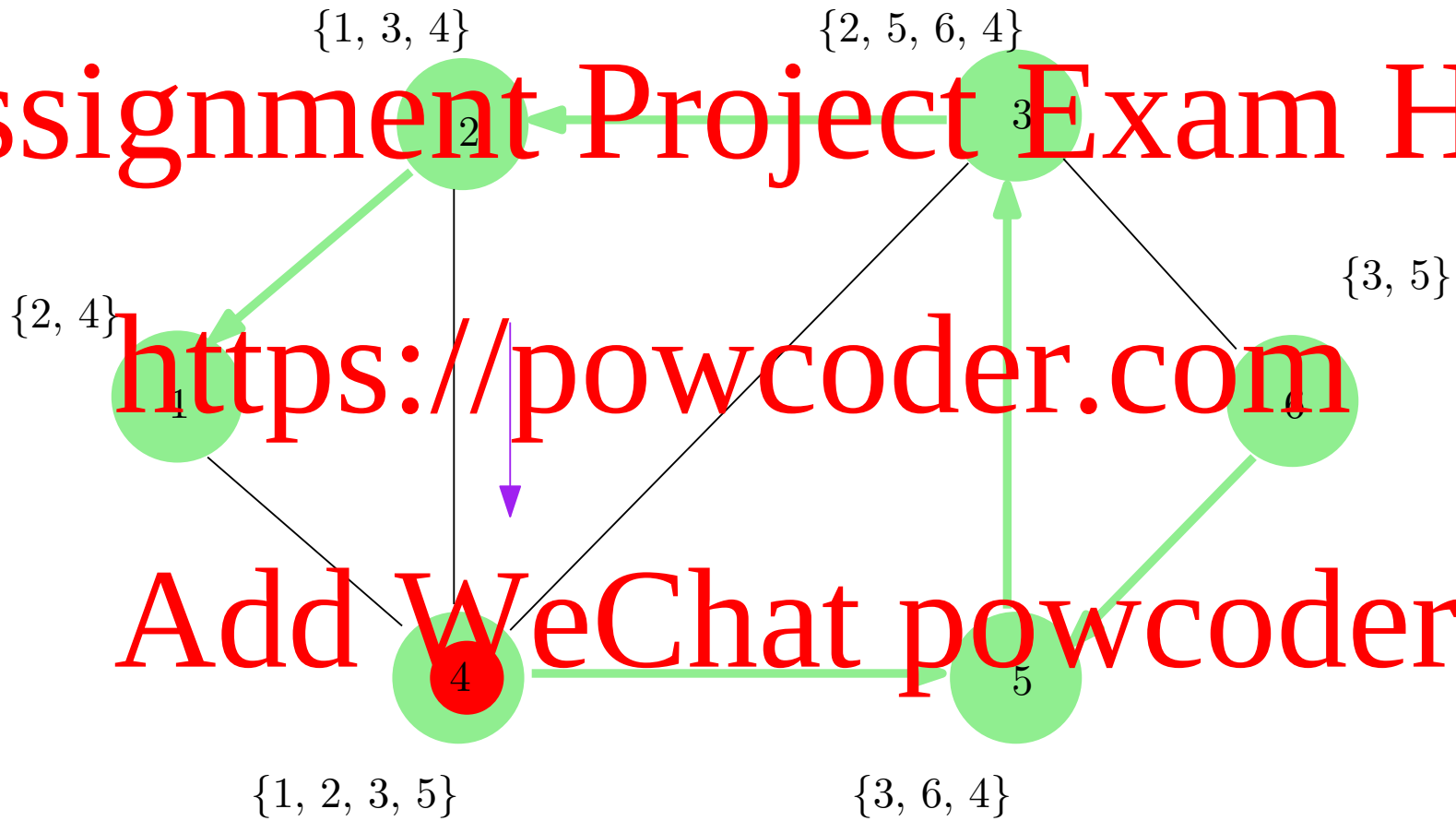
Cidon DFS §2 : 2 ⇢ **vis** ⇢ 4, 4 → **tok+vis** → 2



