Intelligent and Adaptable Software Systems

Advanced Algorithms: Graph Theory

Dr. Arno Formella

Computer Science Department
University of Vigo

09/09

Course organization
course notes

- Homepage:
  `http://www.ei.uvigo.es/~formella/doc/ssia09`
- whiteboard illustrations (notations, ideas for proofs, algorithms)
- very short introduction to graph theory and its applications

Graph Theory, Wednesdays, 18:00–20:00

23.09. class	30.09. class	07.10. lab	14.10. class	21.10. lab
28.10. class	04.11. lab	18.11. class	25.11. class	02.12. lab
09.12. optim.	16.12. lab	13.01. class	20.01. lab	27.01. eval

Course organization

class room hours

- Dra. Marta Pérez Rodríguez
  office hours: Wednesdays, 14-14:30
- Dr. Arno Formella
  office hours: Mondays, 11-14 and 17-20

- Reinhard Diestel. *Graph Theory.* 3rd edition, Springer Verlag, 2005. ISBN 3-540-26183-4.
  Existe una versión electrónica entre–enlazada (no imprimible): `http://diestel-graph-theory.com/index.html`
- Thomas H. Cormen, Charles E. Leiseron, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Second Edition.* Especially Part VI. McGraw Hill, 2001. ISBN 0-262-03292-7.

Bibliography
links (examples...)

(working in september 2009)

- http://www.graphtheory.com
- http://www.ericweisstein.com/encyclopedias/books/GraphTheory.html
- http://mathworld.wolfram.com/Graph.html
- http://en.wikipedia.org/wiki/Graph_theory (take care)

Bibliography

course notes (examples...)

(working in september 2009)

- Gregorio Hernández Peñalver, Universidad Politécnica de Madrid, `http://www.dma.fi.upm.es/docencia/segundociclo/teorgraf`
  (in Spanish)
- Steven C. Locke, Florida Atlantic University, `http://www.math.fau.edu/locke/graphthe.htm`
  (alphabetically ordered)

graph theory: study the material

graph programming libraries: analyze and use of programming tools and libraries that work with graphs (Leda, GraphBase, Boost, etc.)

graph visualization: analyze and use of tools to visualize graphs and the information they contain (OGDF, Graphviz, etc.)

applications: search for applications that use graph algorithms or graphs as data structures (e.g., network planification, route optimization)

Programming

to be completed

graphs can be found, e.g., in the following situations:

- street plans or maps
- networks (data, fluids, traffic etc.)
- transport systems
- chemical connexions in a large molecule
- neighborhood relations in a worldmap
- interference relations between antennas in a wireless communication system
- links between WWW pages

Hence, a graph is an abstract concept behind the representation of relations (edges) between entities (nodes or vertices)

- Determine the order how to dress cloths.
- Is it possible to design a journey through a city that passes through every street exactly once?
- What is the shortest distance a postman must walk to visit (pass along) each street at least once?

- How should we direct the street using one–direction signs such that it is still possible to drive from everywhere to everywhere?
- What are the necessary conditions to organize a group dance at a party such that the pairs consist of partners who knew each other beforehand?
- How should we place the chips on a board to minimize the interconnection length?
- Where should be place the firebrigades to shorten their maximum distances to any house?

Notations

basic issues

V	set of nodes or vertices
$[V]^r$	set of subsets of V of size r
$E \subseteq [V]^2$	set of edges
$v \in V$	vertex or node
$e = \{x, y\} \in E$	edge
$\{x, y\} \Longleftrightarrow: xy$	xy is edge
$G = (V, E)$	graph
$V(G), E(G)$	vertices and edges of the graph G
$v \in G :\Longleftrightarrow v \in V(G)$	v is vertex of the graph G
$e \in G :\Longleftrightarrow e \in E(G)$	e is edge of the graph G
$\|V\| =: n$	number of vertices
$\|V\| = \|V(G)\| = \|G\|$	equivalent notations
$\|E\| =: m$	number of edges
$\|E\| = \|E(G)\| = \\|G\\|$	equivalent notations

Vocabulary
basic concepts

trivial	if $\|G\| = 0$ or $\|G\| = 1$, the graph is trivial
over	if $G = (V, E)$, G is a graph over V
incident	a vertex v is incident to an edge e, if $v \in e$
	an edge e is incident to a vertex v, if $v \in e$
adjacent	two vertices v and w are adjacent, if $\{v, w\} \in E$
	two edges e and f are adjacent, if $e \cap f \neq \emptyset$
connected	an edge connects its vertices
$X - Y$–edge	if $x \in X \subseteq V$ and $y \in Y \subseteq V$, xy is $X - Y$–edge
$E(X, Y)$	set of $X - Y$–edges

