Discrete Mathematics

Sebastian Iwanowski
FH Wedel

Ch. 7: Graph Theory

References:

Iwanowski / Lang 7 (in German)
Rosen 8
Epp 11 (except for Dijkstra’s algorithm)
Biggs 15 (only for 7.1)

7. Graph theory

7.1 Terminology and representation

Definition:

A Graph (V,E) is a construct of vertices (nodes) and edges:
An edge always connects 2 vertices. These vertices are the *endpoints* of the edge.

Representation in the plane:

- The vertices are marked as points in the plane.
- The edges are curve segments (normally line segments) connecting the endpoints.
- The representation of a graph is not unique.

Isomorphism or: When are 2 graphs considered equal?

2 graphs are considered equal (equivalent, isomorphic), when they have the same number of vertices and edges and there is a bijective mapping in between such that vertices connected in the first graph by an edge impose that the images of the vertices in the second graph are connected by the image of the edge and vice versa.

7. Graph theory

7.1 Terminology and representation

Further notation:

- Edges may be directed or undirected.
  Directed edges are called ***arcs***.
  Graphs with undirected edges only are called ***undirected graphs***,
  Graphs having directed edges are called ***Digraphs***.
- adjacent vertices
- incident vertices and edges
- loops, multiple edges
- degree of a vertex
- connectivity, connectivity components, isolated vertices

Simple graphs *have no loops or multiple edges.*

7. Graph theory

7.1 Terminology and representation

Representation of a graph in a computer:

- **Adjacency matrix:**
  At position (i,j) is a 1, if vertex i is connected to vertex j by an edge, else 0.

- **Adjacency list:**
  In line i there is a list of all vertices connected to vertex i by an edge.

- **Incidence matrix:**
  At position (i,j) there is a 1, if edge i has vertex j as endpoint, else 0.
  (Note: lines and columns may be swapped, i.e. the lines hold the vertices and the columns the edges).

7. Graph theory

7.2 Path problems in graphs

Euler paths

Definition *Euler path*:

Path traversing each edge of the graph exactly once

Definition *Euler cycle*:

Closed Euler path (same starting and finishing vertex)

Definition *Eulerian graph*:

Graph having an Euler cycle

Proposition: Graph G is Eulerian ⇔ G is connected
and each vertex has even degree

7. Graph theory

7.2 Path problems in graphs

Euler paths

Algorithm for finding an Euler cycle in an Eulerian graph:

- Start with an arbitrary vertex v_0 and the empty path $P_0 = (v_0)$.
- Repeat:

  Enhance the path $P_i = (v_0, e_1, v_1, ..., v_{i-1}, e_i, v_i)$
  to a path $P_{i+1} = (v_0, e_1, v_1, ..., v_{i-1}, e_i, v_i, e_{i+1}, v_{i+1})$ by an edge e_{i+1}
  starting at the last vertex v_i of P_i,
  such that the remainder graph R_{i+1} arising from G by th removal of all edges
  of W_{i+1} and the hence isolated vertices of G,
  is connected and still contains the vertex v_0.
  (i.e. the removal of e_{i+1} may not isolate the vertex v_0 and must keep the edges not yet selected in one connectivity component)

  until this is not possible anymore.

Proposition: The path P_k generated by this algorithm contains all edges of G, i.e. P_k is an Euler cycle.

Question: How to check if a graph has still only 1 connectivity component?

7. Graph theory

7.2 Path problems in graphs

Euler paths

Algorithm for finding an Euler path
in a connected graph with exactly 2 vertices v_s and v_e of odd degree:

- Start with the vertex v_s and the empty path $P_0 = (v_s)$. (Start at v_e and finish at v_s would also work)
- Repeat:

  Enhance the path $P_i = (v_s, e_1, v_1, ..., v_{i-1}, e_i, v_i)$
  to a path $P_{i+1} = (v_s, e_1, v_1, ..., v_{i-1}, e_i, v_i, e_{i+1}, v_{i+1})$ by an edge e_{i+1}
  starting at the last vertex v_i of P_i,
  such that the remainder graph R_{i+1} arising from G by th removal of all edges of W_{i+1} and the hence isolated vertices of G,
  is connected and still contains the vertex v_e.
  (i.e. the removal of e_{i+1} may not isolate the vertex v_e and must keep the edges not yet selected in one connectivity component)

  until this is not possible anymore.

7.2 Path problems in graphs

Hamilton paths

Definition *Hamilton path*:

Path traversing each vertex of a graph exactly once

Definition *Hamilton cycle*:

Closed Hamilton path (same starting and finishing vertex)

Definition *Hamiltonian Graph*:

Graph with a Hamilton cycle

Problem: There is no efficient algorithm known how to find a Hamilton cycle. ☹

7. Graph theory

7.2 Path problems in graphs

<u>Weighted graphs</u>

Definition *Weighted Graph*:

Graph where the edges are marked with weights

Remark: Weighted graphs may also be directed.

