```
//
//Massachusetts Institute of Technology
//16.412J/6.834J Cognitive Robotics
//
//Russian Doll Search
//
//Problem Set #2
//Due: in class Wed, 3/9/05
//
//Lawrence Bush, Brian Bairstow
//{ bushl2, bairstow }@mit.edu
// _________________________________________
//
// _________________________________________
//  Russian Doll Search - Main program.

#include<iostream>
#include<iomanip>
#include<string>
#include<fstream>
#include<vector>
#include<string>
#include<sstream>
#include "variable.h"
#include "variables.h"
#include "tuple.h"
#include "tuples.h"
#include "constraints.h"

//uncomment the next line to see the full walk through output
//#define DEBUG_OUTPUT_PROBLEM_FILE
//timer code
//#include "long_timer.h"
#include "timer.h"


using namespace std;


variables bnb(constraints C, std::ostream & out, int initial, vector<double> & future_cost↙
    , variables sa, vector<unsigned long long int> & operations, time_t & time_limit,int &↙
     variable_counter) {


    if(time_limit < time(0)) { return variables(); }

    variables ca = C.initialize_assignment(initial);

    //  double old_sa_eval = C.evaluate( sa,operations );// the following line is quicker,↙
     but this one would work also
    double old_sa_eval = future_cost[initial+1];
    C.initialize_upper_bound(sa, ca, operations);
    double new_sa_eval = C.evaluate(sa,operations );
    if(new_sa_eval==old_sa_eval) {
        future_cost[initial] = new_sa_eval;
        return sa;
    }

    double ub=C.get_upper_bound();
    double lb=0;
    double eval;

    while(1) {
        if(time_limit < time(0)) { return variables(); }

#ifdef DEBUG_OUTPUT_PROBLEM_FILE
        ca.print(out);
```

```
#endif

        if(ca.empty()){
            future_cost[initial] = ub;
            return sa;
        }
        eval = C.evaluate(ca,operations, future_cost[initial+ca.size()], ub  );
        //      eval = C.evaluate(ca,operations);// the above line is faster, but this one would work also
        lb = eval + future_cost[initial+ca.size()];
#ifdef DEBUG_OUTPUT_PROBLEM_FILE
        out << "Evaluates to:    " << eval << endl;
        out << "Current lb:      " << lb << endl;
        out << "Current ub:      " << ub << endl;
        out << "-----------------------------------------------------"<<endl;
#endif

        // terminal case
        // change the ub if applicable
        // even if it is ok, go to the right/up
        if(C.is_last_variable(ca)){// last variable

            // if last variable, and eval < ub, then set ub = eval (ub is current best assignment cost)
            if(lb<ub){
                ub=lb;
                sa = ca;
            }

            if(C.is_last_value(ca)){// last value
                ca = C.back_up(ca);
                operations[2]+=1;

            } else {// not last value
                // if last variable and if not last value, try next value
                ca = C.get_next_value(ca);
                operations[2]+=1;
            }

            ////////////////////////////////////////////
        } else {// not last variable yet

            // if eval >= ub && not last value, then try next value or go up(if last value)
            // if eval < ub, then try next variable

            if(lb<ub){//good, then try next variable
                ca = C.get_next_variable(ca);
                operations[2]+=1;
            } else {// if eval >= ub && not last value, then try next value or go up(if last value)
                if(C.is_last_value(ca)){// last value
                    ca = C.back_up(ca);
                    operations[2]+=1;
                } else {// not last value // not last variable
                    //  if not last value, try next value
                    ca = C.get_next_value(ca);
                    operations[2]+=1;
                }


            }




        }
```

```
    }

}


//////////////////////////////////////////////////////////////////////////////////////

double rds(constraints C, ostream & out, vector<unsigned long long int> & operations,     ↙
    time_t & time_limit,int & variable_counter)
{
    int n = C.get_number_of_variables();
    vector<double> future_costs(n+1, -1);
    variables sa;  //subproblem assignment

    future_costs[n] = 0;

    for(int i = n-1; i >= 0; i--)
    {
        sa = bnb(C, out, i, future_costs, sa, operations, time_limit, variable_counter);
        if(time_limit < time(0)) { return -1; }

#ifdef DEBUG_OUTPUT_PROBLEM_FILE
        out << "Subproblem Optimum:   " << future_costs[i] << endl;
        sa.print(out);
        out << endl ;
#endif

        if(sa.empty()) {
            return -2;
        }

        if(future_costs[i] >= C.get_global_upper_bound())
        {
#ifdef DEBUG_OUTPUT_PROBLEM_FILE
            out << endl << "Failure";
            //if for any subproblem that cost is worse than the global upper bound
            // then it is going to fail on any macro problem, so return flag
#endif
            return -2;//flag
        }
    }
    //#ifdef DEBUG_OUTPUT_PROBLEM_FILE
    out << endl << endl << "Value:    " << future_costs[0] << endl;
    sa.print(out);
    //#endif

    return future_costs[0];

}


/////////////////////////////////////////////////////////////////////////////////////


// This function converts doubles to strings.