Notations

degree related

$E(v) := E(v, V \setminus \{v\})$	set of edges incident to v
$N(v)$	set of vertices adjacent to v (neighbors)
$d(v) := \lvert E(v) \rvert = \lvert N(v) \rvert$	degree of vertex v
$d_G(v)$	degree of vertex $v \in G$
$\delta(G)$	minimum degree of the vertices in G
$\Delta(G)$	maximum degree of the vertices in G
$d(G) := 2\lvert E \rvert / \lvert V \rvert$	mean degree of the vertices en G
$\epsilon(G) := \lvert E \rvert / \lvert V \rvert = \frac{d(G)}{2}$	mean number of edges per vertex of G

Notations

substractions

$G \setminus e$	graph $(V, E \setminus \{e\})$
$G \setminus v$	graph $(V \setminus \{v\}, E \setminus E(v))$
$G \setminus E'$	graph $(V, E \setminus E')$
$G \setminus V'$	graph $(V \setminus V', E \setminus E(V'))$

neighbor node	v is neighbor of w, if $vw \in E(v)$, i.e., if v and w are adjacent
neighbor edge	e is neighbor of f, if $e \cap f \neq \emptyset$ i.e., e and f are incident to the same vertex
independent	vertices/edges non adjacent, a set of vertices (edges) mutually independent is an independent set
complete	a graph is complete, if all its vertices are neighbors
partition	the set of set $\{V_0, ..., V_{r-1}\}$ is a partition of V, if $V = \bigcup_i V_i$, $V_i \neq \emptyset$, and $\forall i \neq j : V_i \bigcap V_j = \emptyset$

$\alpha(G)$ size of the largest independent set of vertices

Vocabulary
graphs

digraph	the edges are directed, i.e., instead of the sets $\{v, w\}$ we use pairs (v, w) or (w, v) i.e., $E \subseteq V \times V$
multigraph	permits more than one edge between vertices
pseudograph	permits loops on vertices

A graph can be stored with three basic methods:

- adjacency matrix
  - square matrix, and in the simple case, binary (and symmetric if not digraph) that codes whether there exists an edge between vertices
  - space complexity $\Omega(n^2)$
- adjacency lists
  - list or array of vertices which contain in each entry a list to its adjacent vertices
  - space complexity $\Theta(n+m)$

- hashtables
  - list or array of vertices which contain in each entry a hashtable to its adjacent vertices
  - space complexity $\Theta(n + m)$

What are the principal advantages and disadvantages of each method?

There are more data structures available which are useful to implement certain algorithms more efficiently (especially for planar graphs).

Let $G = (V, E)$ and $G' = (V', E')$ be two graphs.

If there is a bijection $\varphi : V \longrightarrow V'$ between the vertices of the graphs such that

$$xy \in E \Longleftrightarrow \varphi(x)\varphi(y) \in E'$$

(i.e., if x and y are neighbors in G, then $\varphi(x)$ and $\varphi(y)$ are neighbors as well in G'),

then G is isomorph to G', $G \simeq G'$ or as well $G = G'$, i.e., one can say the graph G.

A function f over two graphs with $f(G) = f(G')$ if $G \simeq G'$ is called invariant. E.g., invariants are:

- n
- m
- Are there more invariants?

Invariant

open problem

No one knows an invariant for graphs which can be computed in deterministic polynomial time which decides whether two graphs are isomorphic, however, it is not demonstrated that the problem is *NP*–complete.

Remind what it means *NP*–complete:

A problems belongs to the class of *NP*–complete problems, if there exists deterministic Turing machine (sufficiently powerful computing model) that solves the problem in polynomial time (in respect to input length) and all other problems in the class are at most simpler.

Hence we know for *NP*–complete problems:

- There exists an algorithm for solving it. (We can always use exhaustive search.)
- If someone gives us a potential solution, we can verify in polynomial deterministic time that it is really a solution.
- If we would know a polynomial algorithm for any *NP*–complete problem, implicitely we could solve all problems of the class with that time bound.
- Whether the two classes *P* and *NP* are equal is one of the famous open problems in computer science (and most people think they are different).

Isomorphism
is in NP

Given a bijection between the nodes of two graphs, it is easy to prove whether the graphs are isomorphic: it is sufficient to check whether the adjacency matrices are identical, this can be done in time $O(m)$.

Hence, the isomorphism problem is in *NP*.