Time Units $= 1 + 6\varepsilon$

Messages $= 20$

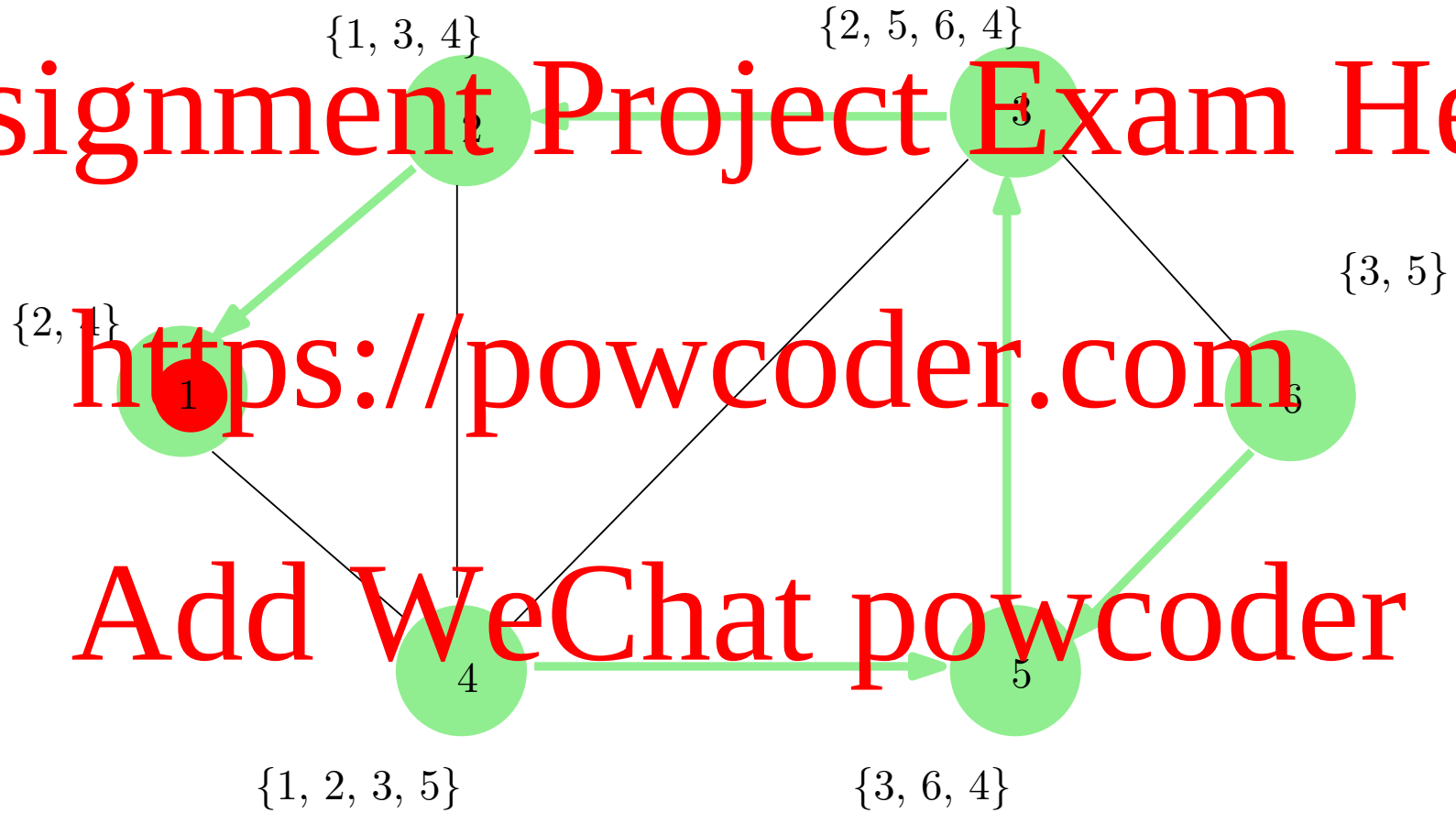
Cidon DFS §2 : 2 → **vis** → 4



Time Units = 2

Messages = 20

Cidon DFS §2



Time Units = 6 $\leq$ 2N - 2

Messages = 24 $\leq$ 3M

Distributed Bellman-Ford algorithm

- Classical Bellman-Ford algorithm finds all shortest paths from a single source – like Dijkstra

- Advantage for Bellman-Ford: can cope with negative weights
- Classical Dijkstra: Time complexity $= O((|E| + |V|) \log |V|)$
- Classical Bellman-Ford: Time complexity $= O(|V||E|)$

- Distributed Dijkstra: more difficult distribution...
- Distributed Bellman-Ford $\approx$ a simple extension of Echo

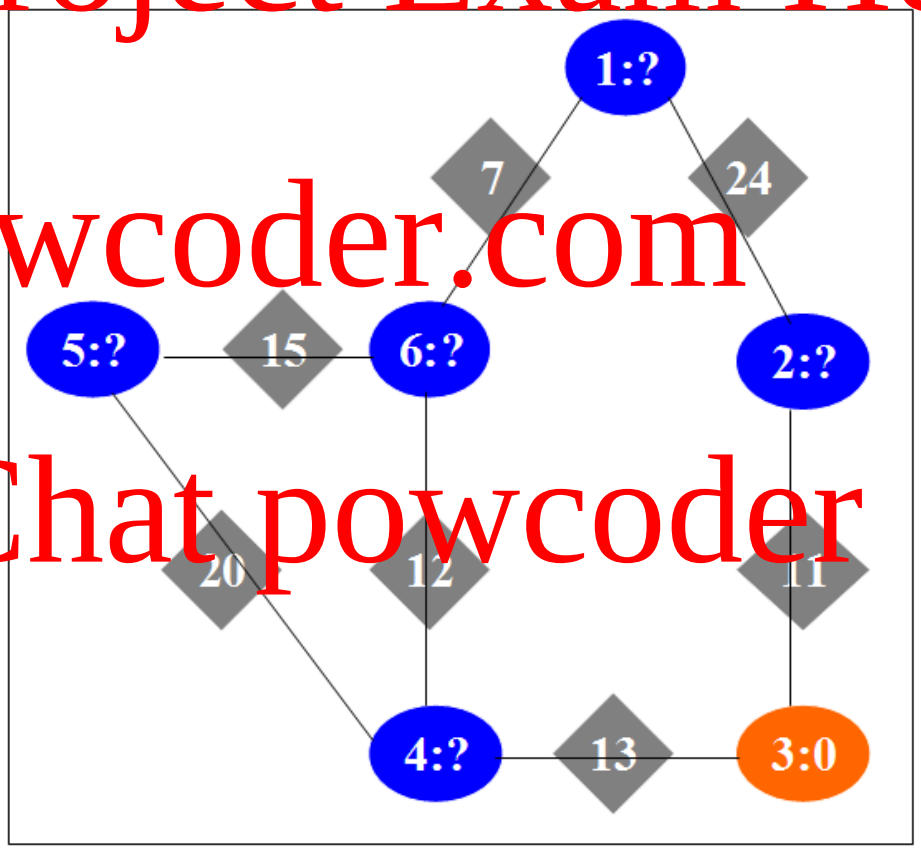
Distributed Bellman-Ford algorithm

- Sync Bellman-Ford ("Echo ++"):
  - Time complexity = $O(|V|)$
  - Message complexity = $O(|V||E|)$
- Async Bellman-Ford:
  - Message complexity: $O(|V|^{|V|})$ (terrible worst case, but often much lower in reality)
  - Time complexity = $O(|V|)$ – if any message is delivered in at most 1 time units – no FIFO [Tel]
  - Time complexity = $O(|V|^{|V|})$ – if we consider the congestion (pileups) on FIFO channels [Lynch]
  - Are these realistic? ...