string ltos(unsigned long long int d) {
    ostringstream ost;
    ost<<d;
    return ost.str();
}
string dtos(double d) {
    ostringstream ost;
    ost<<d;
    return ost.str();
}
int main(int argc, char *argv[] ) { //  start of main, with input of file argument and the↙
```

```
     number of arguments

    vector<unsigned long long int> operations_old;
    vector<unsigned long long int> operations;

    int time_duration = 599;

    if (argc == 2 ) {  //  if there are 2 arguments
        //open out file
        ofstream collective_out;
        collective_out.open(argv[1],ios::out);//write / clear

        if (collective_out == NULL) {// error of out file could not be opened
            cerr << "Error, could not clear the collective outfile" << endl;
            exit(0);
        }

        // 0 is tuple    count
        // 1 is variable count
        // 3 is nodes visited
        collective_out <<"Problem_Name  "<<"  Number of Variables"<<"  Number of Nodes"<< ↙
    "Run_Time(seconds)"<<"  Evaluated Tuple"<<"  Evaluated Variables"<<"  Nodes Visited"< ↙
    <"  Optimal_Value"<<endl;
        collective_out.close();//close output file
        cout<< "Cleared the file : " << argv[1] << endl << endl ;
        cout <<"Problem_Name  "<<"Run_Time(seconds)"<<"  Operation_Counts"<<endl;
        exit(0);

    }

    if (argc != 4 ) {  //  if there are not 4 arguments
        // Error message and command line argument instructions
        cout<< "Command line arguments:\n" << "example:\n" << "problem_in_file_name       ↙
    problem_out_file_name collective_out.txt\n";
        exit(0);
    } else {
        cout<< "There are 4 Command line arguments: "<< argv[0] << " and " << argv[1] << "↙
     and " << argv[2] << " and " << argv[3] << ".\n" << endl;

    }

    //open out file
    ofstream out;
    out.open(argv[2],ios::out);
    if (out == NULL) {// error of out file could not be opened
        cerr << "Error, could not open the file file" << endl;
        exit(0);
    }

    //open out file
    ofstream collective_out;
    collective_out.open(argv[3],ios::app);//append
    if (collective_out == NULL) {// error of out file could not be opened
        cerr << "Error, could not open the collective outfile" << endl;
        exit(0);
    }


    string input_file_name = argv[1];
    cout << "Input Filename = "<<input_file_name<<"\n";

    ifstream in;
    in.open(argv[1],ios::in);
    if (in == NULL) {// error message if in file could not be opened
        cerr << "Error, could not open the output file" << endl;
        exit(0);
    }
    // Instantiate variables for file input
    string problem_name;
    int number_of_variables;
    int maximum_domain_size;
    int number_of_constraint_groups;
    int global_upper_bound;

    // Read in the problem header:
    in>>problem_name>>number_of_variables>>maximum_domain_size>>                  ↙
    number_of_constraint_groups>>global_upper_bound;
    // Output the problem header:
    cout<<problem_name<<" "<<number_of_variables<<" "<<maximum_domain_size<<" "<  ↙
    <number_of_constraint_groups<<" "<<global_upper_bound<<endl;

    variables X; // X is a set of variables

    // Read in the domain sizes and initialize the variables.
    for(int count_var_index = 0; count_var_index < number_of_variables ; count_var_index+ ↙
    +){
        int next_variable_domain_size;
        in>>next_variable_domain_size;
        //      cout<<next_variable_domain_size<<endl;

        int domain_value = -1;

        //  variable next_variable(count_var_index, next_variable_domain_size,       ↙
    domain_value);
        X.insert(   variable(count_var_index, next_variable_domain_size, domain_value) );
        //      cout<<next_variable<<endl;

    }

    X.print(out);

    constraints C(number_of_variables, maximum_domain_size,number_of_constraint_groups,  ↙
    global_upper_bound, X);

    //  vector<constraint> constraint_vector;

    int constraint_arity;
    int next_variables_in_constraint;
    int default_cost;
    int number_of_tuples;
    int next_tup_value;
    int non_default_value;

    // Read in each constraint group:
    for(int count = 0; count < number_of_constraint_groups ; count++){

        ///////////////////////////////////////////////////////////////////////////////
        // Read constraint group header line (num vars, each var, default cost, num    ↙
    tuples).

