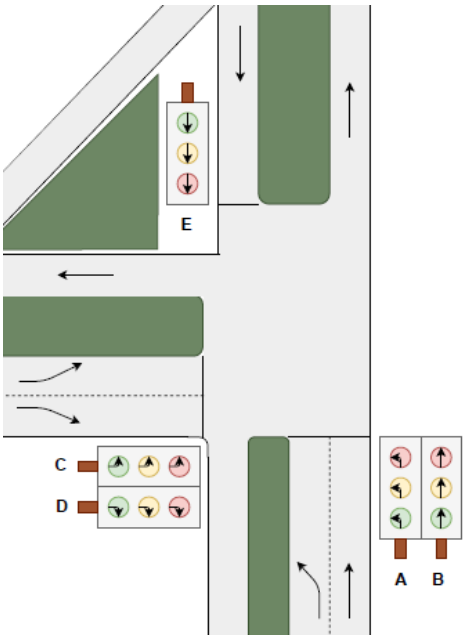
Artificial Intelligence (521495A), Spring 2022
Exercise 3: Answers to Problems 1-3

Problem 1.



(a) Variables: A, B, C, D, E

Domains: { RED, GREEN } for each variable

Constraints: not ((A = GREEN) and (C = GREEN))

not ((A = GREEN) and (E = GREEN))

not ((B = GREEN) and (C = GREEN))

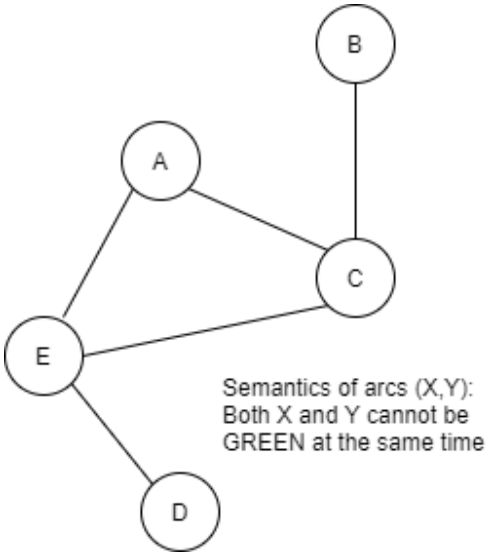
not ((C = GREEN) and (E = GREEN))

not ((D = GREEN) and (E = GREEN))

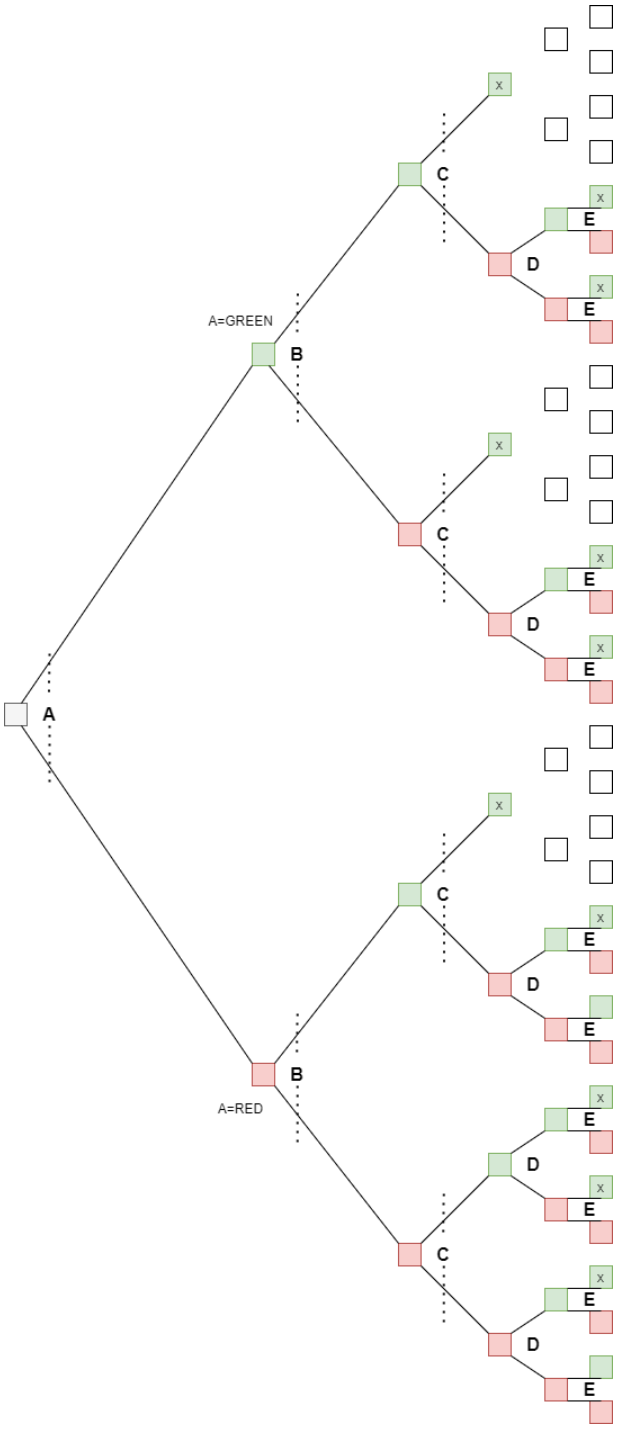
Specific feature of this problem is that we are interested in <u>all</u> possible light combinations, which fulfill the constraints. It is not sufficient to find a single solution using backtracking.