Sync Bellman-Ford - Start

Problem: Find all shortest paths from node 3.

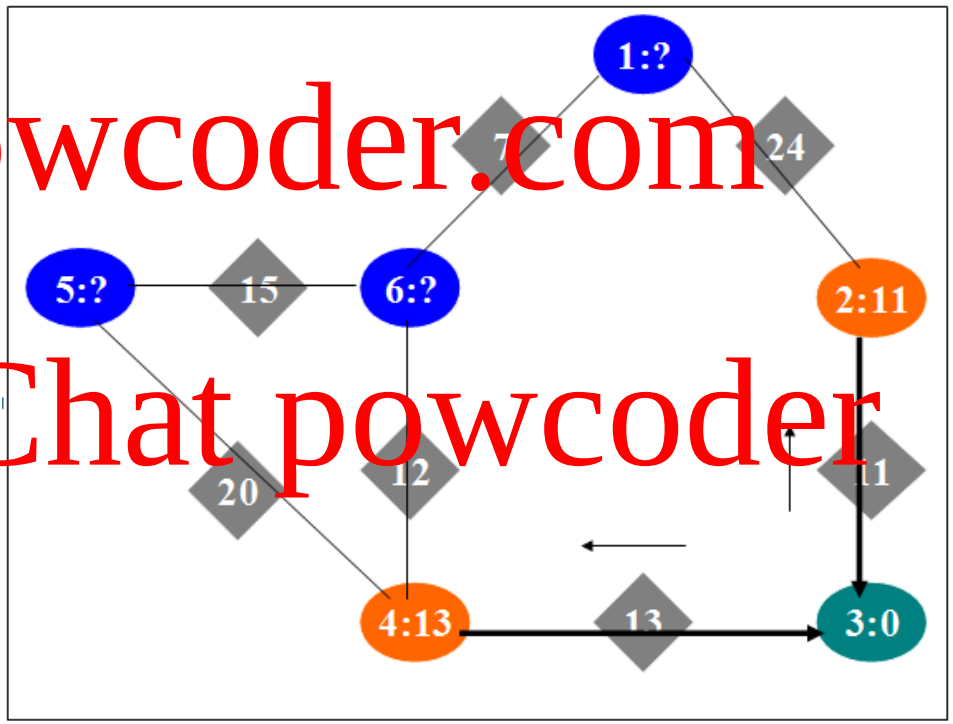
- red = active
- blue = unvisited
- green = visited



Sync Bellman-Ford - Round 1

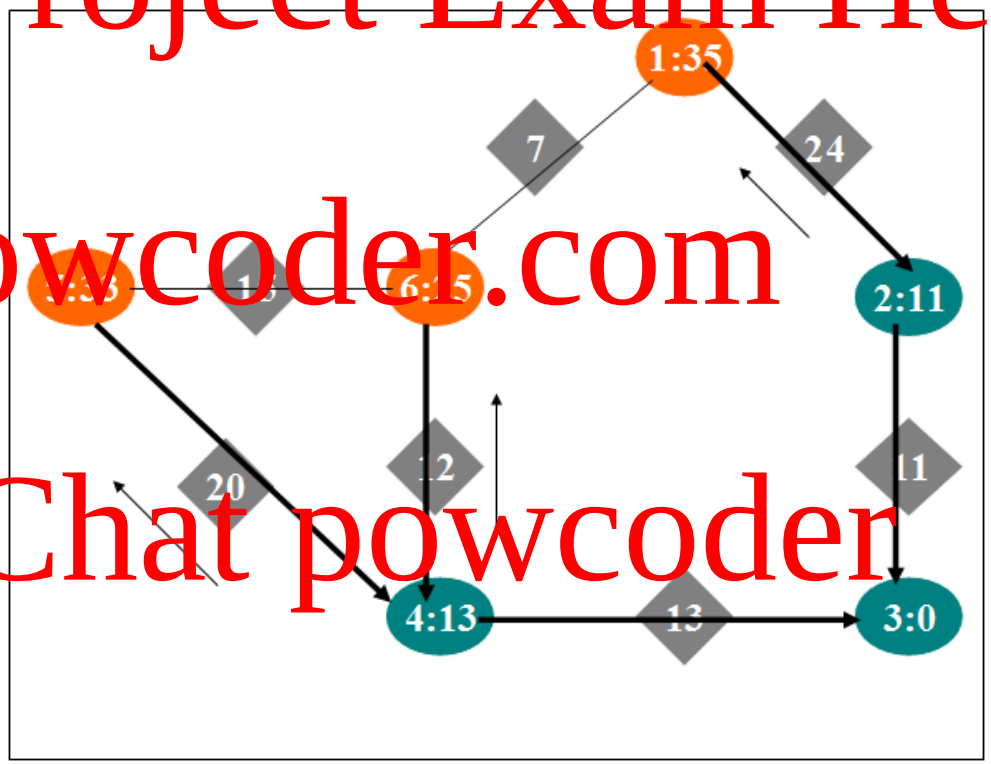
Update formula: *new distance* = minimum between *old distance* and all newly (*received distance* + *edge length*)

