```
/**
 * Contains old code from nodefactory.js (GRO)
 * Code for traversing:
 * https://code.tutsplus.com/articles/data-structures-with-javascript-tree--cms-23393 **/

// node-Konstruktor
// var node = function () {
function node() {
    this.id = -1;
    this.parent = 0;
    this.children = [];
    this.type = 'space';
    this.content = '';
    this.comes_from = 1; //above
    this.way_back = false;
}
;

node.prototype.insertMeOver = function (insertPointId, leaf, nodelist) {
    // insertMeOver <-> JAVA: void InsertUnodeOverCursor(Node u)
    // this (me) <-> JAVA: Node u
    // insertPoint <-> JAVA: Node Cursor
    var insertPoint = nodelist[insertPointId];
    if (this.children.length === 0) {
        this.children.length = 2;
        // console.log('Insert me (leaf) over insertpoint is not possible.');
        // do nothing
    }
    if (this.children.length >= 1) {
        // kindof me = unode. parameter leaf not necessary
        var parent = nodelist[insertPoint.parent];
        // parent.children[insertPoint.position] = this.id;
    }
    this.parent = insertPoint.parent;
    if (this.children.length === 1) {
        this.children = [insertPointId];
    }
    if (this.children.length === 2) {
        this.children = [insertPointId, leaf.id];
    }
    insertPoint.parent = this.id;
    nodelist[insertPointId] = insertPoint;
    return 'ok';
};
node.prototype.addBracket = function (tree) {
    var temp = find_leftmost_bracket(this.content);
    var left_pos = temp[0];
    var bra = temp[1];
    temp = find_corresponding_right_bracket(this.content, bra);
    var left_pos2 = temp[0];
    var bra_len = temp[1];
    var right_pos = temp[2];
    var rightbra_len = temp[3];
```

```
    // this should not happen
    if (left_pos !== left_pos2) {
        throw 'Inconsistent left positions ';
    }
    if (left_pos > -1) {
        var leftpart = this.content.substring(0, left_pos);
        var middlepart = this.content.substring(left_pos + bra_len, right_pos);
        var rightpart = this.content.substring(right_pos + rightbra_len);
        this.content = leftpart + '§' + rightpart;
        var bracket = create_node('bracket-' + bra, '', tree);
        var middle = create_node('unknown leaf', middlepart, tree);
        // first connection
        this.children.push(bracket.id);

        bracket.parent = this.id;
        // second connection
        bracket.children.push(middle.id);
        middle.parent = bracket.id;
    } // else no bracket found, do nothing
    return left_pos;
};
node.prototype.debug = function () {
    var text = this.id + ': parent=' + this.parent;
    // var text = this.id;
    text += ' children=' + this.children;
    text += ' type=' + this.type;
    text += ' content=' + this.content;
    // text += ' rightmost=' + this.isRightmostChild(nodelist);
    if (this.type.startsWith('unknown')) {
    } else {
        // text = '';
    }
    console.log(text);
    return text;
};

function tree() {
    this.list_of_free = [];
    this.nodelist = [];
    this.nodelist[0] = new node();
    console.log('len nodelist ' + this.nodelist.length);
    this.root = this.nodelist[0];
    this.root.type = 'root';
    this.root.id = 0;
    this.root.parent = -1;
    // this.leaf = create_node('unknown leaf content', 'my first leaf', this.nodelist);
    this.leaf = new node();
    this.leaf.type = 'unknown leaf content';
    this.leaf.content = 'my first leaf';
    this.nodelist[1] = this.leaf;
    this.leaf.id = 1;
    this.leaf.parent = this.root.id;
    this.root.children = [this.leaf.id];
```

```
    console.log('len nodelist ' + this.nodelist.length);
    console.log('tree ' + this.root.id + ' ' + this.leaf.id);
}
;
function create_node(type, content, tree) {
    var nodelist = tree.nodelist;
    var lof = tree.list_of_free || [];
    if (lof.length === 0) {
        temp = new node();
        temp.type = type;
        temp.content = content;
        nodelist.push(temp);
        temp.id = nodelist.length - 1;
        return temp;
    } else {
        var last_free = lof.pop();
        temp = nodelist[last_free];
        temp.type = type;
        temp.content = content;
        console.log('recycling ' + last_free);
        return temp;
    }
}
;
function delete_single_nodes(tree) {
    // delete $ nodes
    var list_of_nodes = tree.nodelist;
    for (var i = 0; i < list_of_nodes.length; i++) {
        var node = list_of_nodes[i];
        if (node.content === '$' && node.children.length === 1) {
            console.log('found single $ node at ' + node.id);
            var siblings = list_of_nodes[node.parent].children;
            var position = siblings.indexOf(node.id);
            console.log('position=' + position);
            console.log('siblings[position]=' + siblings[position]);
            // short circuit
            siblings[position] = node.children[0];
            list_of_nodes[node.children[0]].parent = node.parent;
            tree.list_of_free.push(node.id);
        }
    }
    return tree.list_of_free;
}
;
node.prototype.isRightmostChild = function (nodelist) {
    if (this.id === 0) {
        return false;
