Intelligent and Adaptable Software Systems

Advanced Algorithms: Graph Theory

Dr. Arno Formella

Computer Science Department
University of Vigo

11/10

Advanced Algorithms: Graph Theory I

Course organization

course notes

- Homepage:
  `http://www.ei.uvigo.es/~formella/doc/ssia10`
- almost everything will be accessible on our moodle-platform:
  `http://postgrado.ei.uvigo.es/tadsi-online/login/index.php`
- whiteboard illustrations (notations, ideas for proofs, algorithms)
- very short introduction to some specific aspects of graph theory and their applications

Course organization

class room hours

Graph Theory, Thursdays, 18:00–20:00

16.09. (Marta)	23.09. (Marta)	30.10. (Marta)	07.10. (Marta)	14.10. (Marta)
21.10. (Marta)	28.11. (Marta)	04.11. class	18.11. class	25.11. lab
02.12. lab	09.12. class	16.12. lab	13.01. eval	20.01. eval
27.01. eval				

Course organization
office hours

- Dra. Marta Pérez Rodríguez
  office hours: thursdays, 16-17
- Dr. Arno Formella
  office hours: mondays, 10-13 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.

(working in november 2010)

- 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 november 2010)

- 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)

Your work

homework, lab hours, presentations

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, yED, etc.)

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

Quick test

no surprise

- What is a bridge?
- What is a partition?
- What is an independent set?
- What is a bipartite graph?
- What is the time complexity of DFS?

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
- closeness relation in arrangments

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

Motivation
usage to resolve problems

- 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 or the necessary ones 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)
$d(v) := \|E(v)\| = \|N(v)\|$	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
$d(G) := 2\|E\|/\|V\|$	mean degree of the vertices en G
$\epsilon(G) := \|E\|/\|V\| = \frac{d(G)}{2}$	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

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

Representation

data

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 $\Theta(n + m)$

Representation

data

- 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).

Isomorphism
definition

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.

NP-completeness

definition

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 $O(n^2)$.

Hence, the isomorphism problem is in *NP*.

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

$G \cup G' := (V \cup V', E \cup E')$	union of the graphs
$G \cap G' := (V \cap V', E \cap E')$	intersection of the 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 maximum 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)

Vocabulary

walks

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

Theorem (EULER (handshaking lemma))

$$\sum_{v} d(v) = 2m$$

Idea of proof

count

Theorem

each graph contains an even number of odd degree vertices

Idea of proof

use theorem of EULER

Theorem

each graph G contains a path P with $\|P\| \geq \delta(G)$, and each graph G with $\delta(G) \geq 2$ contains a cycle C with $|C| > \delta(G)$

Idea of proof

observe the neighbors of the last vertex on a longest path

Theorem

for G not trivial: $\kappa(G) \leq \lambda(G) \leq \delta(G)$

Theorem

each graph G with $|E| > 1$ *contains a subgraph H with* $\delta(H) > \epsilon(H) \geq \epsilon(G)$

Theorem

G is bipartite with $|V_0| \neq |V_1| \implies$ *G is not hamiltonian*

Theorems

more basic theorems

Theorem

$\forall v, w \in V, vw \notin E : d(v) + d(w) \geq n \quad \Longrightarrow \quad G \text{ is hamiltonian}$

Theorem

$G \text{ is hamiltoniano} \quad \Longrightarrow \quad \forall S \subset V : c(G \setminus S) \leq |S|$

Theorem

to decide whether a graph G is hamiltonian is NP–complete

Theorem

G is bipartite $\iff$ *G does not contain odd cycles*

Idea of proof

bi–coloring of a spanning forest

Algorithm that decides whether G is bipartite?

Theorem (Euler)

G is eulerian $\iff$ *G is connected and* $\forall v \in V : d(v)$ *is par*

Idea of proof

analyse path of maximum length that passes through a vertex

Algorithm that computes a euler walk?