- Nodes 4 and 2: update distances



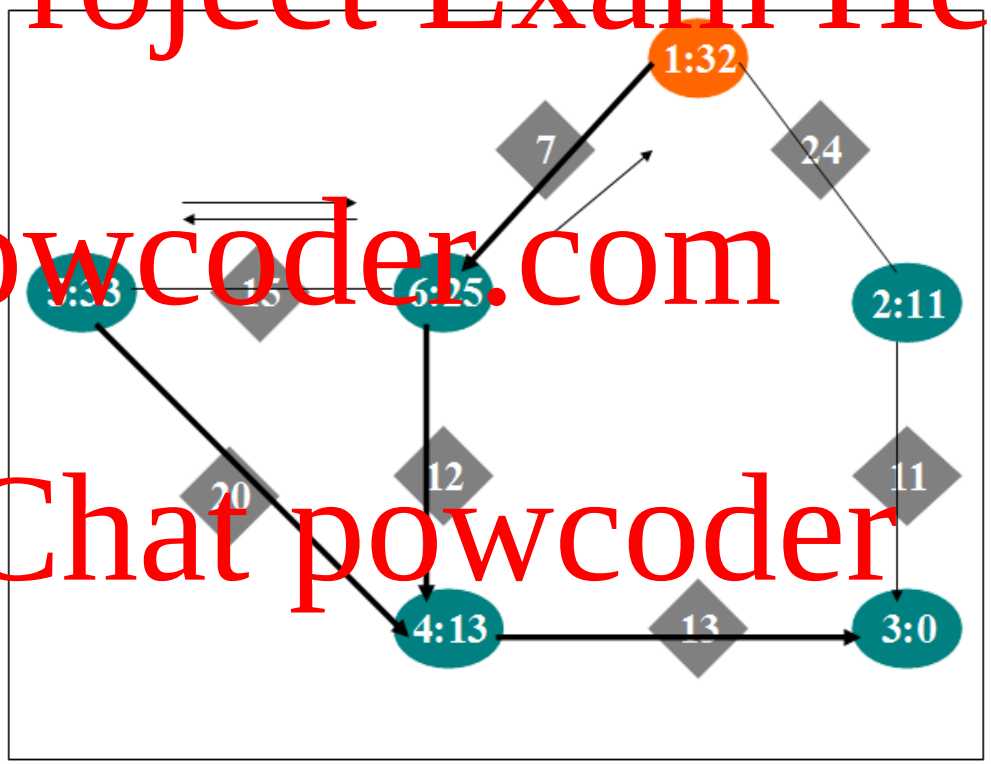
Sync Bellman-Ford - Round 2

- Nodes 5, 6, 1: update distances



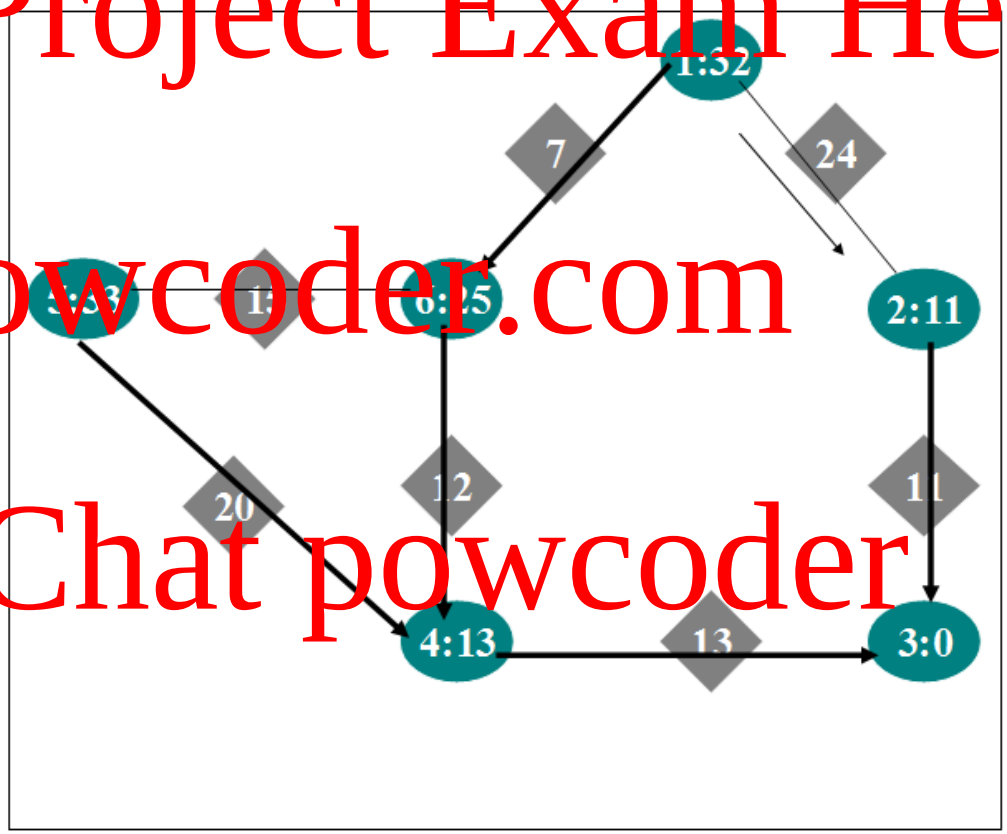
Sync Bellman-Ford - Round 3

- Node 1:
  update distance
  (recalculation!)