Representation of a weighted graph in the computer:

- **Adjacency matrix:**
  Position (i,j) holds the edge weight, if vertex i and j are connected, else ∞.

- **Adjacency list:**
  Line i holds the pairs (vertex number, edge weight) of all vertices being connected to vertex i.

7. Graph theory

7.2 Path problems in graphs

Dijkstra's algorithm for the computation of the shortest path:

Prerequesite: All edge weights must be nonnegative.

Goal: Find the path with minimal weight from a chosen source S to a chosen target T.

Algorithm:

- Put S into the set **`Done`**. Label S by *distance*(S) := 0.
  Put all other nodes into the set **`YetToCompute`**.
  Label all neighbors N of S by *distance* (N) := *length* (S,N)
  and all othe nodes by *distance* (V) := ∞.
- Repeat:
  Choose node V from **`YetToCompute`** with minimum *distance* (V)
  and shift V to the set **`Done`**.
  Update all neighbors N of V that are still in **`YetToCompute`**:
  distance (N) := min {*distance* (N), *distance* (V) + *length* (V,N)}.
  until V = T

7. Graph theory

7.2 Path problems in graphs

Dijkstra‘s algorithm for the computation of the shortest path:

Proposition: The labels of all vertices V in the set `Done` correspond to the weight of the shortest path from S to V.

Extension to the *Output* of the shortest path:

- Repeat:
  Choose node V from `YetToCompute` with minimum *distance* (V)
  and shift V to the set `Done`.
  Update all neighbors N of V that are still in `YetToCompute`:
  distance (N) := min {*distance* (N), *distance* (V) + *length* (V,N)}.
  If *distance (N)* changed, let V be the predecessor of N.
  until V = T
- Collect the predecessors of T subsequently and output them in revers order.

Proposition: Dijkstra‘s algorithm does not only compute the shortest path from S to T, but also the shortest paths from S to all other vertices that are closer away from S than T.

7. Graph theory

7.3 Trees

Definition *Tree*:

A tree is a connected graph without cycles.

Definition *Forest*:

A forest is a graph without cycles (and need not be connected).

Theorem: The following propositions are equivalent:

- G is a tree (i.e. connected and without cycles).
- G is a graph without cycles with maximum number of edges among the given vertices (i.e. whenever an edge is inserted among vertices, a cycle will arise).
- G is a connected graph with n-1 edges (where n is the number of vertices).
- G is a graph without cycles having n-1 edges (where n is the number of vertices).

7. Graph theory

7.3 Trees

Definition *Spanning Tree*:

A spanning tree of a graph is a subgraph being a tree (i.e. connected and without cycles) and containing all vertices of the original graph.

Construction of a spanning tree for an arbitrary graph G:

- Start with a empty forest F containing no edge.
- Repeat for all edges e_1, e_2, ..., e_m of graph G (prechosen arbitrary order):
  Check if e_i may be addes to F,
  such that F remains without cycles:
  If so, add e_i to F.
  until F contains n-1 edges (where n is the number of vertices of G.

Theorem: Thus constructed forest F is a spanning tree of G.

7. Graph theory

7.3 Trees

Definition *Minimum Spanning Tree*:

A minimum spanning tree of a graph is a spanning tree where the global edge cost is minimal among all spanning trees.

Kruskal‘s algorithm:

Construction of a minimum spanning tree for an arbitrary graph G:

- Start with a empty forest F containing no edge.
- Repeat for all edges e_1, e_2, ..., e_m of graph G (prechosen sorted order):
  Check if e_i may be addes to F,
  such that F remains without cycles:
  If so, add e_i to F.
  until F contains n-1 edges (where n is the number of vertices of G.

Theorem: Thus constructed forest F is a minimum spanning tree of G.

7. Graph theory

7.3 Trees

Definition *Rooted Tree*:

- Ein *rooted tree* is a tree with one node marked as the root
- The *(search) level of a node* is the number of edges needed to get to the root.
- Die *depth (height) of a rooted tree* is the maximum level of all its nodes.
- The neighbors of a node being on a higher level than the node itself are called *children* of this node.
- The (unique) neighbor of a node being on a lower level than the node itself is called *parent* of this node.
- A *leaf* is a node without children.

Rooted trees with maximum limits for the number of children:

- A binary *rooted tree* is a rooted tree where nodes have got at most 2 children.
- A ternary *rooted tree* is a rooted tree where nodes have got at most 3 children.
- A d-ary *rooted tree* is a rooted tree where nodes have got at most d children.