        // Read number of variables in the constraint (constraint arity).
        in>>constraint_arity;
        // Read the indecies of variables in the constraint.
        // Note: we have to remember this order, so we can apply the right value      ↙
    assignment to the right variable.
        // This should be an iterable set of variables.
        vector<int> cX_indecies; // cX is the subset of variable indecies in the      ↙
    constraint.
        for(int count_constraint_arity = 0; count_constraint_arity < constraint_arity ;  ↙
    count_constraint_arity++){
            in>>next_variables_in_constraint;
            cX_indecies.push_back(next_variables_in_constraint);
        }
        // Read the Default cost value.
        in>>default_cost;
        // Read the Number of tuples (variable set assignments) without the default cost.
        in>>number_of_tuples;

        // create the constraint
        //constraint temp_constraint(constraint_arity, default_cost, number_of_tuples,    ↙
    cX_indecies);

        // create the tuple set
        tuples temp_tuples(constraint_arity, default_cost, number_of_tuples, cX_indecies);
        //  cout<<number_of_tuples_without_default_cost_value<<endl;
        //
        ///////////////////////////////////////////////////////////////////////////////////↙
    //

        ///////////////////////////////////////////////////////////////////////////////////↙
    //
        // Read in each constraint tuple (variable set assignments without the default    ↙
    cost).
        for(int count_number_of_tuples = 0; count_number_of_tuples < number_of_tuples ;   ↙
    count_number_of_tuples++){
            variables c_vars; // a set of variables in the sub-constraint

            // Read each the tuple:
            // Here we should iterate through the variable set.
            // Then create a variable / value "pairing" which goes into a value object.
            // The value objects are the combined into a constraint object, which also    ↙
    contains the default value.

            for(vector<int>::iterator count_icx = cX_indecies.begin(); count_icx !=       ↙
    cX_indecies.end() ; count_icx++ ){
                //          for(int count_number_of_variables_in_constraint= 0;           ↙
    count_number_of_variables_in_constraint < number_of_variables_in_constraint ;         ↙
    count_number_of_variables_in_constraint++){

                // read the n tuple assignments
                in>>next_tup_value;

                c_vars.insert(  variable(*count_icx, X[*count_icx].get_domain_size(),     ↙
    next_tup_value) );
            }

            // read the non-default cost.
            in>>non_default_value;
            // Create a new value.
            tuple temp_tuple = tuple(c_vars,non_default_value);
            temp_tuples.insert(temp_tuple);
        }
        C.insert_tuples(temp_tuples);
    }// end read each constraint group

    C.sort_tuples_by_num_non_default_cost();



    in.close();//close input file

    /////////////////////////////////////////////////////
    /////////////////////////////////////////////////////
    /////                                          /////
    /////               Time Trials                /////
    /////                                          /////
    /////////////////////////////////////////////////////
    /////////////////////////////////////////////////////
    /////         To seconds trials first          /////
    /////////////////////////////////////////////////////
    srand(time(0));//timer
    time_t initial, final;
    initial = time(0);
    time_t time_limit = initial + time_duration;
    int memory_used = 0;
    int variable_counter = 0;
    //operations=0;
    operations.clear();
    operations.push_back(0);//
    operations.push_back(0);//
    operations.push_back(0);//

    ///////////////////////
    ///     run rds     ///
    ///////////////////////
    double optimal_value = rds(C, out,operations, time_limit, memory_used);
    cout<<"hi"<<endl;
    final = time(0);
    double elapsed_seconds = difftime(final,initial);

    if(elapsed_seconds> 1){

        ////////////////////
        //  output lines  //
        ////////////////////
        // parse the problem name
        string problem_name = argv[1];
        string::size_type ssl = problem_name.find("/");
        problem_name.replace(0,ssl+1,"");
        ssl = problem_name.find("/");
        problem_name.replace(0,ssl+1,"");
        ssl = problem_name.find(".");
        problem_name.replace(ssl,ssl+4,"");
        ////////////////////////////////////

        string optimal_value_string=dtos(optimal_value);
        if(optimal_value == -1){ optimal_value_string = "time expired"; }
        if(optimal_value == -2){ optimal_value_string = "failure";}



        collective_out <<problem_name<<"  "<<C.get_number_of_variables()<<"  "<<    C.    ↙