Sync Bellman-Ford - Round 4

- Node 2:
  no recalculation (but could have been)



Distributed Bellman-Ford – termination?

All nodes have successfully terminated

We can see this, but the nodes don't know this yet

How to detect and, optionally, disseminate the termination info?

- For Sync and Async Bellmann-Ford: by convergecast and, optionally, a subsequent broadcast (like Echo)
- For Sync Bellmann-Ford: by attaching a time-to-live (TTL) to the broadcast token
  - Initially equal to the number of nodes $|V|$ (if this is known)
    - After receiving it, each node decrements this by 1, at each round (thus sync mode required)
    - When TTL $= 0$, each node knows that the algorithm has terminated (guaranteed)

Async Bellman-Ford – worst case (sketch, cf. Lynch §15.4)

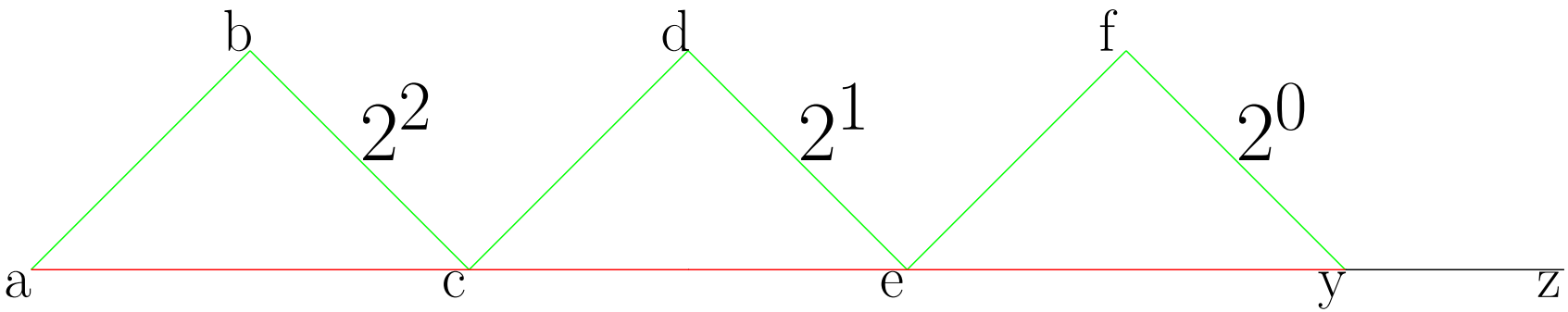
Sketch for

- $k = 3$ (can be any number)
- $N = 2k + 2 = 8$ nodes
- $M = 3k + 1 = 10$ edges

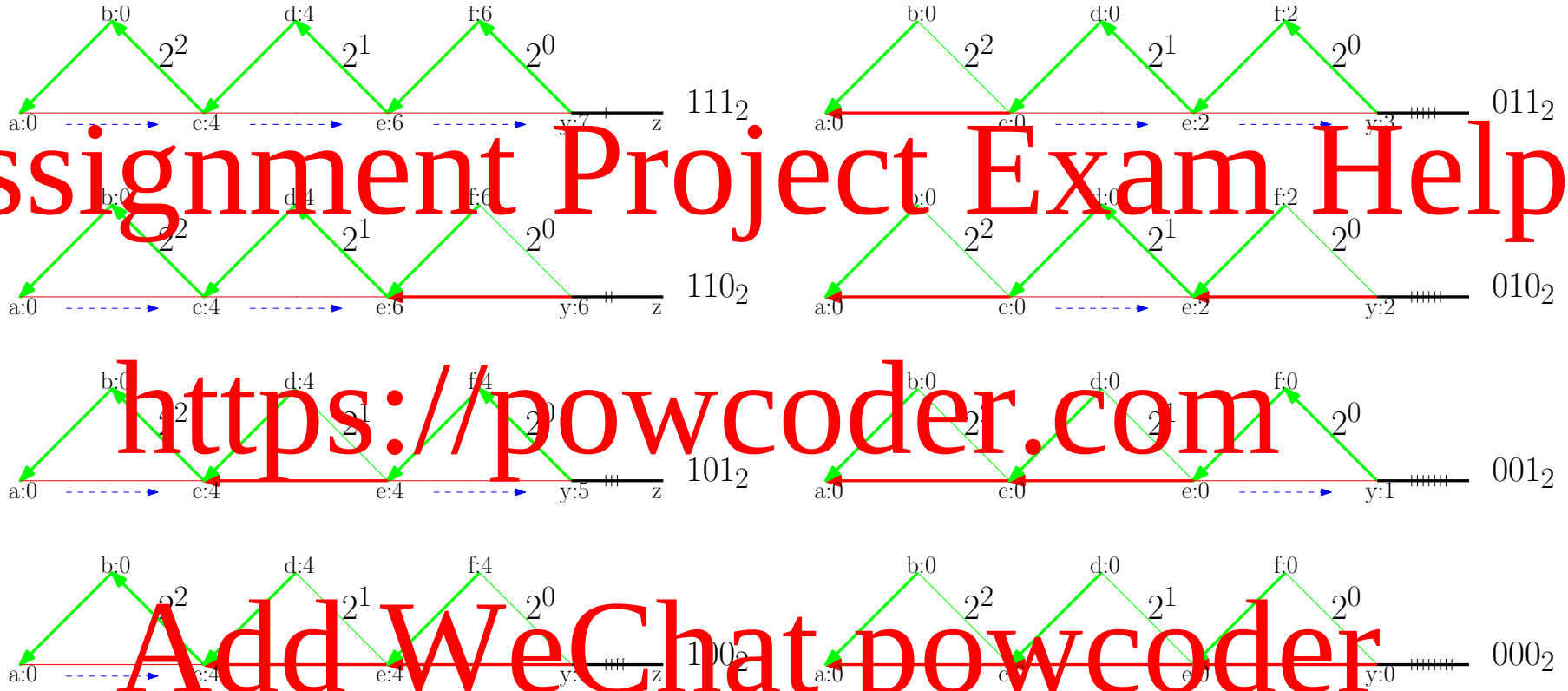
Costs as indicated or 0 (default)