Proposition: The depth of a d-ary rooted tree with n leaves must be at least $\log_d n$

7. Graph theory

7.4 Planar graphs

Definition *Planar Graph*:

A graph that *may* be represented in a plane such that no edges cross each other.

Definition *Face (stands for a country in a map)*:

A face of a planar graph represented by a *crossing-free embedding in the plane* is a maximum area of the plane in which any two area points may be connected by a curve that does not intersect any edge of the graph.

A face is normally characterised by the bounding edges („Needles“ are considered).
This characterisation is unique for the faces in a given embedding, but not vice versa, i.e. different faces may have the same bounding edges.

7. Graph theory

7.4 Planar graphs

Proposition: The characterisation of faces by their bounding edges depends on the embedding in the plane.

Proposition: The *number* of faces by their bounding edges does *not* depend on the embedding in the plane:

$$n - m + f = 2$$

Euler‘s polehedron formula for connected graphs

$$n - m + f = 1 + c$$

Euler‘s polyhedron formula for graphs with c connectivity components

7. Graph theory

7.4 Planar graphs

How does a graph‘s structure show that the graph is planar ?

Def.: The ***complete graph*** $\boldsymbol{C_n}$ is a graph with n vertices, each of which pairwise connected by an edge.

Def.: The ***complete bipartite graph*** $\boldsymbol{C_{m,n}}$ is a graph with two sets made of m resp. n vertices such that each vertex of one set is connected to each vertex of the other set by an edge.

Proposition: C_n is planar if and only if $n \leq 4$.

Proposition: $C_{m,n}$ is planar if and only if $\min\{m,n\} \leq 2$.

Kuratowski‘s theorem:

A graph is planar if and only if it does not contain any subdivision of C_5 or $C_{3,3}$.

Def: A subdivision of a graph is obtained by inserting additional vertices into existing edges.

7. Graph theory

7.5 Graph coloring

Def.: An *admissible graph coloring* is the assignment of numbers from a finite set (the „colors“) to the vertices of the graph such that no different adjacent vertices have got the same color.

Def.: The chromatic number $\chi(G)$ is the minimum number of colors necessary to achieve an admissible graph coloring.

4-Color Theorem (vertices):

For each planar graph holds: $\chi(G) \leq 4$ (*Four colors suffice!*)

ca. 1850:	conjecture of the British mathematics student Francis Guthrie
1879:	„Proof“ by Alfred Kempe
1890:	Detection of a crucial error in the „proof“ of Kempe by Percy Heawood
20. century:	4-color conjecture as incentive for many developments in graph theory
ca. 1965:	Preparation of a computer driven proof by H. Heesch, no fund for computer
1976:	Computer driven proof by K. Apel und W. Haken based on ideas of Heesch

7. Graph theory

7.5 Graph coloring

How does our definition of χ(G) relate to maps ?

Def.: For a planar graph embedded with no edge crossings into the plane, the *dual graph D* is defined as the graph generated by the following way:
i) Replace each face in G by a vertex in D.
ii) Connect two vertices in D by an edge if and only if the corresponding faces in G are adjacent by an edge. For each bounding edge in G there must be a connecting edge in D which may lead to multiple edges.

Proposition: i) The dual graph of a planar graph is again planar and always connected.
ii) If the original graph is connected,
then the dual graph of a dual graph is the original graph.
i) and ii) imply that each planar connected graph may be considered a dual graph of some other graph. This makes each vertex coloring of one connected graph a face coloring of another graph and vice versa.

4-Color Theorem (maps):

For each map holds: The map may be colored with 4 colors such that any two countries adjacent by a 1-dimensional border have different colors.

7. Graph theory

7.5 Graph coloring

Some bounds for the (vertex-) chromatic number

Definition: A bipartite graph is a graph with two vertex sets M and N such that edges are only connecting a vertex of M with a vertex of N, but there are no edges between vertices of M or between vertices of N. Thus, any bipartite graph is a subgraph of some $C_{m,n}$ (the complete bipartite graph between m resp. n vertices).

Proposition: For any bipartite graph G holds: $\chi(G) = 2$.

Corollary: The inverse of the 4 color theorem does *not* hold:
If $\chi(G) \leq 4$, then G may still not be planar.

Proposition: If G contains C_n as a subgraph then $\chi(G) \geq n$.

Warning: Also here, the inverse is *not* true:
If $\chi(G) \geq n$, G need not contain C_n as a subgraph.

Examples:

i) An odd cycle needs 3 colors even if it does not contain C_3.

ii) A graph having a vertex where all neigbors form an odd cycle needs 4 colors even if it does not contain C_4.