Università degli Studi dell'Aquila

Academic Year 2016/2017

Course: Distributed Systems (6 CFU, integrated within the NEDAS curriculum with "Web Algorithms" (6 CFU), by Prof. Michele Flammini)

Instructor: Prof. Guido Proietti

Schedule:	Tuesday:	14.30 – 16.15 – Room A1.2
	Thursday:	14.30 – 16.15 – Room A1.1
Questions?:	Tuesday	16.30 - 18.30 (or send an email to guido.proietti@univaq.it)

Slides plus other infos:

http://www.di.univaq.it/~proietti/didattica.html

Distributed Systems (DS)

In the old days: a number of workstations over a LAN

Today

Collaborative Computing Systems

- Military command and control
- Online strategy games
- Massive computation

Distributed Real-time Systems

- Process Control
- Navigation systems, Airline Traffic Monitoring (ATM)

Mobile Ad hoc Networks

Rescue Operations, emergency operations, robotics

Wireless Sensor Networks

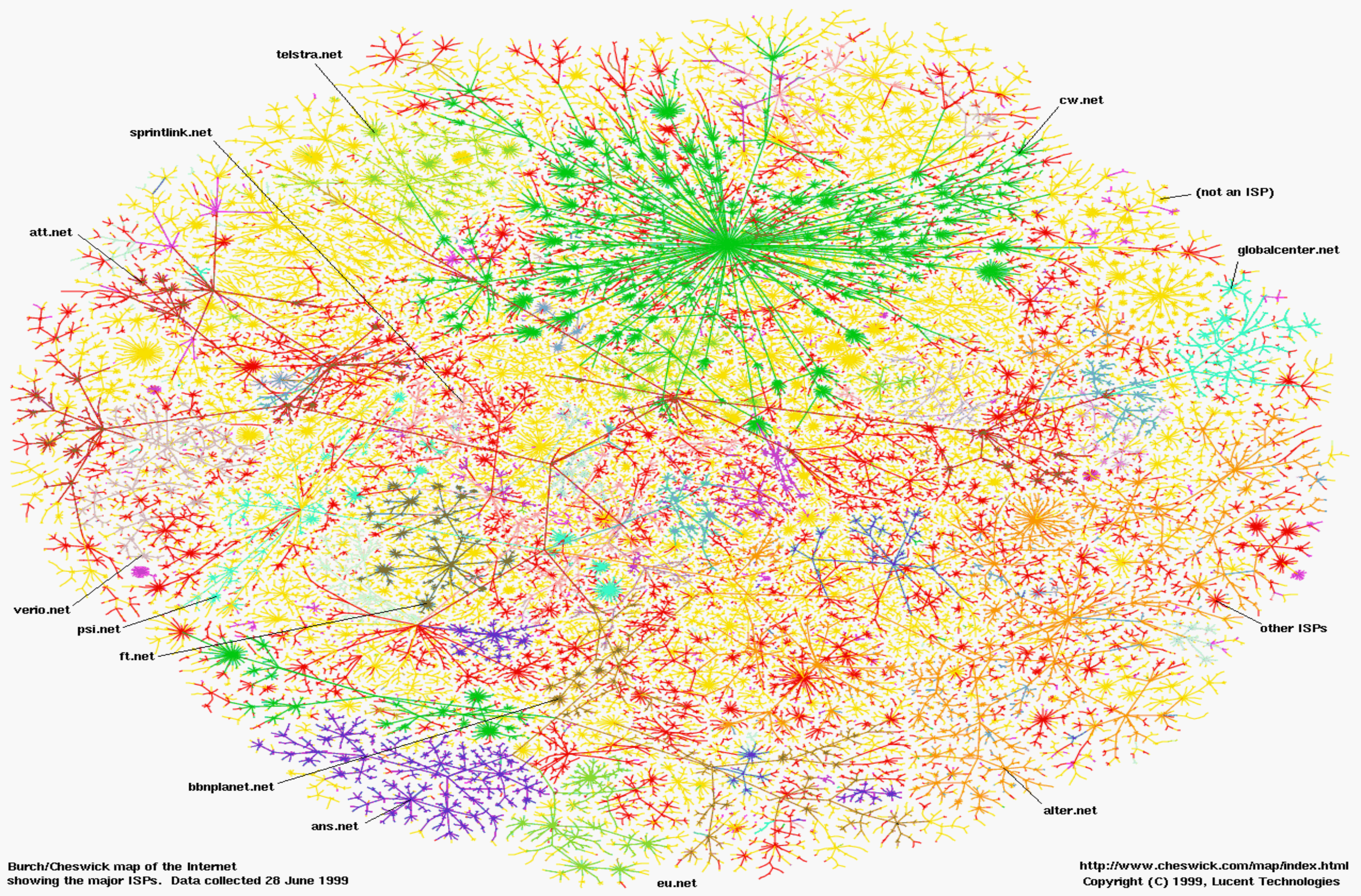
Habitat monitoring, intelligent farming

Social Networks

Grid and Cloud computing

…

And then, the mother of all DS: the Internet



Two main ingredients in the course: Distributed Systems + Algorithms

Distributed system (DS): Broadly speaking, we refer to a set of **autonomous computational devices** (say, **processors**) performing multiple operations/tasks simultaneously, and which influence reciprocally either by **taking actions** or by **exchanging messages** (using an underlying wired/wireless **communication network**)

We will be concerned with the **computational aspects** of a DS. We will analyze a DS depending on the **behaviour** of its processors:

- Obedient: always cooperate **honestly** with the system
  - $\Rightarrow$ Classic field of **distributed computing**
- Adversarial: may operate **against** the system
  - $\Rightarrow$ Classic field of **fault-tolerance** in DS

The emerging field of **game-theoretic aspects of DS**, where processors behave **strategically** in order to maximize her personal welfare will be studied next year in the class of Autonomous Networks

Two main ingredients in the course: Distributed Systems + Algorithms (2)

Algorithm (informal definition): effective method, expressed as a finite list of well-defined instructions, for solving a given problem (e.g., calculating a function, implementing a goal, reaching a benefit, etc.)

The actions performed by each processor in a DS are dictated by a local algorithm, and the global behavior of a DS is given by the "composition" (i.e., interaction) of these local algorithms

