Dedux

Setup

To use dedux you need to simply wrap your root reducer with the Dedux object.

```
const Dedux = require('redux-dedux');
const rootReducer = (state, action) => {
  switch(action.type) {
    default:
      return state;
  }
}
const store = createStore(Dedux(rootReducer));
```

TODOS:

- where clause should probably be strings to preserve serialization of actions.

Major features

- Common Actions for Booleans, String, Numbers, Arrays, and Objects.
- The Array and Object collections expose actions from the type of items they contain.
    - Example: 'SET', 'INCREMENT', and 'DECREMENT' can be used on an Array of Numbers to manipulate the Numbers in the Array.

    - Key, index, and where properties can be added to actions to deal with array and object operations.
- Object keys are available for reference in the action type.
    - Example: You can access the todos array in `{todos: ['Make list', 'Review list']}` using {Action}_TODOS. E.G: 'ADD_TO_TODOS', 'REMOVE_FROM_TODOS'.
    - This is to aid the use of objects to organize state.

Common Actions

The action types that are available to you as a user depend on the top-level object in your Redux state. The most common is an object, but it's also possible to have arrays, strings, numbers, and booleans as your state.

Number

SET(value)
INCREMENT(value)
DECREMENT(value)

Boolean

SET(value)
TOGGLE

String

SET(value)

ARRAY

ADD(value)
CONCAT(array)
SET_ALL(value)
SET(value, [index, where])
INSERT(value, index)
REMOVE_ALL()
REMOVE(value, [index, where])

ARRAY_OF[Booleans, Numbers, String]

Boolean
TOGGLE_ALL()
TOGGLE([index, where])
Number
INCREMENT_ALL(value)
INCREMENT(value, [index, where])
DECREMENT_ALL(value)
DECREMENT(value, [index, where])
String

ARRAY_OF[Objects]

MERGE_ALL(obj)
MERGE(obj, [key, where])

OBJECT

ADD(value, key)
MERGE(value, [key, where])
REMOVE([key, value])

OBJECT_PATHS_OF[Booleans, Numbers,

Strings]

Boolean

SET_{PATH}(value)

TOGGLE_{PATH}()

Number

SET_{PATH}(value)

INCREMENT_{PATH}(value)

DECREMENT_{PATH}(value)

String

SET_{PATH}(value)

OBJECT_ARRAY

ADD_TO_{PATH}(value, [index, where])

REMOVE_ALL_{PATH}()

REMOVE_FROM_{PATH}([index, where])

UPDATE_ALL_{PATH}(value)

UPDATE_{PATH}(value, [index, where])

SET_ALL_{PATH}(value)

SET_{PATH}(value, [index, where])

TOGGLE_ALL_{PATH}()

TOGGLE_{PATH}([index, where])

INCREMENT_ALL_{PATH}()

INCREMENT_{PATH}([index, where])

DECREMENT_ALL_{PATH}()

DECREMENT_{PATH}([index, where])

OBJECT_OBJECT

// Most likely used for normalization, the path syntax should take care of objects used for organization.

ADD(value) -- auto-increment option?

MERGE_ALL(value)

MERGE(value, [key, where])

REMOVE_ALL()

REMOVE(value, [key, where])

Number

SET

```
{
  type: 'SET',
  value: 3
}
```

```
0 -> 3
```

INCREMENT/DECREMENT

```
{
  type: 'INCREMENT',
  value: 4
}
```

```
5 -> 9
```

```
{
  type: 'DECREMENT',
  value: 3
}
```

```
5 -> 2
```

Boolean

SET

```
{
  type: 'SET',
  value: true
}
```

```
false -> true
```

TOGGLE

```
{
  type: 'TOGGLE',
}
```

```
true -> false
```

String

SET

```
{
  type: 'SET',
  value: 'new string'
}
```

```
'old string' -> 'new string'
```

Array of Numbers

ADD

```
{
  type: 'ADD',
  value: 3
}
```

```
[1, 2] -> [1, 2, 3]
```

SET

```
{
  type: 'SET',
  value: 5
  index: 1
}
```

```
[1, 2, 3] -> [1, 5, 3]
```