    } else {
        var siblings = nodelist[this.parent].children;
        var rightmost = siblings[siblings.length - 1];
        var isRightmost = (this.id === rightmost);
        return isRightmost;
    }
```

```
};

function find_left_bracket(node, bra) {
    var long = '\\left' + bra;
    if (bra === '{') {
        long = '\\left\\{';
    }
    var min_pos = -1;
    var bra_kind = 'nothing';
    var pos = node.indexOf(long);
    var masked = node;
//      console.log('Start searching ' + bra + ' in ' + masked);
    if (pos >= 0) {
        min_pos = pos;
        bra_kind = long;
        // mask all occurrencies of long
        var stop = false;
        pos = -1;
        do {
            // console.log(masked);
            var pos = masked.indexOf(long, pos + 1);
            // console.log('found ' + long + ' at ' + pos);
            if (pos === -1) {
                stop = true;
            } else {
                // console.log('found ' + long + ' at ' + pos);
                if (pos > 0) {
                    var part1 = masked.substring(0, pos - 1);
                } else {
                    var part1 = '';
                }
                var part2 = '\\left@';
                if (bra === '\\{') {
                    part2 = '\\left\\@';
                }
                var part3 = masked.substring(pos + long.length);
                masked = part1 + part2 + part3;
                // console.log('masked:' + masked);
            }
        } while (stop === false);
    }
    // All occurrencies of long are masked
    // Look for short bracket
    var pos = masked.indexOf(bra);
    if (pos >= 0) {
        if (min_pos === -1) {
            min_pos = pos;
            bra_kind = bra;
        } else {
            if (pos < min_pos) {
                min_pos = pos;
                bra_kind = bra;
            }
```

```
        }
    }
//    console.log('Result for ' + bra + ': Found ' + bra_kind + ' at ' + min_pos);
    return [min_pos, bra_kind];
}

function find_leftmost_bracket(node) {
    var left_pos = -1;
    var bra_kind = 'nothing';
    var result = find_left_bracket(node,"(");
    var pos = result[0];
    if (pos > -1) {
        if (left_pos === -1) {
            left_pos = pos;
            bra_kind = result[1];
        } else {
            if (pos < left_pos) {
                left_pos = pos;
                bra_kind = result[1];
            }
        }
    }
// maybe there is a better (smaller) pos for a [ bracket
    var result = find_left_bracket(node,'[');
    var pos = result[0];
    if (pos > -1) {
        if (left_pos === -1) {
            left_pos = pos;
            bra_kind = result[1];
        } else {
            if (pos < left_pos) {
                left_pos = pos;
                bra_kind = result[1];
            }
        }
    }
// maybe there is a better (smaller) pos for a { bracket
    var result = find_left_bracket(node,'{');
    var pos = result[0];
    if (pos > -1) {
        if (left_pos === -1) {
            left_pos = pos;
            bra_kind = result[1];
        } else {
            if (pos < left_pos) {
                left_pos = pos;
                bra_kind = result[1];
            }
        }
    }
    return [left_pos, bra_kind];
}
```

```
function find_corresponding_right_bracket(node, bra) {
    var rightbra = '';
//     console.log('look for ' + bra + ' in ' + node);
    var pos = ['(','[','{','\\left(','\\left[','\\left\\{'].indexOf(bra);
//     console.log('pos=' + pos);
    if (pos === -1) {
        var rightbra = 'no corresponding bracket found error';
    } else {
        var rightbra = [')',']','}','\\right)','\\right]','\\right\\}'][pos];
    }
//     console.log('rightbra=' + rightbra);
    var stop = false;
    var mass = [];
    for (var i = 0; i < node.length; i++) {
        mass[i] = 0;
    }
    var left_pos = -1;
    var right_pos = -1;
    pos = -1;
    do
    {
        var pos = node.indexOf(bra, pos + 1);
//         console.log(pos);
        if (pos === -1) {
            stop = true;
        } else {
            mass[pos] = 1;
//             console.log('mass=1 at ' + pos);
            if (left_pos === -1) {
                left_pos = pos;
            }
        }
    } while (stop === false);
    pos = -1;
    stop = false;
    do
    {
        var pos = node.indexOf(rightbra, pos + 1);
//         console.log('rightbra pos=' + pos + ' in ' + node);
        if (pos === -1) {
            stop = true;
        } else {
            mass[pos] = -1;
//             console.log('mass=-1 at ' + pos);
        }
    } while (stop === false);
    // sum of masses
    for (var i = 1; i < node.length; i++) {
        var sum = mass[i - 1] + mass[i];
        if (mass[i] === -1 && sum === 0) {
            right_pos = i;
            break;
        }
```

```
        mass[i] = sum;
//          console.log('mass[' + i + ']=' + sum);
    }
    return [left_pos, bra.length, right_pos, rightbra.length];
}

function remove_operators(tree, kind_of_operators) {
    var index = 1;
    var stop = false;
    var pos = -1;
    var op_one = '+';
    var op_two = '-';
    if (kind_of_operators === 'timesdivided') {
