

Algorithmics

Sebastian Iwanowski
FH Wedel

4. Graph algorithms

4.1 Minimal spanning trees as motivation for basic algorithms

Algorithmics 4

4.1 Minimal spanning Trees

Kruskal's Algorithm (simple variant):

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

- Start with an empty forest F consisting of no edge
 - Repeat for all edges e_1, e_2, \dots, e_m of G (edges are in sorted order):
 - Check if e_i may be inserted into F
 - such that F is still without circles;
 - If so, insert e_i into F ;
- until F consists of $n-1$ edges (let n be the number of vertices of G).

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

Exercise: Prove this theorem!

Time complexity: $O(m \log m + n^2)$ (n^2 due to determination of connectivity component)

How to do this smarter?

References for catching up and delving into:

Skript Diskrete Mathematik 6, Folien 2,3,4,8,11,12,13 (graph theoretic basics)

Turau Kap. 2.4 (Grundlagen), 3.6.1 (Kruskal)

Cormen ch. 23 (Minimal spanning trees)

Algorithmics 4

4.1 Basic algorithms for graph theory

Union-Find-Structure

In general: works on sets of sets,
implements efficient location of the set of a given element
and efficient union of sets

Graph theoretic application: efficient location and union of connectivity components

$O(\log n)$ **Find** (v) returns a unique reference node of the connectivity component of v .

$O(1)$ **Union** (v, w) unifies the connectivity components of v and w after reference node has been determined

With path compression:

Expected time complexity of Find is in $O(\log^*n)$

Data representation:

Array of nodes: The contents are pairs of the form (index of the root, distance to the root)

References:

Skript Alt, Kap. 3.2 (p. 56 ff.), Cormen ch. 21 (Data structures for disjoint sets)

Algorithmics 4

4.1 Basic algorithms for graph theory

Heap

Efficient management of a priority queue

Invariants:

- 1) A heap is a complete binary tree (elements may be missing only in the last depth level).
- 2) The keys of the children of each node are not less than the key of each node.

$O(\log n)$	<code>DeleteMin()</code>	deletes the minimal element of the heap.
$O(\log n)$	<code>Insert (v)</code>	inserts an arbitrary new element into the heap.
$O(1)$	<code>SearchMin()</code>	finds the minimal element of the heap.

Data representation:

Array of the heap nodes:

The contents are the contents of the heap nodes.

The children of the node with index i are the nodes with indices $2i$ und $2i+1$
(assuming that the array starts with index 1)

References:

Cormen, ch. 6 (Heapsort)

Algorithmics 4

4.1 Minimal spanning Trees

Kruskal's Algorithm (efficient variant):

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

- Start with an empty forest F consisting of no edge
- Start with a **union-find-structure** in which each vertex has its own connectivity component
- Insert all edges into a **heap**
- While F consists of less than $n-1$ edges:
 - Search and delete the minimal element e_{\min} from the heap;
 - Check if the vertices v and w incident with e_{\min} are in the same connectivity component
 - If not: Insert e_{\min} into F and unify the connectivity components of v and w .

Time complexity: $O(m \log m)$ (m is the number of edges in G)

References:

Cormen, ch. 23.2 (Algorithms of Kruskal and Prim)