(b) Constraint graph:

Binary constraint graph is sufficient.



(c) Search tree for the assignment order (A, B, C, D, E) below. Nodes denoted with 'x' violate some constraint. It can be seen that there are 12 possible combinations.



(d) Better assignment order for the variables:

Two principles were mentioned in the class:

1. Minimum Remaining Values (MRV: choose the variable with the fewest legal values).

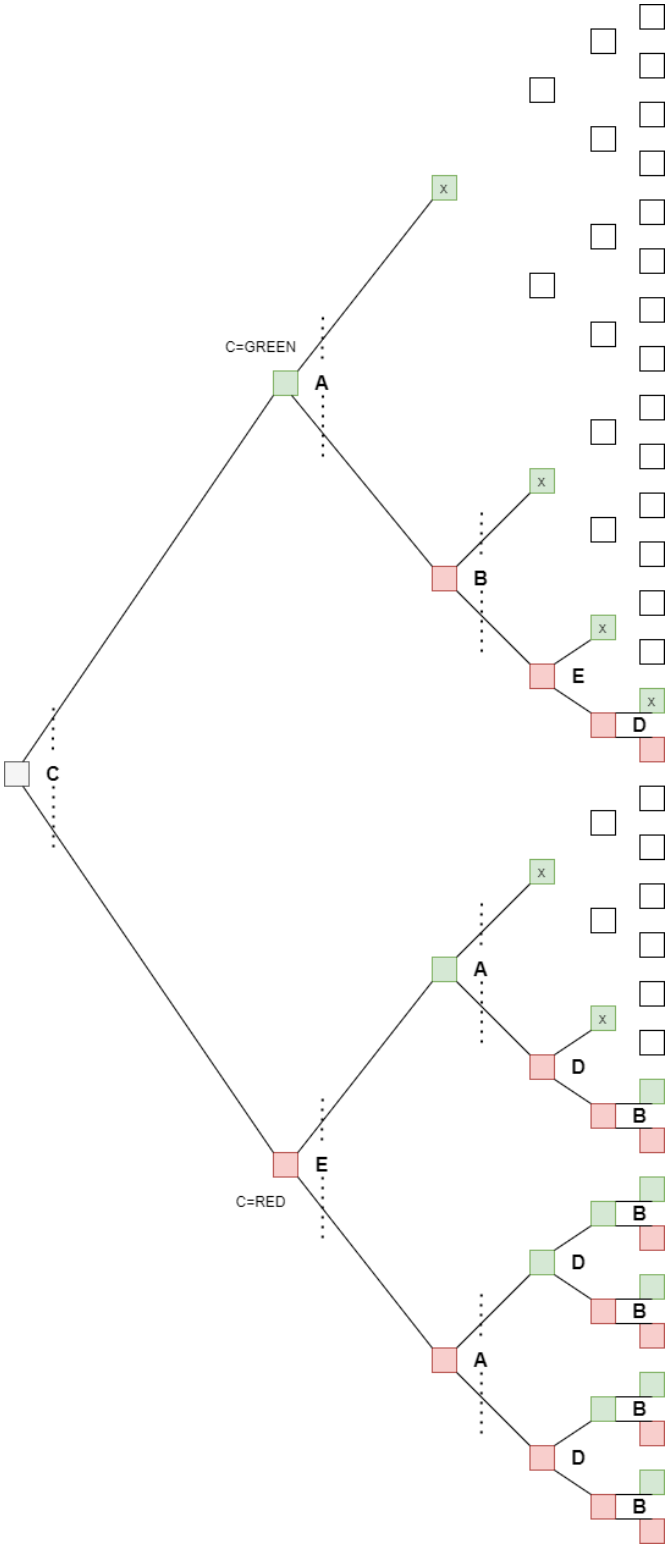
2. Degree Heuristic (DH: choose the variable involved in the largest number of constraints).