        op_one = '\\cdot';
        op_two = ':';
    }
// before power, \int has to be parsed
    if (kind_of_operators === 'power') {
        op_one = '^';
        op_two = '@%';
    }
// before sub, \int has to be parsed
    if (kind_of_operators === 'sub') {
        op_one = '_';
        op_two = '@%';
    }
    var op_one_len = op_one.length;
    var op_two_len = op_two.length;
    do {
        var node = tree.nodelist[index];
        // console.log(index + ': ' + node.content);
        var omitted = true;
        if (node.type.startsWith('unknown')) {
            var pos_one = node.content.lastIndexOf(op_one);
            var pos_two = node.content.lastIndexOf(op_two);
            var pos_one_flag = false;
            if (pos_one === -1 && pos_two === -1) {
                pos = -1;
            } else {
                if (pos_one === -1) {
                    pos = pos_two;
                }
                if (pos_two === -1) {
                    pos = pos_one;
                    pos_one_flag = true;
                }
                if (pos_one > -1 && pos_two > -1) {
                    pos = pos_one;
                    pos_one_flag = true;
                    if (pos_two > pos) {
                        pos = pos_two;
                        pos_one_flag = false;
                    }
```

```
            }
        }

        if (pos === -1) {
            index++;
        } else {
            // found an operator op_one or op_two in node[index]
            // console.log(index + ': ' + node.content + ' pos=' + pos);
            var leftpart = node.content.substring(0, pos);
            if (pos_one_flag) {
                var middlepart = node.content.substring(pos, pos + op_one_len);
                var rightpart = node.content.substring(pos + op_one_len);
            } else {
                var middlepart = node.content.substring(pos, pos + op_two_len);
                var rightpart = node.content.substring(pos + op_two_len);
            }
            // number of § markers
            var leftcount = (leftpart.match(/§/g) || []).length;
            var rightcount = (rightpart.match(/§/g) || []).length;
//                  console.log('leftpart=' + leftpart + ' leftcount=' + leftcount);
//                  console.log('rightpart=' + rightpart + ' rightcount=' + rightcount);
//                  console.log('# of children=' + node.children.length || 0);
            var check = ((leftcount + rightcount) === node.children.length);
            if (node.type.startsWith('definite')) {
                // children[0] = lower_boundary, children[1] = upper_boundary
                check = ((leftcount + rightcount) === node.children.length - 2);
            }
            if (check === false) {
                ///throw('(remove operators) Wrong number of bracket markers');
                console.log('(remove operators) Wrong number of bracket markers');
            }
            var rememberchildren = node.children;
            if (leftcount > 0) {
                var leftchildren = rememberchildren.slice(0, leftcount);
            } else {
                var leftchildren = [];
            }
            if (rightcount > 0) {
                var rightchildren = rememberchildren.slice(leftcount, rememberchildren.length);
            } else {
                var rightchildren = [];
            }
            var operator = create_node('plusminus', middlepart, tree);
            if (kind_of_operators === 'timesdivided') {
                operator.type = 'timesdivided';
            }
            if (kind_of_operators === 'power') {
                operator.type = 'power';
            }
            if (kind_of_operators === 'sub') {
                operator.type = 'sub';
            }
            var rest = create_node('unknown leaf', rightpart, tree);
```

```
            var siblings = tree.nodelist[node.parent].children;
            var position = siblings.indexOf(node.id);
//              console.log('position=' + position);
//              console.log('siblings[position]=' + siblings[position]);
//              console.log('node.id=' + node.id);

            // Upper connection: connect new node operator with former parent of node
            tree.nodelist[node.parent].children[position] = operator.id;
            operator.parent = node.parent;
            // Left and right connection:
            // connect new node operator at left side with old node, but left part only
            // connect new node operator at right side with new node rest
            // Direction "up"
            node.content = leftpart;
            node.parent = operator.id;
            rest.parent = operator.id;
            // Direction "down"
            operator.children = [node.id, rest.id];
            // children of node and rest have to be adjusted
            node.children = leftchildren;
            rest.children = rightchildren;
        }
    } else {
        // omit if type is not 'unknown'
        index++;
    }
    if (index > tree.nodelist.length - 1) {
        stop = true;
    }
  } while (stop === false);
  return tree.nodelist;
}
;
```