Initiator: left-most node (a)

Green: fast link; Red: slow link; Black: slow & critical if FIFO



Async Bellman-Ford – worst case (cont)



Dotted blue arrows : messages still in transit

Shapes of the shortest-so-far cost paths $\overset{1:1}{\longleftrightarrow}$ base 2 numbers

Exponential message complexity $\geq 2^k = 2^{(N-2)/2} = \Omega(\sqrt{2}^N)$

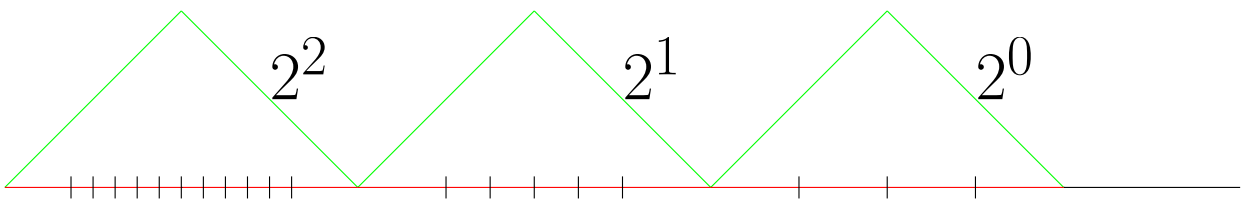
Exponential time complexity if FIFO – congestion on black edge

Async Bellman-Ford – worst case

- How to explain the time complexity?
- Time complexity without FIFO pileups $= O(|V|)$?
- Time complexity with FIFO pileups = exponential?
- If the last edge, before the rightmost node, is slow enough, say t, then all 2^k messages can pile-up in this segment
- If we consider congestion, then these piled-up messages can be successively delivered at t intervals
- Total delivery time on the last edge: $2^k t = 2^{O(N)}$ time units, i.e. exponential time complexity [Lynch §15.4]
- This argument fails, if we do NOT consider FIFO congestion, because then even the slowest message will not be affected by the others, and will only take a maximum 1 time unit.

Echo and Bellman-Ford – complexity highlights

- Sync Echo (aka Sync BFS) : BFS ST, no link changes, fast
- Async Echo : arbitrary ST, no link changes, but not so fast
- Sync BF : shortest paths ST, many link changes, not so fast
- Async BF : shortest paths ST, exponential link changes
  - if FIFO : worst case exponential time
- Why the worst case argument does NOT apply to Sync BF?
  - emulating slow links by extra edges exponentially increases V!
  - the formula is still exponential on k but not any more on $N = exp(k)$!



Maximal Independent Set

- Independent = no two neighbours
- Maximal = cannot be extended
  - Maximal is not necessarily largest

- Here we assume that nodes do not have IDs
- Impossible to solve with conventional means in an absolutely symmetric case!

- Luby's algorithm can still break the ties with randomization techniques

Luby's algorithm

- Sync algorithm, which works in stages, each consisting of the following 3 rounds

1. Each processes chooses a new random value and sends it to its neighbours. Processes that have values greater than all their neighbours become winners
2. Winners notify their neighbours. Processes that receive such messages from their neighbours become losers

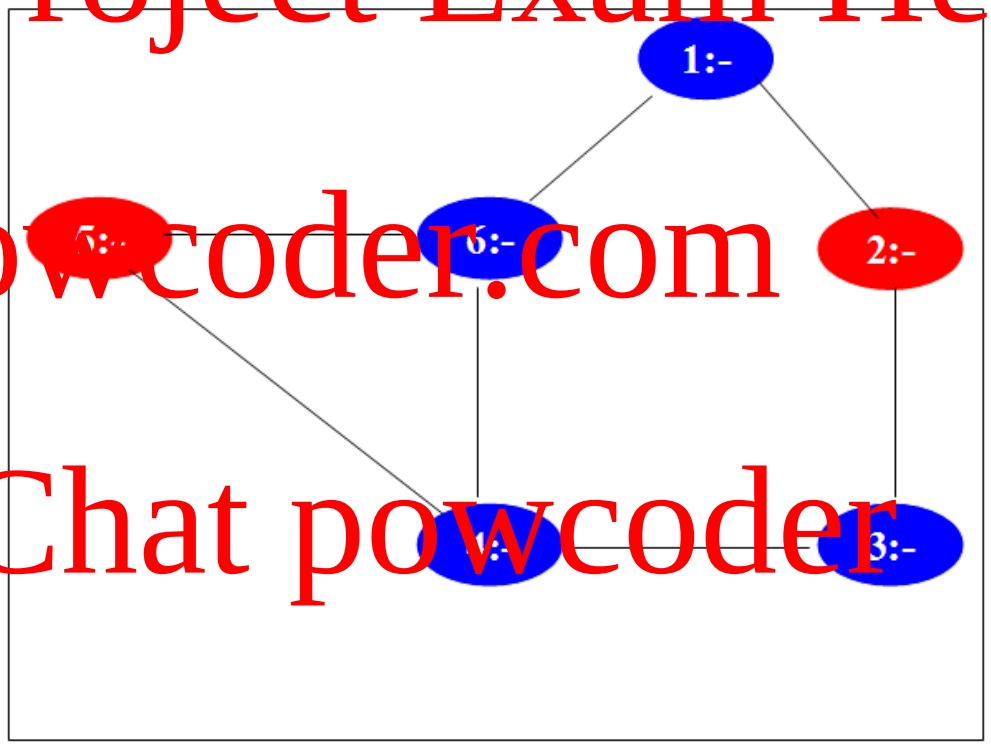
3. Losers notify their neighbours. All processes make a conceptual reshaping of the graph. Both winners and losers are conceptually disconnected from further participation. Remaining nodes are still competing and will regenerate new random values in their next stage!