We will analyze these distributed algorithms in (almost) every respect: existence, correctness, finiteness, efficiency (computational complexity), effectiveness, robustness (w.r.t. to a given fault-tolerance concept), etc.

Course structure

FIRST PART: Algorithms for COOPERATIVE DS

1. Leader Election
2. Minimum spanning tree
3. Maximal independent set

SECOND PART: Algorithms for UNRELIABLE DS

1. Benign failures: consensus problem
2. Byzantine failures: consensus problem
3. Failure monitoring

THIRD PART: CONCURRENT DS: Mutual exclusion

FOURTH PART (to be confirmed): SEMINARS: each student or group of students will read a paper and present it in the classroom

Mid-term **Written** Examination (week 7-11 of November): 10 multiple-choice tests, plus an open-answer question

Final **Oral** Examination: this will be concerned with either the whole program or just the second part of it, depending on the outcome of the mid-term exam. There will be fixed a total of 6 dates, namely:

- 3 in January-February
- 2 in June-July
- 1 in September

For those enrolled in the NEDAS curriculum, there will be a single final grade as a result of the grades obtained in this course and in the "Web Algorithms" course; the corresponding exams can be done separately, but they must be sustained within the same calendar year

Cooperative DS: Message Passing System

A Formal Model

The System

- Topology: a network (connected undirected graph)
  - Processors (nodes)
  - Communication channels (edges)
- Degree of Synchrony: *asynchronous* versus *synchronous* (**universal clock**)
- Degree of Symmetry: *anonymous* (**processors are indistinguishable**) versus *non-anonymous*: this is a very tricky point, which refers to whether a processor has a distinct ID which can be used during a computation; as we will see, there is a drastic difference in the powerful of a DS, depending on this assumption
- Degree of Uniformity: *uniform* (**number of processors is unknown**) versus *non-uniform*

Local *versus* distributed algorithm

Local algorithm: the algorithm associated with each single processor

Distributed algorithm: the "composition" (i.e., interaction) of local algorithms

General assumption: local algorithms are all the same (so-called homogenous setting), otherwise we could force the DS to behave as we want by just mapping different algorithms to different processors, which is unfeasible in reality!

