Systematic Conversion of Memoization to Dynamic Programming

Consider this memoizing function that computes the number of different ways a set of coins can total to a given amount:

```
function change(amt,coins,index,size)
    {
    if (amt == 0) return 1;
    if (amt < 0) return 0;
    if (index == size) return 0;

    /* check if we have previously computed the value */
    if (table[amt][index] != EMPTY)
        return table[amt][index];

    /* compute new values, with using the coin and without
    with = change(amt-coins[index],coins,index,size);
    without = change(amt,coins,index+1,size);

    /* save the newly computed result */
    table[amt][index] = with + without;

    return result;
    }
```

A systematic conversion follows these steps:

- Identify all the variables that never change over all the recursive calls.
- Remove those variables from the recusive calls.

  Example

  The recursive call `change(amt-coins[index],coins,index,size)` becomes `change(amt-coins[index],index)`, assuming *coins* and *size* do not ever change over all recursive calls.

- Change the remaining variables in the recursive calls to loop variables. Do the same for the table update and the base cases.

  Example

  The call `change(amt-coins[index],index)` becomes `change(a-coins[i],i)`, assuming *a* is a variable that will loop over all values of *amt* and *i* is a variable that will loop over all values of *index*. The table update `table[amt][index] = with + without` becomes `table[a][i] = with + without`. The base case `if (amt < 0) ...` becomes `if (a < 0) ...`.

- Convert recursive calls to calls to a table lookup function (this in preparation for dealing with base cases in the original function - see

below).

Example

The call `change(a,i+1)` becomes `getTable(a,i+1)`. At the beginning, *getTable* looks like:

```
function getTable(a,i)
    {
    return table[a][i];
    }
```

- Surround the table update and lookups with loops that loop over the loop variables. The loop variables must reach the original values of the formal variables with which they are associated. Moreover, if a table lookup references a table value with a reduced value of the loop variable, the loop for that variable must run from low to high. On the other hand, if a table lookup references a table value with an increased value of the loop variable, the loop for that variable must run from high to low.

  Example

  The statements:

  ```
  with = getTable(a-coins[i],i);
  without = getTable(a,i+1);
  table[a][i] = with + without;
  ```

  become:

  ```
  for (a = 0; a <= amt; ++a)
      for (i = size-1; i >= 0; --i)
          {
          with = getTable(a-coins[i],i);
          without = getTable(a,i+1);
          table[a][i] = with + without;
          }
  ```

  Note that the *a* loop goes from low to high since the table lookups reference indices less than or equal to *a*. The *i* loop goes from high to low since the table lookups reference indices greater than or equal to *i*.

- Use the base cases to handle the results of exceptional table lookups. Do this by moving the base cases to the lookup function. Be sure to modify the definition of and the calls to the lookup function if additional information needs to be passed in to handle the base cases.

  Example

  The table lookup function:

  ```
  function getTable(a,i)
      {
      return table[a][i];
      }
  ```

  becomes:

  ```
  function getTable(a,i,size)
      {
      if (a == 0) return 1;
      if (a < 0) return 0;
      if (i == size) return 0;
  ```

```
return table[a][i];
}
```

Note that we had to modify the formal parameters of *getTable* in order to reference *size*.

- Finally, remove the original table lookup, and return the table value at the values passed to the function.

The final result is:

```
function change(amt,coins,index,size)
    {
    var a,i;

    for (a = 0; a <= amt; ++a)
        for (i = size-1; i >= 0; --i)
            {
            with = getTable(a-coins[i],i);
            without = getTable(a,i+1);
            table[a][i] = with + without;
            }

    return table[amt][index];
    }
```