Luby's algorithm

- Luby's algorithm will stop with probability 1, expected $O(\log n)$ rounds -– proof quite hard
- The random values are best chosen in the range $1, 2, \ldots N^4$, will be likely distinct (but this is not necessary)
- Our diagrams will be sketched only, we won't detail all these rounds individually

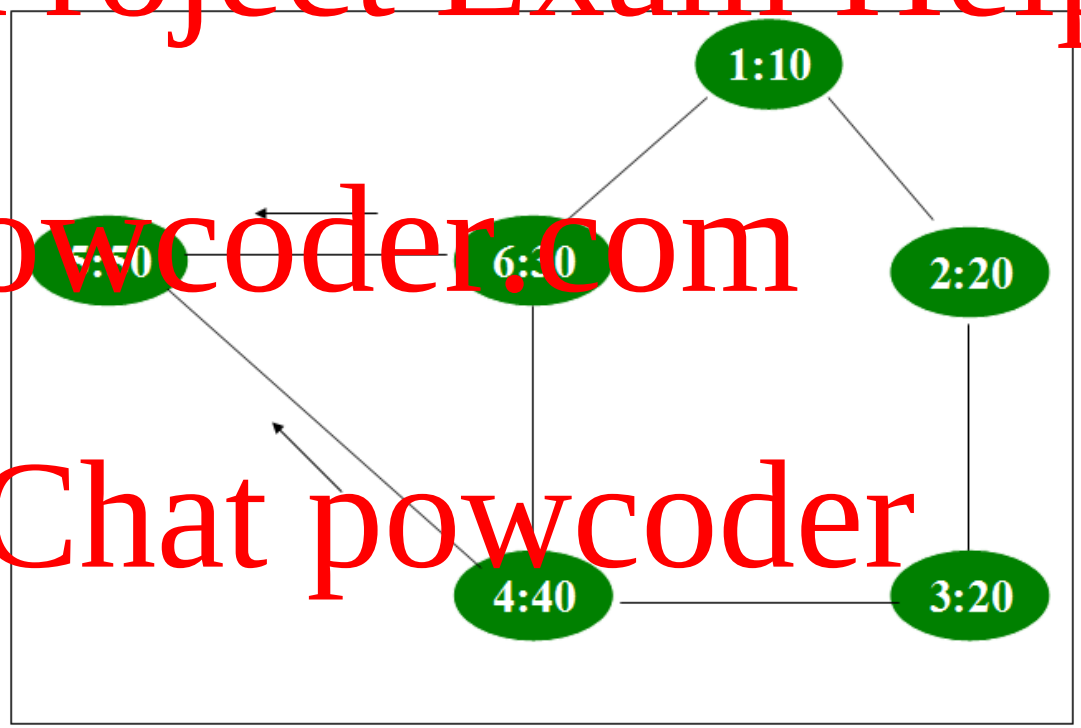
MIS – Example

- One possible solution, formed by $\{5, 2\}$
- Another solution could be $\{5, 1, 3\}$ (which is larger)