SET_ALL

```
{
  type: 'SET_ALL',
  value: 0
}
```

```
[1, 2, 3] -> [0, 0, 0]
```

INSERT

```
{
  type: 'INSERT',
  index: 2
}
```

```
[1, 2, 4] -> [1, 2, 3, 4]
```

UPDATE

```
{
  type: 'UPDATE',
  value: 3,
  where: (elem) => elem === 4
}
```

```
[1, 2, 4] -> [1, 2, 3]
```

```
{
  type: 'UPDATE',
  value: {id: 2, value: 2},
  where: (elem) => elem.id === 2
}
```

```
[{ id: 1, value: 1 }, { id: 2, value: 3 }] ->
[{ id: 1, value: 1 }, { id: 2, value: 2 }]
```

REMOVE

```
{
  type: 'REMOVE',
  where: (val) => val === 4
}
```

```
[1, 2, 4] -> [1, 2]
```

Array of Numbers

INCREMENT/DECREMENT

```
{
  type: 'INCREMENT',
  value: 3,
  index: 1
}
```

```
[1, 2, 3] -> [1, 5, 3]
```

INCREMENT/DECREMENT ALL

```
{
  type: 'INCREMENT_ALL',
  value: 1
}
```

```
[1, 2, 3] -> [2, 3, 4]
```

Array of Booleans

TOGGLE

```
{
  type: 'TOGGLE',
  index: 2
}
```

```
[true, true, true] -> [true, false, true]
```

TOGGLE_ALL

```
{
  type: 'TOGGLE_ALL',
}
```

```
[true, false, true] -> [false, true, false]
```

Array of Objects

ADD with auto-incrementing ID.

Objects

Dedux action types for objects employ a special **'PATH'** and **'ITEM_ACTION'** syntax to allow you to reference keys within nested objects.

An **ITEM_ACTION** is an action type that can be run on an item inside your object. These might be action types such as 'SET' or 'TOGGLE'.

A **PATH** is a sequence of keys used in your objects that allow you to uniquely identify the location of a value in your store.

For example:

```
{a:1, b:2, c:3}
```

The following paths available: 'A', 'B', and 'C'. This allows you to use action types such as 'SET_A' and 'SET_B'. In general, this allows you to use objects to organize your state.

A more complicated example:

```
{
  counters: {c1: 0, c2: 0, c3: 0},
  togglers: {'a': true, 'b': false}
}
```

The paths 'C1', 'C2', 'C3', 'A', and 'B' are available and allow you to use an action such as 'INCREMENT_C1' or 'TOGGLE_B'. #Check if this use case adds unnecessary complexity. Implement full paths first.

You can also use the full path: 'INCREMENT_COUNTERS_C1'. This is useful when you have multiple keys with the same name.

It's tempting to use this syntax when you have an array or object of similar objects, but you should use the 'UPDATE' action type with the where property instead.

In many cases you won't know the exact key until a user performs an action in your appliction. For these cases you can use the key property inside your action:

```
{
  type: 'INCREMENT_COUNTERS',
  value: 1,
  key: 'c1'
}
```

Here are some common uses of **ITEM_ACTIONS** and **PATHS**.
{ITEM_ACTION}_{PATH}

```
{
  type: 'SET_A',
  value: 'new string'
}
```

```
{a: 'old', b: 'old'} -> {a: 'new string', b: 'old'}
```

```
{
  type: 'ADD_TODOS',
  value: { id: 2, text: 'A new todo' }
}
```

```
{
  todos: [
    { id: 1, text: 'Make todo list' }
  ]
} ->
{
  todos: [
    { id: 1, text: 'Make todo list' },
    { id: 2, text: 'A new todo' }
  ]
}
```

```
{
  type: 'UPDATE_TODOS',
  value: { completed: true },
  where: (elem) => elem.id === 1
}
```

```
{
  todos: [
    { id: 1, text: 'Make todo list', completed: false }
  ]
} ->
{
  todos: [
    { id: 1, text: 'Make todo list', completed: true }
  ]
}
```