Notation

- ✓ n **processors**: $p_0, p_1, \ldots, p_{n-1}$.
- ✓ Each processor has a consistent knowledge of its neighbors, numbered from 1 to r
- ✓ Depending on the context, a processor (or more precisely, its algorithm) may make use of global information about the network, e.g., the size, the topology, etc.
- ✓ **Communication** of each processor takes place only through **message exchanges**, using buffers associated with each neighbor, say OUT_BUFFER_i and IN_BUFFER_i, for each neighbor i=1,…,r.
- ✓ Q_i: the **state set** for p_i, containing a distinguished initial state; each state describes the internal status of the processor and the status of the buffers

Configuration and events

- ✓ **System configuration:** A vector $[q_0, q_1, \ldots, q_{n-1}]$ where $q_i \in Q_i$ is the state of p_i

- ✓ **Events:** Computation events (internal computations plus sending of messages), and message delivering (receipt of messages) events

Execution

$C_0\ \phi_1\ C_1\ \phi_2\ C_2\ \phi_3\ \ldots$ where

- ✓ C_0 : The initial configuration
- ✓ C_i : A configuration
- ✓ ϕ_i : An event

Asynchronous Systems

- ✓ No **upper bound** on delivering times
- ✓ **Admissible** execution: each message sent is eventually delivered

Synchronous Systems

- ✓ Each processor has a (universal) clock, and computation takes place in **rounds**.
- ✓ At each round each processor:
  1. Reads the incoming messages buffer
  2. Makes some internal computations
  3. Sends messages which will be read in the next round.

Message Complexity

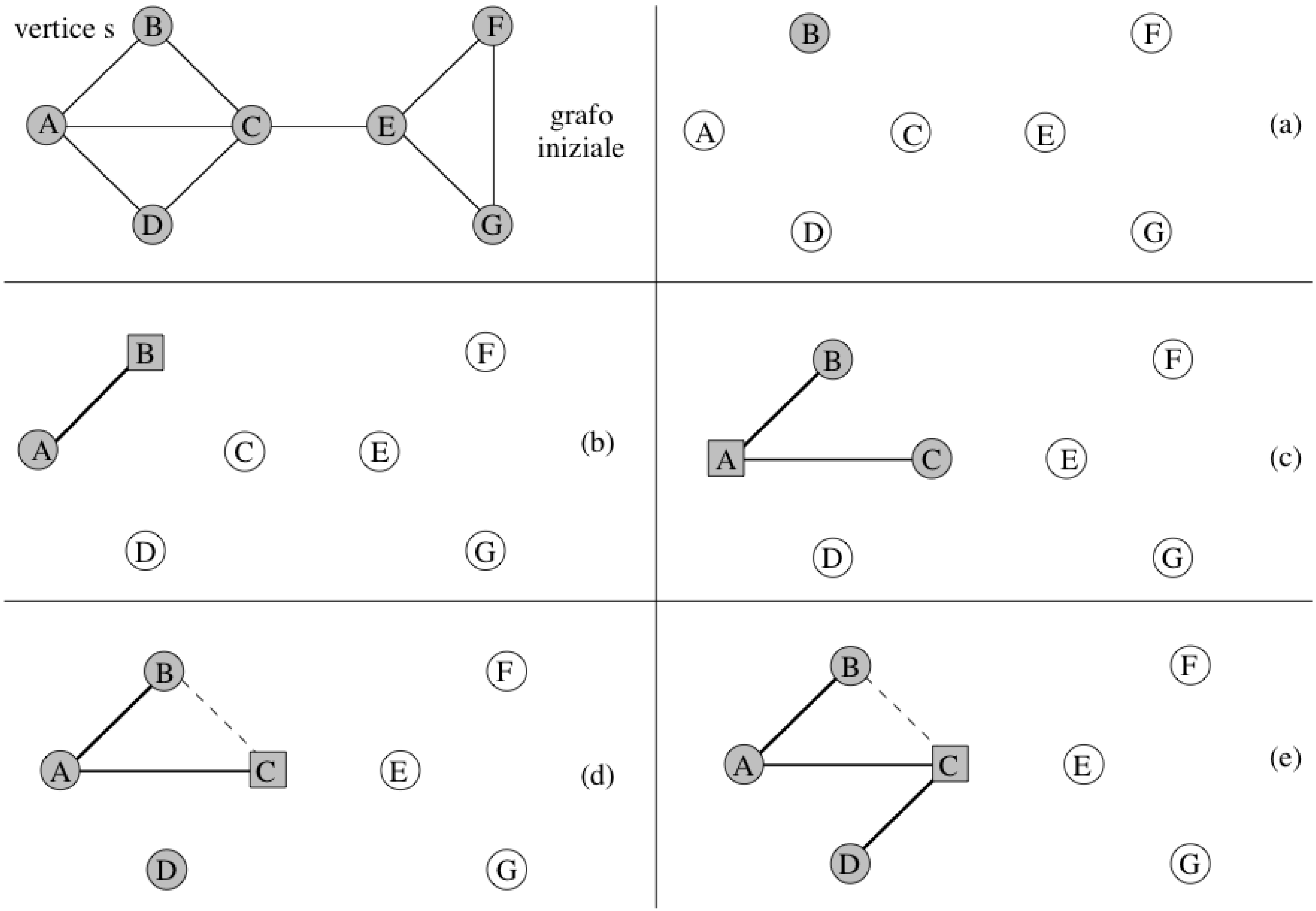
- ✓ We will assume that each message can be **arbitrarily** long
- ✓ According to this model, to establish the efficiency of an algorithm we will only count the **total number** of messages sent during any admissible execution of the algorithm (in other words, the number of *message delivery* events), regardless of their size

