Assignment 2 – Binary Decision Diagrams

Create a program, where a data structure called BDD (Binary Decision Diagram) with a focus for representation of Boolean functions can be created.

Implement these functions:

- BDD *BDD_create(string *bfunction,* string *order*);
- char BDD_use(BDD **bdd*, string *input_values*);

Of course, you can implement any other functions, which can be used inside your program, helping with implementation of the abovementioned functions. You cannot, however, use existing code/functions for BDD implementation.

Function **BDD_create** serves to create a reduced binary decision diagram that is supposed to represent/describe any given Boolean function. The Boolean function is provided as function argument called *bfunction.* The Boolean function is described in a form of an expression in a string. The second argument of **BDD_create** is the *order* of variables used for the Boolean function. This order specifies the order of usage of these variables for creation of BDD (i.e. the BDD is created in the order of variables according to the *order* argument). Function **BDD_create** returns a pointer to the created BDD structure. This structure has to contain these parts at least: number of variables, size of BDD (i.e. number of BDD nodes) and a pointer to the root of BDD tree. Of course, you will need your own structure for representation of one node too. This **BDD_create** function already includes the reduction of BDD – you can choose whether you reduce the BDD during its creation or you reduce BDD after it is fully created. If you reduce BDD after it is fully created, then your solution is evaluated with fewer points (4 points penalization).

Function **BDD_use** serves for using created BDD for a specific given combination of values for input variables of Boolean function and for obtaining the result of Boolean function for the given input. Within this function, you „walk through" BDD tree from its root down to a leaf. The path from root to leaf is determined by the given combination of values of input variables. The arguments of this function are a pointer to a specific BDD that is used and a string called *input_values*. This string / array of chars is representing the given combination of values for input variables of Boolean function. For example, index of the array/string represents one variable and value at this index represents the value of this variable (i.e. for variables A, B, C and D, where A and C are 1s and B and D are 0s, the string can be "1010"), but this is just an example – you can choose to represent the *input_values* in a different way. The return value of function **BDD_use** is a char, which represents the result of Boolean function – it is either '1' or '0'. In case of an error (e.g. incorrect input), this result should be negative (e.g. -1).

Apart from implementation of the BDD functionality, it is required to test your solution appropriately. Your solution must be 100% correct. Within the testing it is needed to use some randomly generated Boolean functions, which will be used for creation of BDDs using the BDD_create. The correctness of BDD can be proved with iterative calling of function BDD_use in such a way that you will use all possible combinations of values for the input variables of Boolean function and compare the output/result of BDD_use with expected result. The expected results can be obtained from application of the values to variables within the Boolean function expression (evaluating the expression). Test and evaluate your solution for various numbers of variables of Boolean function – the more variables your program can handle, the better (it should be able to handle at least 13 variables). The number of different Boolean functions within the same amount of variables should be at least 100. Within your testing, you should evaluate the relative rate of BDD reduction (i.e the number of deleted nodes / number of all nodes for a full diagram).

Example of very simple test (just for understanding the problem):
```
#include <string.h>
int main(){
  BDD* bdd;
  bdd = BDD_create("AB+C");
  if (BDD_use("000") != '0') /* expected output value can be
obtained by assigning the concrete values to variables of the
expression - it is better to make a function to this */
    printf("error, for A=0, B=0, C=0 result should be 0.\n");
  if (BDD_use("001") != '1')
    printf("error, for A=0, B=0, C=1 result should be 1.\n");
  if (BDD_use("010") != '0')
    printf("error, for A=0, B=1, C=0 result should be 0.\n");
  if (BDD_use("011") != '1')
    printf("error, for A=0, B=1, C=1 result should be 1.\n");
  if (BDD_use("100") != '0')
    printf("error, for A=1, B=0, C=0 result should be 0.\n");
  if (BDD_use("101") != '1')
    printf("error, for A=1, B=0, C=1 result should be 1.\n");
  if (BDD_use("110") != '1')
    printf("error, for A=1, B=1, C=0 result should be 1.\n");
  if (BDD_use("111") != '1')
    printf("error, for A=1, B=1, C=1 result should be 1.\n");
  return 0;
}
```

Create a documentation, where you solution, individual functions, your structures, way of testing and test results are described. The test results should include average percentage rate of BDD reduction and average execution time of your BDD functions, of course, based on the size of Boolean function (i.e. number of variables). Documentation must contain a header (who, which assignment), brief description of the used algorithms with some pictures showing how the solution works or selected parts of your code. Try to explain why you think that your solution is correct – reasons why it is good/correct, and the way how you tested/evaluated it. At the end, your documentation should contain your estimation of time and memory complexity of your solution. Overall, it must be clear what you did and how can you prove that it is correct and how efficient it is.

The solution of the assignment must be uploaded in AIS by the set deadline (delays are allowed only in very serious cases, such as sickness, the decision whether the solution is accepted after deadline is decided by the instructor). One zip archive should be uploaded, which contains individual source code files with implementations, a test file and one documentation file in **pdf** format.

Evaluation

You can obtain a total of 20 points at most. The **minimum requirement is 8 points**.

For implementation of solution, you can obtain 10 points in total – 8 points for **BDD_create** and 2 points for **BDD_use**. For testing, you can obtain 6 points in total (2 points are for automatic obtaining of expected output/result of BDD by assigning values to variables in expression and automatic evaluation of the expression). For documentation, 4 points can be obtained. The number of obtained points depends on presentation to instruction as well (e.g. if you cannot answer questions of your instructor, you can get penalizations or even 0 points if you are not convincing enough that your solution is really your code). If your solution does not perform any reduction of BDD (i.e. full BDD is created only) or if your solution is not working 100% correctly, then it is not accepted (0 points).