Theorem

G *k–connected* $\Longrightarrow \quad |E| \geq \lceil kn/2 \rceil$

Theorem (Whitney)

G is biconnected ($|G| > 2$) $\implies \forall v, w \in V \exists vPw, vQw : P \cap Q = \emptyset$

i.e., there exist always two paths that do not intersect (for all possible pairs of vertices in G)

Theorem (Mader)

each graph G with mean number of edges $\epsilon(G) \geq 4k$ *contains a k–connected graph as partial subgraph*

Algorithm

Depth first search

DFS generates a rooted tree T with back–edges (that do not belong to T). DFS serves, for examples, to

- determine connected components
- detect bridges
- detect cut vertices
- detect blocks
- sort topologically

DFS

properties

Theorem

Let T be a tree of a connected graph G with v as root generated by DFS.
v is a cut vertex $\iff$ v has more then one son in T

Theorem

Let T be a tree of a connected graph G generated by DFS and v not its root.
v is a cut vertex $\iff$ there does not exist a back–edges from the subtree below v towards an predecesor of v in T

DFS has complexity $\Theta(n + m)$
(assuming adjacency lists, and with the other storing possibilities?)

Algorithm

breadth first search

BFS generates a rooted tree T with back–edges and cross–edges.
BFS serves, for example, to:

- determine the shortest paths between vertices
- sort topological

BFS has complexity $\Theta(n + m)$
(assuming adjacency lists, and with the other storing possibilities?)

Let G be a graph and let $\overline{G}$ be a directed graph (digraph).

strongly connected	$\overline{G}$ is strongly connected, if there exists a walk between each pair of vertices de G
orientable	G is orientable, if there exists a graph $\overline{G} \simeq G$ that is strongly connected
strongly connected component	maximum set of vertices of $\overline{G}$ whose induced subgraph is strongly connected

$d_i(v)$	indegree
$d_o(v)$	outdegree

Theorem (ROBBINS)

G is orientable $\iff$ G is connected and does not contain bridges.

Algorithm that computes an orientation?

(What is an *optimal* orientation?

It depends: minimizing the average distance, minimizing the maximum distance, minimizing the differences between distances in G (in some sense) and the corresponding distances in $\overline{G}$ etc.)

DFS and BFS can be used over digraphs as well.

DFS now generates forward–edges and cross–edges as well.

DFS can be used to compute a topological sorting of a acyclic digraph, i.e., in the ordering a vertex v appears before a vertex w, if there exists a path from v to w.

DFS can be used to determine the strongly connected components.

Algorithm that computes the strongly connected components?

one can follow euler walks as well::

Theorem

$\overline{G}$ is eulerian $\iff$ $\overline{G}$ is connected and $\forall v \in V : d_i(v) = d_o(v)$

Algorithm that calculates a euler walk in a digraph?

(G, W)	weighted graph with $W : E(G) \longrightarrow \mathbb{R}^+$
$w(G)$	weight of the graph G, $w(G) = \sum_{e \in G} w(e)$
$w(P)$	weight of a path P, $w(P) = \sum e \in Pw(e)$
$d(v, w)$	distance between two vertices, $d(v, w) = \min_P\{w(vPw)\}$
$dt(v) = \sum_w d(v, w)$	total distance of a vertex

with $w(e) = 1 \forall e \in E$ we reproduce the distance

$e(v)$	excentricity, $e(v) = \max_{w \in G}\{d(v, w)\}$
$\mathrm{rad}(G)$	radius of a graph G, $\mathrm{rad}(G) = \min_{v \in G}\{e(v)\}$
$\mathrm{diam}(G)$	diameter of a graph G, $\mathrm{diam}(G) = \max_{v \in G}\{e(v)\}$

center	the center of a graph G is the subgraph of G induced by the vertices with minimum excentricity
median	the median of a graph G is the subgraph of G induced by the vertices with minimum total distance

Theorem

G connected $\implies$ $\text{rad}(G) \leq \text{diam}(G) \leq 2 \cdot \text{rad}(G)$

Theorem

Every graph is the center of a graph.

Theorem

The center of a tree consists of one or two vertices.

Algorithm to calculate the center of a tree?

given a weighted graph G (or a digraph $\overline{G}$)

interesting questions are:

- What is the minimum spanning forest for G?
- given a vertex $s \in \overline{G}$, What are the minimum paths to all other vertices?
- What are the minimum paths among all pairs of vertices?

- To each vertex $u \in G$ there exists a shortest path from s.
- The paths to all vertices along the path are shortest paths as well.
- If G is connected, we can construct a tree with root $s \in G$ which deternines all shortest paths to the other vertices in the graph.
- Algorithm that calculates a minimum forest of a graph G?

Algorithm

KRUSKAL, join greedily trees with minimum edges, complexity $O(m \log n)$

Algorithm

PRIM, construct iteratively a tree with minimum edges, complexity $O(m + n \log n)$

Algorithm that computes a minimum tree starting at a vertex s in $\overline{G}$?