Time Complexity

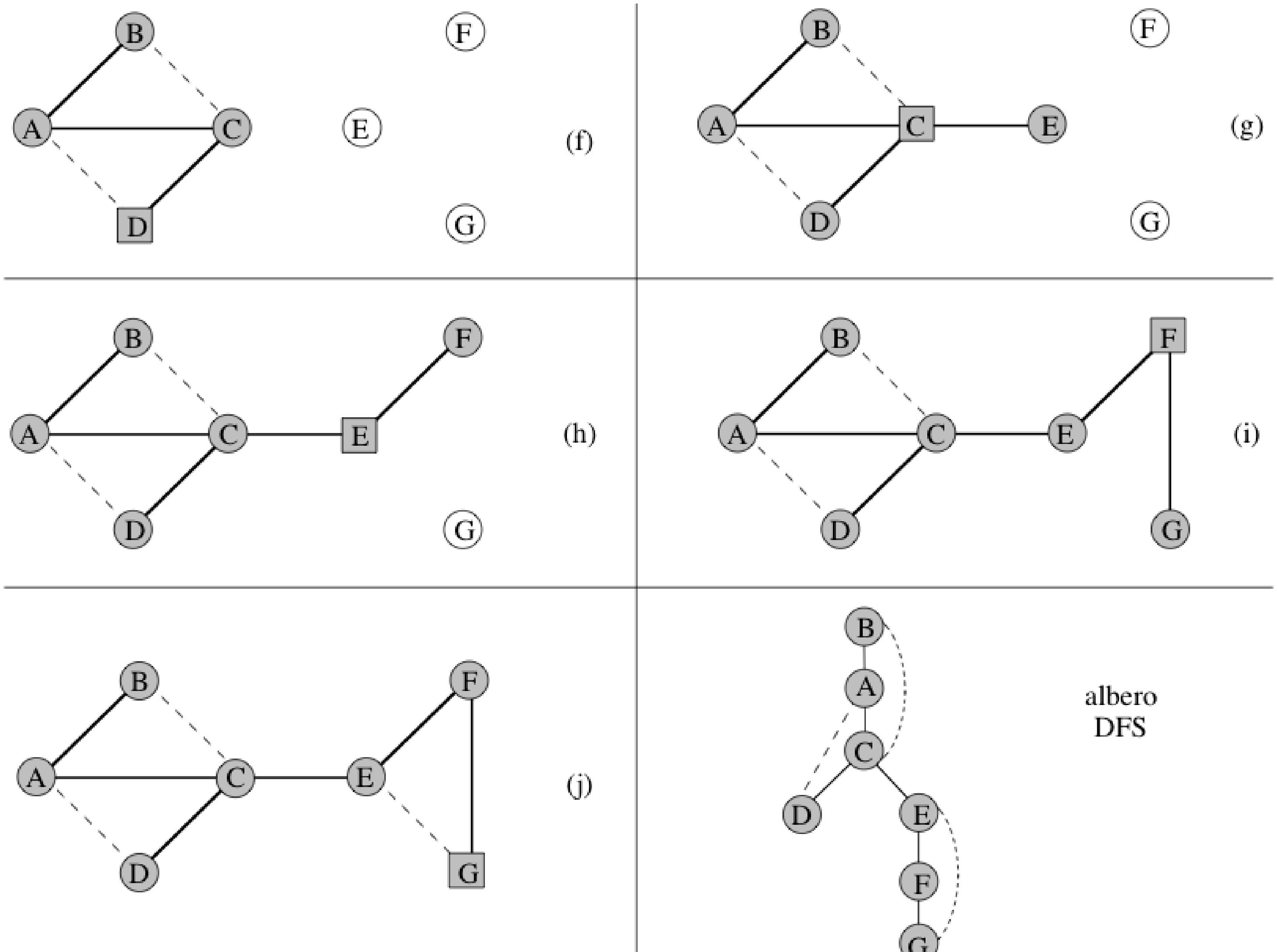
- ✓ We will assume that each processor has **unlimited** computational power
- ✓ According to this model, to establish the efficiency of a synchronous algorithm, we will simply count the number of rounds until termination.
- ✓ Asynchronous systems: the time complexity is not really meaningful, since processors do not have a consistent notion of time