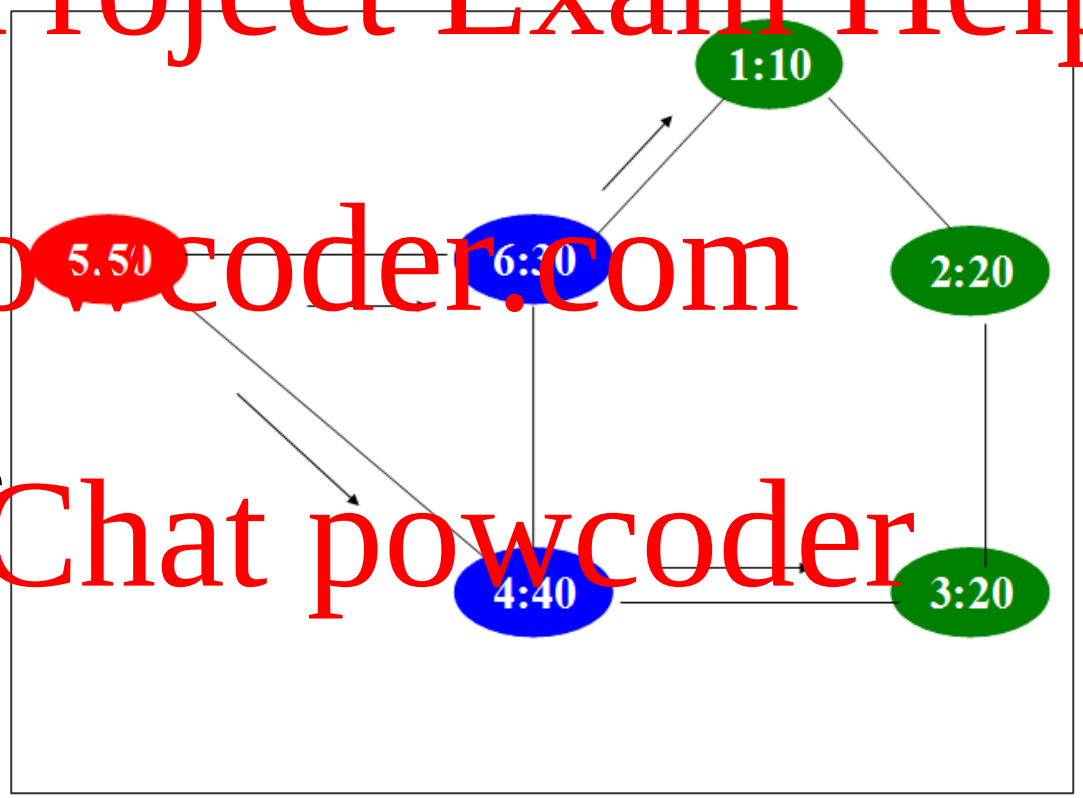
Luby – Stage #1, Round #1

- green = hopeful
- blue = loser
- red = winner
- we only show most relevant messages



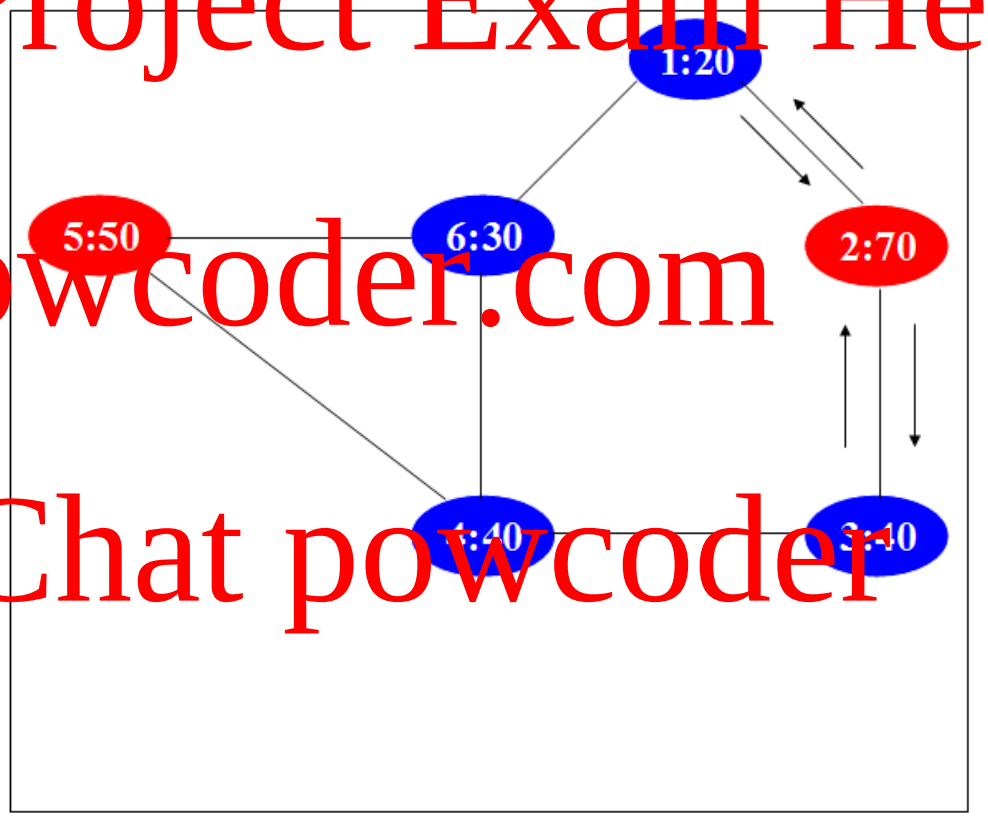
Luby – Stage #1, Rounds #2&#3

- one winner so far: 5
- losers: 4 and 6
- losers notify their neighbours
- winners and losers are conceptually disconnected
- still competing: 1, 2 and 3



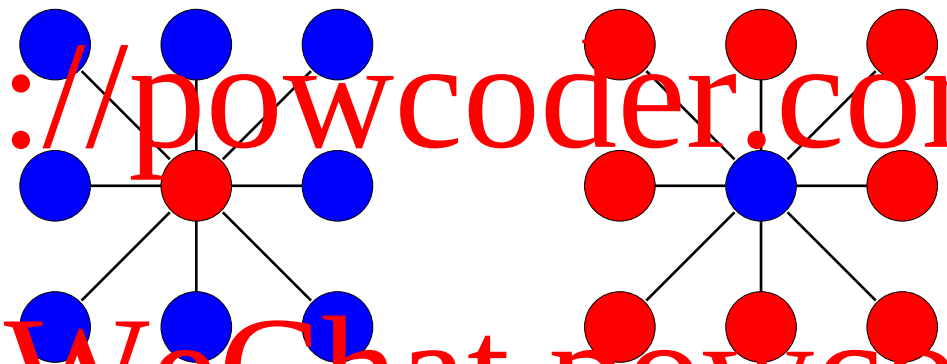
Luby – Stage #2

- a new winner: 2
- new losers, this winners' neighbours: 1 and 3
- the end



More about MIS

- Minimum Maximal Independent Set vs. Maximum Maximal Independent Set



- Related readings (NOT required) – S. Butenko, PhD Thesis `https://ufdc.ufl.edu/UFE0001011/00001/pdf`