Pass-LLVM-Value-Numbering

> Writing a pass to find all the LVN-identifiable redundancy. Project 2 from CS201 UCR.

Setup(From HelloPass)

1. Go to the test folder, and use the command below:

```
./run.sh
```

Code Explanation

1. The implemented Pass extends from `FunctionPass` class and overrides `runOnFunction(Function &F)` function. And it basically finds the redundancy expression in basic block.
2. In this implement, I used two hash_table, the first one is to store the operand of each expression, the second one is to store the expression in string format.

    ```
    std::map<llvm::Value*, int> hashmap;
    std::map<std::string, int> expressionmap;
    ```

3. For hashmap which stores the operands. There is a function to judge whether this operand in the hashmap or not. And for each operand, it'll have one unique value number.

    ```
      int searchHash(Value* v, bool found){
        auto search = hashmap.find(v);
        auto temp = vn;
        // errs() << v;
        if(search != hashmap.end()){
          temp = search->second;
          found = true;
          errs() << "true\t";
        }
        else{
          found = false;
          errs() << "false\t";
          hashmap.insert(make_pair(v, vn++));
        }
        return temp;
      }
    ```

4. Also, there would be the situation that the expression is same but the operands are in different places. So I swap these two value numbers. And use the value numbers and the operation symbol to represent the expression. Also becaues we want to output the right expression, so I use two temps to represent the origin operands.

```
    op_1 = searchHash(op1, found);
    op_2 = searchHash(op2, found);

    // errs() << to_string(op_1) << "\t" << to_string(op_2) << "\n";
    op_temp_1 = op_1;
    op_temp_2 = op_2;

    if(op_1 > op_2){
      swap(op_temp_1, op_temp_2);
    }

    string expression = to_string(op_temp_1) + operation +
to_string(op_temp_2);
```

5. If the expression is in the expression map, it will return the value number of this expression. If not, it will insert to the expression map with the string "expression" and the value number.

```
  int searchExpression(string exp, bool *found){
    auto search = expressionmap.find(exp);
    auto temp = vn;
    // vn_expression = vn;
    // errs() << "\n" << exp << "\n";
    if(search != expressionmap.end()){
      temp = search->second;
      *found = true;
      // errs() << "true\t";
    }
    else{
      expressionmap.insert(make_pair(exp, vn++));
      *found = false;
      // errs() << "false\t";
    }
    return temp;
  }
```

6. Also we need to store the value number of the destination value number which is in the left of the expression. What I do is to use the value number of the expression to represent this value. If not in the hashmap where store all the values appear in the BB, it will insert the value with the number which is the expression value number.

```
    op_expression = searchExpression(expression, found);

    op3 = dyn_cast<Value>(&inst);
    auto search_expression = hashmap.find(op3);
    if(search_expression != hashmap.end()){
      op_3 = search_expression->second;
    }
    else{
      hashmap.insert(make_pair(op3, op_expression));
      op_3 = op_expression;
    }
```

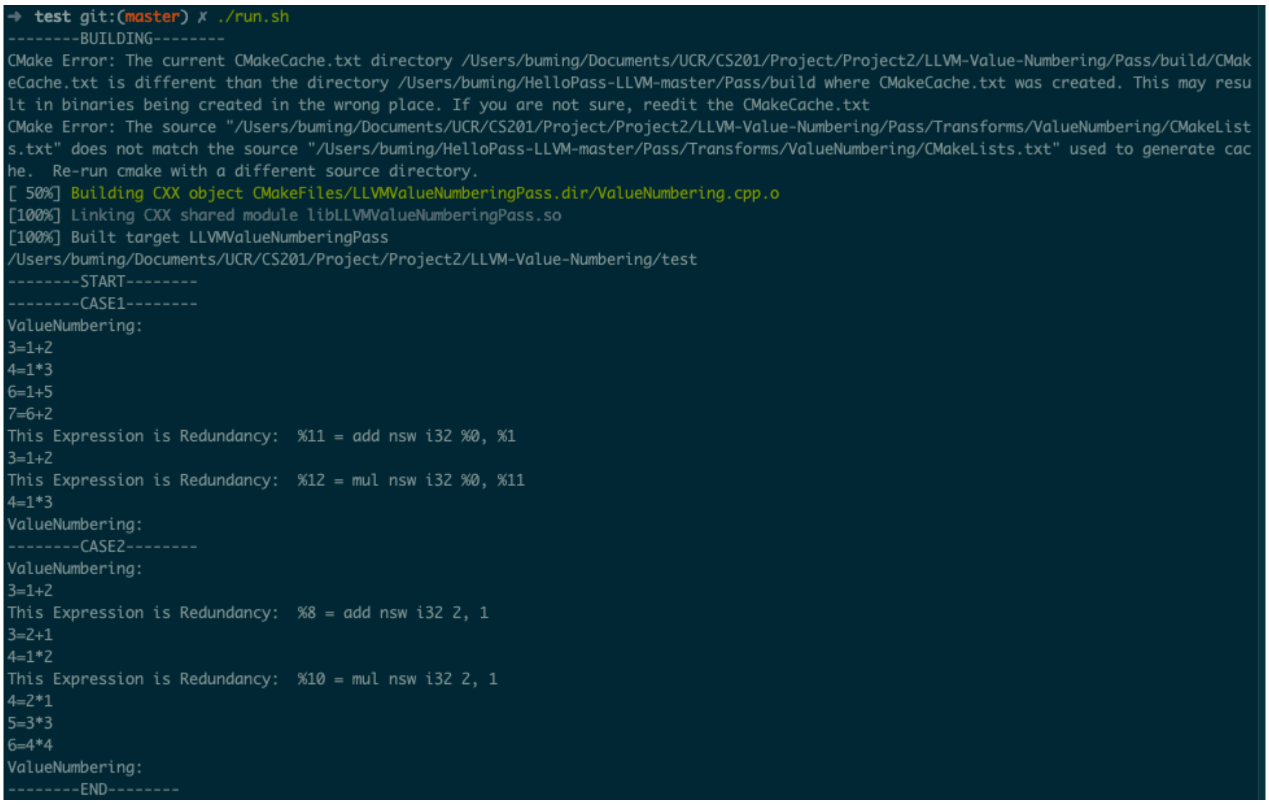
7. Finally, output the expression using the code below:

```
outfile << op_3 << "=" << op_1 << operation << op_2 << "\n";
```

