===================================
Anthony Dubis (ajd2194)
Aikaterini Iliakopoulou (ai2315)
COMS W4112 Project #2
DUE April 22 2015
===================================

===================================
[001] LIST OF FILES
===================================
```
BPOptimizer.java : The main class of the program
Record.java      : The objects for the records that populate A[]
config.txt       : The provided config file for architecture costs
query.txt        : A list of 10 different sets of queries used for testing.
output.txt       : The output results for the sets of queries in query.txt
stage2.sh        : The shell script to run the program as specified
Makefile         : The makefile to compile the program
```

===================================
[002] HOW TO RUN
===================================
To compile go to the directory where the folders are located and run "make".
To run the algorithm, run "./stage2.sh query.txt config.txt”

===================================
[003] HIGH LEVEL DESCRIPTION
===================================
The main class of the program is BPOptimizer, which reads the input files, runs the algorithm to find the optimal plan for each set of selectivities, and outputs the plan to standard output. The Record class is used for storing all necessary information for a particular subset plan, including the terms the plan utilizes, their selectivities, plan's cost, the left and right branch, and the value of bit b that shows whether the plan is a no-branch plan or not.

In BPOptimizer, the input files, query.txt and config.txt are read in that order inside main to get the list of selectivities and the processor's parameters the algorithm is going to use to evaluate the costs of the different plans respectively. As soon as the parameters are set inside the program, the algorithm runs for each row of selectivities to produce the optimal plan for that particular set of functions. This is done by calling the method findOptimalPlan(...) of BPOptimizer. In summary, this method first creates all the subsets of records in an “increasing” order. It then runs the first part of the dynamic algorithm that calculates costs for the logical-& and for the no-branching plans. Finally, it then runs the second part of the algorithm, which uses dynamic programming to assemble optimal paths by going through the subsets in the “increasing” order. The last subset, the one that contains all of the terms, can then be used to recursively build the optimized plan by &&-ing the left &-term with the right term. The right term must be built by recursively working on its left &-term and its right term until it has no children.

The subsets are created with the help of bitmaps. Specifically, if we have for example a list of 4 functions then we define a binary number that can be thought of as [0 0 0 0], where each bit represents the absence or presence of the function in the respective subset plan. We use 1 for presence and 0 for absence. We then manipulate those bitmaps to create all unique subset combinations in an increasing order. This is done inside createSubsetsOfTerms(...).

The first part of the algorithm runs inside considerLogicalAndNoBranchingPlans(...) method, where all the logical-& costs of the subset plans are computed using example 4.5. Then, if the cost of the respective no-branch plan given by example 4.4 is smaller than the previous cost, the logical-& cost is replaced by the no-branch cost and the bit for using a no-branch subset plan is set to true.

The second part of the algorithm is executed inside considerBranchingAndPlans(...) method for all valid combinations of two subset (one as the left child, one as the right) plans with the goal of finding an optimized plan. Specifically, if two subset plans don't overlap, the conditions for c- and d-metrics support optimality, and the combined cost of the plan from uniting these left and right subsets (formula (1) in paper [1]) is lower than the current cost of the existing plan for their union, the updated plan and lower cost are updated in the array A[]. By the end of the execution, record in the last index of A[] contains the optimal plan based on the recursive relationship defined above and in the algorithm

This final record (again, the one that can be used to recursively compute the optimal plan) is then passed on method outputPlan(...). This method is responsible for printing the output plan to the standard output (console). Ultimately, the optimal plan is found by joining this final record’s left terms (combined as an &-term) by a branching && with the recursively solved plan for its right terms. We continue to recurse down the right links of what can be represented as a binary tree until we reach the last node (the last &-term). If this node, or Record, has an optimal plan that uses the no-brach technique, it is removed form the if-statement and a no-branch component is added to the solution.

Finally, the output of our algorithm for the sample set of selectivities contained in the query.txt file can be found in output.txt file.

Please not that we copied the output from the console to the output.txt and that the program doesn't write to the latter file, but instead prints the output to the standard output, as requested by the instructors.

==================================

[004] SOURCES

==================================

[1] K. A. Ross. Selection conditions in main memory. TODS, 29:132–161, 2004.

[2] To learn how to read variables from .properties files (see getCostValues() in BPOptimizer.java), the following link was referenced:

http://crunchify.com/java-properties-file-how-to-read-config-properties-values-in-java/

[3] To learn how to create Java Makefiles (see Makefile), the following link was reference:
http://www.cs.swarthmore.edu/~newhall/unixhelp/javamakefiles.html

[4] In order to create the subsets of the terms (see createSubsetsOfTerms() in BPOptimizer.java), I referenced this post on StackOverflow:
http://stackoverflow.com/questions/7206442/printing-all-possible-subsets-of-a-list