- Example: Distributed Depth-First Search visit of a graph
  - Visiting a (connected) graph $G=(V,E)$ means to explore all the nodes and edges of the graph
  - General overview of a sequential algorithm:
    - Begin at some source vertex, r_0
    - when reaching any vertex v
      - » if v has an unvisited neighbor, then visit it and proceed further from it
      - » otherwise, return to parent(v)
    - when we reach the parent of some vertex v such that $parent(v)$ = NULL, then we terminate since $v = r_0$
  - DFS defines a tree, with r_0 as the root, which spans all vertices in the graph
    - sequential time complexity = $\Theta(|E|+|V|)$ (we use Θ notation because every execution of the algorithm costs exactly $|E|+|V|$, in an asymptotic sense)

DFS: an example (1/2)



DFS: an example (2/2)



Distributed DFS: an asynchronous algorithm

- **Distributed version** (token-based): the token traverses the graph in a depth-first manner using the algorithm described above

  1. Start exploration (visit) at a waking-up node (root) r (who wakes-up r? Good question, we will see…)
  2. When v is visited for the first time:

     2.1 Inform all of its neighbors that it has been visited:

     2.2 Wait for acknowledgment from all neighbors:

     (we will see steps 2.1 and 2.2 are useful in the synchronous case)

     2.3 Select an unvisited neighbor node and pass the token to it; if no unvisited neighbor node exists, then pass the token back to the parent node

- Message complexity is $\Theta(|E|)$ (optimal, because of the trivial lower bound of $\Omega(|E|)$ induced by the fact that every node must know the status of each of its neighbors – this requires at least a message for each graph edge)

Distributed DFS (cont'd.)

Time complexity analysis (synchronous DS)

- Through steps 2.1 and 2.2, we ensure that vertices visited for the first time know which of their neighbors have been visited; this way, each node knows which of its neighbors is still unexplored
- Number of rounds for steps 2.1 and 2.2:
  1. inform all neighbors of *v* that *v* has been visited;
  2. get *Ack* messages from those neighbors;
  3. restart DFS process

$\Rightarrow$ constant number of rounds (i.e., 3) for each new discovered node

- |V| nodes are discovered $\Rightarrow$ time complexity = $\Theta(|V|)$

Homework: What does it happen to the algorithm's complexity if we do not inform the neighbors about having been visited?