Algorithm

DIJKSTRA, complexity $O(n^2)$

The algorithm works equally with graphs and with digraphs.

One can permit negative weights?

There might exist negative cycles.

Algorithm that calculates a mimimum tree starting at a vertex *s* en *G* if *G* contains negative weights?

Algorithm

BELLMANN–FORD, complexity $O(mn)$

Algorithm that calculates the minimum paths between all pairs of vertices including the case of negative weights?

Algorithm

FLOYD–WARSHALL, complexity $O(n^3)$

Some algorithms detect negative cycles in which case they simply stop.

if we have more information over the graph and its weights we can improve the algorithms:

- if the weights are confined by a constant W:
- if the number of edges m is confined by $O(n \log n)$:

- contraction of edges
- minor
- topological minor

Consequence:

Theorem (ROBERTSON/SEYMOUR)

Each heritable property over graphs can be represented by a finite number of prohibited minors.

There are interesting questions how certain properties of graphs are related to the size of the graph (normally number of edges in relation to number of nodes).

- How many edges a graph must have as minimum such that it is connected?
- How many edges a graph can hve as maximum such that it is planar?

planar	a graph is planar if we paint its vertices and its edges in the plane such that no pair of edges crosses
region	a region is an area of the plane where we have painted a planar graph that is surrounded by edges
triangular graph	a graph is triangular, if in its planar representation all regiones are triangles

Theorem (EULER)

G is planar and connected with $n \leq 1$, m *and* $l \implies n - m + l = 2$

Theorem

G planar with $n \geq 3 \implies m \leq 3n - 6$

Theorem

G maximum planar if it is a triangular graph (and contains $3n - 6$ *edges)*

Theorem (KURATOWSKI)

G is planar $\iff$ *does not contain a* K^5 *or a* $K^{3,3}$ *as minor (or topological minor)*

- Coloring of vertices
- Coloring of edges
- Coloring of planar graphs

Theorem

G planar $\Longrightarrow$ *the vertices of G can be colored with 4 colors*

Theorem

G planar with no triangle $\Longrightarrow$ *the vertices of G can be colored with 3 colors*

$\chi(G)$	minimum number of colors needed to color the vertices of a graph
$\chi'(G)$	minimum number of colors needed to color the edges of a graph
$\chi''(G)$	minimum number of colors needed to color the vertices and the edges of a graph

Theorem (Brooks)

G connected $G \neq C^{impar}$ and $G \neq K^n \quad \Longrightarrow \quad \chi(G) \leq \Delta(G)$

Theorem (Koenig)

G bipartite $\quad \Longrightarrow \quad \chi'(G) = \Delta(G)$

Flows

types

1. flows and networks
2. flows and cuts
3. flows and coloring

feasable	$f(e) \leq w(e)$, i.e., the flow at most is equal to the capacity
conserving	$\sum_v f_{in}(e) = \sum_v f_{out}(e)$, i.e., as much as enters exits a node (obviously, besides sources and drains).

Theorem (FORD–FULKERSON)

Maximum flow is equal to minimum cut.

Algorithm

FORD–FULKERSON, complexity $O(F * m)$ where F is the maximum flow (asuming integers as capacities).

Algorithm

EDMONDS–KARP, complexity $O(nm^2)$.

Algorithm

DINIC, complexity $O(n^2 m)$.

Algorithm

KARZANOV–GOLDBERG–TARJAN, complexity $O(n^3)$.

There are more varieties of restrictions: maximum capacities for the nodes, more sources/drains, minimum capacities, etc.

Vocabulary
matching

matching	$M \subset E$ and $\forall e, f \in M : e \cup f = \emptyset$, i.e., pairs of edges that do not have common vertices
perfect	M covers all vertices of G
M–alternating	a path in G alternating with edges of M
M–aumenting	M–alternating path that can be aumented

Theorem (BERGE)

matching M is maximum $\implies$ G does not contain M–aumenting paths

Theorem (HALL)

wedding theorem: Let $G = (U \cup V, E)$ be a bipartite graph. G has a perfect matching (for U) $\implies$ $\forall S \subseteq U : |N(S)| \geq |S|$, where $N(S)$ is the set of neighbors of S.

1. Planification
2. Circuit design
3. Games
4. Optimization
5. Coding theory
6. Visualization