Tests

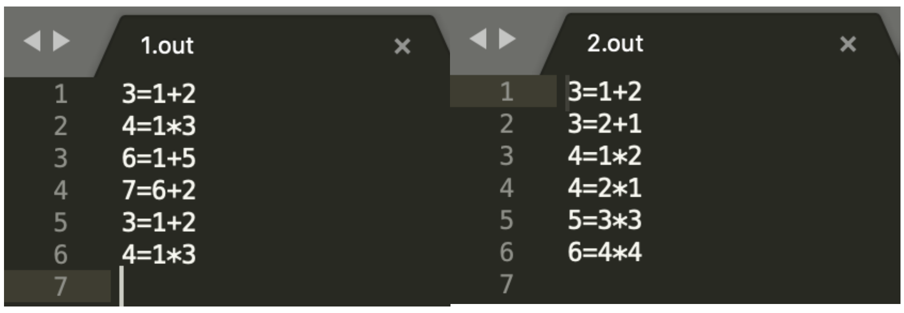
Run the script in the test folder. We can get two result, one is in the terminal, the other is the output under the same folder.

1. In the terminal, we can see which expression is redundancy.



```
➜ test git:(master) ✗ ./run.sh
--------BUILDING--------
CMake Error: The current CMakeCache.txt directory /Users/buming/Documents/UCR/CS201/Project/Project2/LLVM-Value-Numbering/Pass/build/CMak
eCache.txt is different than the directory /Users/buming/HelloPass-LLVM-master/Pass/build where CMakeCache.txt was created. This may resu
lt in binaries being created in the wrong place. If you are not sure, reedit the CMakeCache.txt
CMake Error: The source "/Users/buming/Documents/UCR/CS201/Project/Project2/LLVM-Value-Numbering/Pass/Transforms/ValueNumbering/CMakeList
s.txt" does not match the source "/Users/buming/HelloPass-LLVM-master/Pass/Transforms/ValueNumbering/CMakeLists.txt" used to generate cac
he.  Re-run cmake with a different source directory.
[ 50%] Building CXX object CMakeFiles/LLVMValueNumberingPass.dir/ValueNumbering.cpp.o
[100%] Linking CXX shared module libLLVMValueNumberingPass.so
[100%] Built target LLVMValueNumberingPass
/Users/buming/Documents/UCR/CS201/Project/Project2/LLVM-Value-Numbering/test
--------START--------
--------CASE1--------
ValueNumbering:
3=1+2
4=1*3
6=1+5
7=6+2
This Expression is Redundancy:   %11 = add nsw i32 %0, %1
3=1+2
This Expression is Redundancy:   %12 = mul nsw i32 %0, %11
4=1*3
ValueNumbering:
--------CASE2--------
ValueNumbering:
3=1+2
This Expression is Redundancy:   %8 = add nsw i32 2, 1
3=2+1
4=1*2
This Expression is Redundancy:   %10 = mul nsw i32 2, 1
4=2*1
5=3*3
6=4*4
ValueNumbering:
--------END--------
```

2. And in the test folder, there are two output.



```
3=1+2
4=1*3
6=1+5
7=6+2
3=1+2
4=1*3

```

```
3=1+2
3=2+1
4=1*2
4=2*1
5=3*3
6=4*4

```