Let $G = (V, E)$ and $G' = (V', E')$ be dos graphs.

$$G \cup G' := (V \cup V', E \cup E') \quad \text{union of the graphs}$$
$$G \cap G' := (V \cap V', E \cap E') \quad \text{intersection de graphs}$$

partial graph	G' is partial graph of G, if $V' \subseteq V$ and $E' \subseteq E$
subgraph	G' is subgraph of G, of $G' \cap G = G'$, i.e., G' is a partial graph of G that contains all edges of G whose incident vertices are in V'.
induced	for $V' \subseteq V$ the graph $(V', E(V', V'))$ is the subgraph G' of G induced by V'

Vocabulary
subgraphs

$G' \subseteq G$	G' is partial graph of G
$G[V']$	subgraph of G over V' assuming $V' \subseteq V$
$G[G']$	subgraph of G over $V(G')$ assuming $G' \subseteq G$

Vocabulary

partition

r–partite	a graph is an r-partition, if their exists a partition of V defining r independent sets
bipartite	2–partite

k–regular graphs	$d(G) = \epsilon(G) = k \ \ (= \delta(G) = \Delta(G))$
complete graphs K^r	$G = (V, [V]^2), \lvert V \rvert = r$
paths P^k	$G = (\{v_0, \ldots, v_k\}, \{v_0 v_1, v_1 v_2, \ldots v_{k-1} v_k\})$
cycles C^k	$G = (\{v_0, \ldots, v_k\}, \{v_0 v_1, v_1 v_2, \ldots v_{k-1} v_0\})$

Obviously, one can describe a path or cycle by its node sequence.

length	number of edges of a path or cycle
cyclic	a graph that contains a cycle is cyclic
acyclic	a graph that does not contain a cycle is acyclic
girth	a minimum length cycle of the graph
circumference	a maximum length cycle of the graph

$g(G)$	length of the girth of G ($g(G) = \infty$ if G acyclic)
$D(G)$	length of the circumference of G ($D(G) = 0$ if G acyclic)

- bipartite graphs B
- bipartite complete graphs K^{n_1,n_2}
- hypercubes Q^k
  properties (with exceptions for Q^0 and Q^1):
  - $n = |V| = 2^k$
  - k–regular
  - bipartites (How to partition?)
  - $g(Q^k) = 4$
  - $D(Q^k) = 2^k$ (What is a maximum cycle?)
  - $\alpha(Q^k) = 2^{k-1}$ (and there are two)

Distance

must be a metric

$d(v, w)$	distance between two vertices being the length of the shortest path between v and w
$d_G(v, w)$	distance between v and w in G

The distance defines a metric, i.e.,

1. $d(v, w) \geq 0$
2. $d(v, w) = 0 \iff v = w$
3. $d(v, w) = d(w, v)$
4. $d(u, w) \leq d(u, v) + d(v, w)$

connected	v and w are connected, if $d(v, w) < \infty$; G is connected, if all pairs $\{v, w\} \subset V$ are connected
unconnected	G is unconnected if it is not connected
bridge	edge $e \in G$ is a bridge if G is connected but $G \setminus e$ is unconnected
cut vertex	vertex $v \in G$ if G connected but $G \setminus v$ is unconnected
connected components	connected and maximum subgraphs

Notations

and more invariants

$c(G)$	number of connected components of G
$\kappa(G)$	minimum size of vertex subset of G such that $G \setminus V$ is unconnected
$\lambda(G)$	minimum size of edge subset of G such that $G \setminus E$ is unconnected

k–connected	G is k–connected, if $\kappa(G) \geq k$
biconnected	2–connected
k–edgeconnected	G is k–edgeconnected, if $\lambda(G) \geq k$
block	maximum biconnected subgraph

Forests

and its basic parts: trees

- if G does not contain cycles, G is a forest
- if G is a forest and connected, G is a tree
- the connected components of a forest are trees

Theorem

the following properties are equivalent:

- *G is a tree*
- *between edge pair of vertices there exists exactly one path*
- *each edge is a bridge*
- *G is acyclic and $n = m - 1$*
- *G is connected and $n = m - 1$*
- *G is acyclic and maximal in $|E|$*
- *G is connected and minimum in $|E|$*

free tree	a tree where no vertex is marked
rooted tree	a tree with one vertex marked as root
spanning tree	T is spanning tree of a graph G, if $T \subseteq G$ and $V(T) = V(G)$

Theorem

each graph G contains a spanning forest, and if G is connected, it contains a spanning tree (with any vertex as root)

euler walk	path between two vertices that does not visit more than once an edge
euler graph	graph with euler walk using all its edges
hamilton graph	graph with path (cycle) over all its vertices