Chapter 11 Formulas / Theorems sheet

[Def] A *tree* is an undirected graph that is connected and has no simple circuits (cycles).

[Theorem] An undirected graph is a tree iff there is a unique simple path between any two of its vertices.

[Def] A *rooted tree* is a tree in which one vertex has been designated as the *root* and every edge is directed away from the root.

[Def] The *level of a vertex* is its distance from to the root.

[Def] The *height of a tree* is the highest level of any vertex.

[Corollary] There is a unique path from the root of the tree to each vertex of the tree.

[Def] The tree is call a *full m-ary tree* if every internal vertex has exactly m children.

[Def] A *complete m-ary tree* is a *full m-ary tree* in which each leaf is at the same level.

[Theorem 2] A rooted tree with n vertices has $n-1$ edges

[Theorem 4] A *full m-ary tree* with

(1) ***n*** vertices has $i=\frac{n-1}{m}$ internal vertices, and $l=\frac{(m-1)n+1}{m}$ leaves;

(2) ***i*** internal vertices has $n = mi+1$ vertices, and $l = (m-1)i+1$ leaves;

(3) ***l*** leaves has $n=\frac{ml-1}{m-1}$ vertices, and $i=\frac{l-1}{m-1}$ internal vertices.

[Def] A rooted *m-ary tree* of height h is *balanced* if all leaves are at levels h or $h-1$.

[Theorem 5] There are at most m^h leaves in an *m-ary tree* of height h, i.e. $l \le m^h$

[Corollary]
1) For a *full* and *balanced m-ary tree* of height h with l leaves, $h = \lceil \log_m l \rceil$
2) For an *m-ary tree* of height h with l leaves, $h \ge \lceil \log_m l \rceil$.

Binary search tree (BST), is a binary tree with *keys* assigned to vertices/nodes, where every vertex/node v has the following property:
- each *key* in the left subtree is less than the *key* at the vertex v,
- each *key* in the right subtree is greater than the *key* at the vertex v.

length of the longest path in BST = height of the BST = $\begin{cases} \lceil \log_2(n+1) \rceil = \lceil \log(n+1) \rceil & \text{if a balanced tree} \\ n & \text{if an unbalanced tree} \end{cases}$

[**Def**] A rooted tree in which each internal vertex corresponds to a decision, with a subtree at these vertices for each possible outcome of the decision, is called a *decision tree*.
The *possible solutions of the problem* correspond to the paths to the leaves of this rooted tree.

[Def] *prefix codes*: letters are encoded in such a way that *the bit string for a letter never occurs as the first part of the bit string for another letter*.

[Def] Let G be a simple graph. A *spanning tree* of G is a subgraph of G that is a tree containing every vertex of G.

[Theorem 1] A simple graph is connected iff it has a *spanning tree*.

[Def] A *minimum spanning tree* in a connected weighted graph is a spanning tree that has the smallest possible sum of weights of its edges.

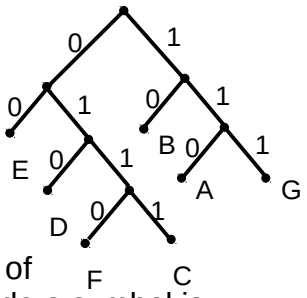
Huffman Coding

procedure *Huffman*(C: symbols a_i with freq. w_i, *i = 1,...,n*)
F := forest of *n* rooted trees, each contains a single vertex a_i and assigned weight w_i.
while F is not a tree
  Take two rooted trees, T and T', with the least weights and w(T) ≥ w(T').
  Replace them with a tree having a *new root* that has T as its left subtree and T' as its right subtree.
  Label the edge to T with 0, and the edge to T' with 1.
  Assign w(T)+w(T') as the weight of the new tree.

Output: Huffman code for a_i: the concatenation of labels of the edges in the unique path from the root to the vertex v_i.

Example: Use Huffman coding to encode the following symbols with the frequencies:

A: 0.10	110
B: 0.25	10
C: 0.05	0111
D: 0.15	010
E: 0.30	00
F: 0.07	0110
G: 0.08	111



the average number of bits required to encode a symbol is
3 * 0.10 + 2 * 0.25 + 4 * 0.05 + 3 * 0.15 + 2 * 0.3 + 4 * 0.07 + 3 * 0.08 = 2.57

Tree traversals:

preorder traversal :
- visit parent *r*, then
- traverse child T_1 in preorder,
- …
- traverse child T_n in preorder.

inorder traversal :
- traverse child T_1 in inorder,
- visit parent *r*,
- traverse child T_2 in inorder,
- …
- traverse T_n in inorder.

postorder traversal :
- traverse child T_1 in postorder,
- …
- traverse child T_n in postorder,
- visit parent *r*.

An *inorder traversal* of T (fully parenthesised) produces the expression in *infix notation*: $(-3)\sqrt{4}-(5+\frac{1}{4})$

The *preorder traversal* of T produces the expression in *prefix notation*: $-*(-3)\sqrt{}\,4+5\div 1\,4$ ←

The *postorder traversal* of T produces the expression in *postfix notation* / *Polish notation*:
$(-3)\,4\sqrt{}\,*5\;1\;4\div+-$ →

Depth-First Search:

procedure *DFS*(*G*: connected graph with vertices $v_1, v_2, \ldots, v_n$)
T := tree consisting only of vertex v_1
visit(v_1)

procedure *visit*(v: vertex of *G*)
for each vertex w adjacent to v and not yet in T,
  add vertex w and edge {v,w} to T
  visit(w)

Breadth-First Search:

procedure *BFS*(*G*: connected graph with vertices $v_1, v_2, \ldots, v_n$)
T := tree consisting only of vertex v_1
L := empty list
put v_1 in the list L of unprocessed vertices
while L is not empty
  remove the first vertex, v, from L
  for each neighbor w of v
    if w is not in L and not in T **then**
      add w to the end of the list L
      add w and edge {v,w} to T

Prim's algorithm:

procedure *Prim*(*G*: weighted connected undirected graph with *n* vertices)
T := a minimum weight edge
for *i* := 1 to *n-2*
  e := an edge of min. weight incident to a vertex in T and not forming a simple circuit in T if added to T
  T := T with added edge *e*
return T

We will assume that the edges are ordered when we need to choose between two or more edges with the same weights.

Kruskal's algorithm:

procedure *Kruskal*(*G*: weighted connected undirected graph with *n* vertices)
T:= empty graph
for *i* := *1* to *n-1*
  e:= any edge in *G* with smallest weight that does not form a simple circuit when added to T
  T := T with added *e*
return T