```
'use strict'; //jshint node:true
var has = {}.hasOwnProperty
  , slice = [].slice

exports.get = get
function get(obj, key, fallback) {
  return has.call(obj, key)
    ? obj[key]
    : fallback
}

exports.getIn = getIn
function getIn(obj, keys, fallback) { keys = keys.slice()
  var key = keys.shift()
  return keys.length
    ? getIn(get(obj, key, {}), keys, fallback)
    : get(obj, key, fallback)
}

exports.assoc = assoc
function assoc(obj, key, value) {
  return arguments.length === 3
    ? assocM(clone(obj), key, value)
    : assoc.apply(null, arguments)
}

assoc.apply = function(_, args) {
  var obj = args[0]
    , ret = clone(obj)
  for (var i = 1, len = args.length; i < len; i += 2)
    assocM(ret, args[i], args[i + 1])

  if (i !== len)
    throw new Error('missing key')

  return ret
}

exports.assocIn = assocIn
function assocIn(obj, keys, value) { return assocIn_(obj, keys.slice(), value) }
function assocIn_(obj, keys, value) {
  var key = keys.shift()
  return keys.length
    ? assoc(obj, key, assocIn_(obj[key] || {}, keys, value))
    : assoc(obj, key, value)
}

exports.dissoc = dissoc
function dissoc(obj, key) {
  if (arguments.length === 1)
    return obj
  else if (arguments.length <= 4) {
    var ret = {}
      , keyA = key  ? (key  + '') : ''
      , keyB = keyB ? (keyB + '') : ''
      , keyC = keyC ? (keyC + '') : ''
    for (var curKey in obj) if (has.call(obj, curKey))
      if (curKey !== keyA && curKey !== keyB && curKey !== keyC)
        ret[curKey] = obj[curKey]
    return ret
  }
  else
    return dissoc.apply(null, arguments)
}

dissoc.apply = function(_, args) {
  var obj = clone(args[0])
  for (var i = 1, len = args.length; i < len; i++)
    dissocM(obj, args[i])
  return obj
}

exports.merge = merge
function merge(obj, src) {
  return reduce(_mergeM, {}, slice.call(arguments))
}

exports.zipmap = zipmap
```

```
function zipmap(keys, vals) {
  var ret = {}
  for (var i = 0, len = keys.length; i < len; i++)
    ret[keys[i]] = vals[i]
  return ret
}

exports.selectKeys = selectKeys
function selectKeys(obj, keys) {
  var ret = {}
  for (var i = 0, len = keys.length; i < len; i++) {
    var key = keys[i]
    if (has.call(obj, key))
      ret[key] = obj[key]
  }
  return ret
}

var keys = exports.keys = Object.keys || function keys(obj) {
  var ret = []
  for (var key in obj) if (has.call(obj, key))
    ret.push(key)
  return ret
}

exports.vals = vals
function vals(obj) {
  var ret = []
  for (var key in obj) if (has.call(obj, key))
    ret.push(obj[key])
  return ret
}

exports.keyvals = keyvals
function keyvals(obj) {
  var ret = []
  for (var key in obj) if (has.call(obj, key))
    ret.push([key, obj[key]])
  return ret
}

exports.hashmap = hashmap
function hashmap() { return hashmap.apply(null, arguments) }

hashmap.apply = function(_, keyvals) {
  var ret = {}
  for (var i = 0, len = keyvals.length; i < len; i += 2)
    assocM(ret, keyvals[i], keyvals[i + 1])

  if (i !== len)
    throw new Error('missing key')

  return ret
}

// mutations ahead!

exports.assocInM = assocInM
function assocInM(obj, keys, value) {
  var ret = obj
    , key

  for (var i = 0, len = keys.length; i < (len - 1); i++) {
    key = keys[i]
    obj = obj[key] || (obj[key] = {})
  }

  key = keys[i]
  obj[key] = value

  return ret
}

exports.assocM = assocM
function assocM(obj, key, value) {
  if (arguments.length === 3) {
    obj[key] = value
    return obj
  }
  return assocM.apply(null, arguments)
}

assocM.apply = function(_, args) {
  var obj = args[0]
  for (var i = 1, len = args.length; i < len; i += 2)
    assocM(obj, args[i], args[i + 1])

  if (i !== len)
    throw new Error('missing key')

  return obj
}

exports.dissocM = dissocM
function dissocM(obj, key) {
  if (arguments.length > 2)
    return dissocM.apply(null, arguments)
  delete obj[key]
```

```
  return obj
}

dissocM.apply = function(_, args) {
  var obj = args[0]
  for (var i = 1, len = args.length; i < len; i++)
    delete obj[args[i]]
  return obj
}

exports.mergeM = mergeM
function mergeM(obj, src) {
  return arguments.length === 2
    ? _mergeM(obj, src)
    : reduce(_mergeM, obj, slice.call(arguments, 1))
}

function _mergeM(obj, src) {
  for (var key in src) if (has.call(src, key))
    obj[key] = src[key]
  return obj
}

function reduce(fn, initial, values) {
  var acc = initial
  for (var i = 0, len = values.length; i < len; i++)
    acc = fn(acc, values[i])
  return acc
}

function clone(obj) {
  var ret = {}
  for (var key in obj) if (has.call(obj, key))
    ret[key] = obj[key]
  return ret
}
```