    get_number_of_nodes()   <<string(31-problem_name.length()-dtos(elapsed_seconds).length↙
    (),' ')<<elapsed_seconds <<string(18-ltos(operations[0]).length(),' ')<< ltos          ↙
    (operations[0]) <<string(18-ltos(operations[1]).length(),' ')<< ltos(operations[1]) < ↙
    <string(18-ltos(operations[2]).length(),' ')<< ltos(operations[2]) <<string(15-       ↙
    optimal_value_string.length(),' ')<<"     "<<  optimal_value_string  <<endl<<flush;
        cout <<problem_name<<"  "<<C.get_number_of_variables()<<"  "<<  C.                   ↙
    get_number_of_nodes()   <<string(31-problem_name.length()-dtos(elapsed_seconds).length↙
    (),' ')<<elapsed_seconds <<string(18-ltos(operations[0]).length(),' ')<< ltos          ↙
    (operations[0]) <<string(18-ltos(operations[1]).length(),' ')<< ltos(operations[1]) < ↙
    <string(18-ltos(operations[2]).length(),' ')<< ltos(operations[2]) <<string(15-       ↙
    optimal_value_string.length(),' ')<<"     "<<  optimal_value_string  <<endl<<flush;
        ////////////////////


    } else {
        ////////////////////////////////////////////////////
        ////////////////////////////////////////////////////
        ////////////////////////////////////////////////////
        // if it is a short time frame, then use this more precise method:
        // timer declarations
        unsigned int reps;
        unsigned long N;
        N=10;
        reps = 10;
        srand(time(0));//timer
        timer tim;
        // Compute the baseline time for N
        tim.start_baseline(reps);
        do {
            //////////////////////////////
            // Baseline Operations Here
            // (anything that isn't really the algorithm
            //////////////////////////////
            // nothing //
            // End of Baseline Operations
            ///////////////////////////////
        } while (tim.check());

        tim.report(false);

        time_t time_limit = time(0) + time_duration;
        tim.start(reps, N);
        do {

            //////////////////////////////
            // Baseline Operations Here
            // (anything that isn't really the algorithm
            //////////////////////////////

            // End of Baseline Operations
            ///////////////////////////////
            // Main Timed Operation
            // this is the actual timing
            //      for(int count_rds=1;count_rds<20000;count_rds++) {
            //operations=0;
            // 0 is tuple    count
            // 1 is variable count
            // 3 is nodes visited
            operations.clear();
            operations.push_back(0);//
            operations.push_back(0);//
            operations.push_back(0);//
            double optimal_value = rds(C, out,operations, time_limit, variable_counter);
            //////////////////////////////////////
        }
        while (tim.check());

        tim.report(false);

        cout<<"longer that 3 seconds"<<endl;
        elapsed_seconds = tim.report();
        tim.report(false);

        ////////////////////
        //  output lines  //
        ////////////////////
        string problem_name = argv[1];
        string::size_type ssl = problem_name.find("/");
        problem_name.replace(0,ssl+1,"");
        ssl = problem_name.find("/");
        problem_name.replace(0,ssl+1,"");
        ssl = problem_name.find(".");
        problem_name.replace(ssl,ssl+4,"");

        string elapsed_seconds_string=dtos(elapsed_seconds);
        string optimal_value_string=dtos(optimal_value);
        if(optimal_value== -1){ optimal_value_string = "time expired";}
        if(optimal_value== -2){ optimal_value_string = "time expired";}

        collective_out <<problem_name<<"  "<<C.get_number_of_variables()<<"  "<<    C.    ↙
    get_number_of_nodes()    <<string(31-problem_name.length()-dtos(elapsed_seconds).length↙
    ()," ')<<elapsed_seconds <<string(18-ltos(operations[0]).length(),' ')<< ltos          ↙
    (operations[0]) <<string(18-ltos(operations[1]).length(),' ')<< ltos(operations[1]) < ↙
    <string(18-ltos(operations[2]).length(),' ')<< ltos(operations[2]) <<string(15-      ↙
    optimal_value_string.length(),' ')<<"     "<<  optimal_value_string  <<endl<<flush;
        cout <<problem_name<<"  "<<C.get_number_of_variables()<<"  "<<  C.               ↙
    get_number_of_nodes()   <<string(31-problem_name.length()-dtos(elapsed_seconds).length↙
    (),' ')<<elapsed_seconds <<string(18-ltos(operations[0]).length(),' ')<< ltos         ↙
    (operations[0]) <<string(18-ltos(operations[1]).length(),' ')<< ltos(operations[1]) < ↙
    <string(18-ltos(operations[2]).length(),' ')<< ltos(operations[2]) <<string(15-      ↙
    optimal_value_string.length(),' ')<<"     "<<  optimal_value_string  <<endl<<flush;

        //sa.print(collective_out);//you can print out the final assignment again here if ↙
    you wish.
        ////////////////////


    }

    out.close();//close input file

    return 0;

}
```