Considering the choice of the first variable, MRV is not useful as all variables have 2 legal values. Based on DH, the candidates are the variables C and E (both are involved in three constraints). Let us choose C.

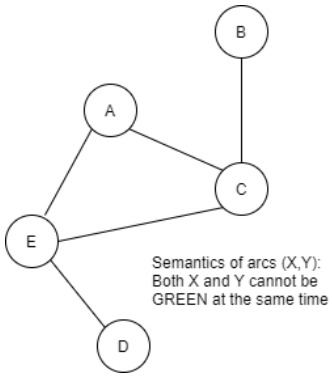
For the branch C = RED, other variables still have 2 legal values. Let us use E as the second variable based on DH: it is connected to both A and D. There are no constraints for other variables if E = RED. If E = GREEN, A and D both must be RED. Finally, B can be either RED or GREEN.

For the branch C = GREEN, A, B, and E must be RED so they should be assigned next. Finally, D has two possible values, RED and GREEN.

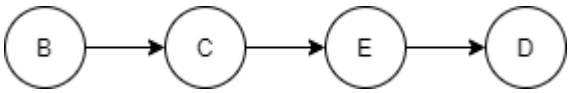
To summarize, we get the search tree shown on the right. If we compare this to the search tree derived in problem (c), we see that there are more nodes, which do not have to be considered at all (shown as white squares). *It means that the search is more effective, it backtracks earlier so there is benefit in using MRV and DH heuristics.*



(e) The constraint graph is nearly tree structured.



To obtain a tree-structured CSPs, let us use { A } as the cutset.[1] To solve the resulting CSPs, one alternative is to select B as the root note. As a result, we get the following topological ordering:



Cutset conditioning process:

1) Condition: A = RED.

Condition does not constrain other variables.

Constraint propagation: Arc consistency does not prune the domains of E, C, and B.

Backtracking search (all solutions)

B = RED => C = RED => E = RED => D = RED

B = RED => C = RED => E = RED => D = GREEN

B = RED => C = RED => E = GREEN => D = RED

B = RED => C = GREEN => E = RED => D = RED

B = RED => C = GREEN => E = RED => D = GREEN

B = GREEN => C = RED => E = RED => D = RED

B = GREEN => C = RED => E = RED => D = GREEN

B = GREEN => C = RED => E = GREEN => D = RED

2) Condition A = GREEN.

Due to condition, domains of C and E are pruned, they can only be RED.

Constraint propagation: Arc consistency does not prune the domains of E, C, and B.

Backtracking search (all solutions)

B = RED => C = RED => E = RED => D = RED

B = RED => C = RED => E = RED => D = GREEN

B = GREEN => C = RED => E = RED => D = RED

B = GREEN => C = RED => E = RED => D = GREEN

All twelve combinations found.

(f) For each light X, when X = GREEN, find the maximum number of other lights that can be green. By inspecting the search trees, we get: For A, {B, D}. For B, {A, D}. For C, { D }. For D, {A, B}. For E, { B }. Major GREEN light combinations for the crossing are { A, B, D }, { C, D } and { B, E }.

[1] Other alternatives for a minimal cutset are { C } and { E }.

Problem 2.

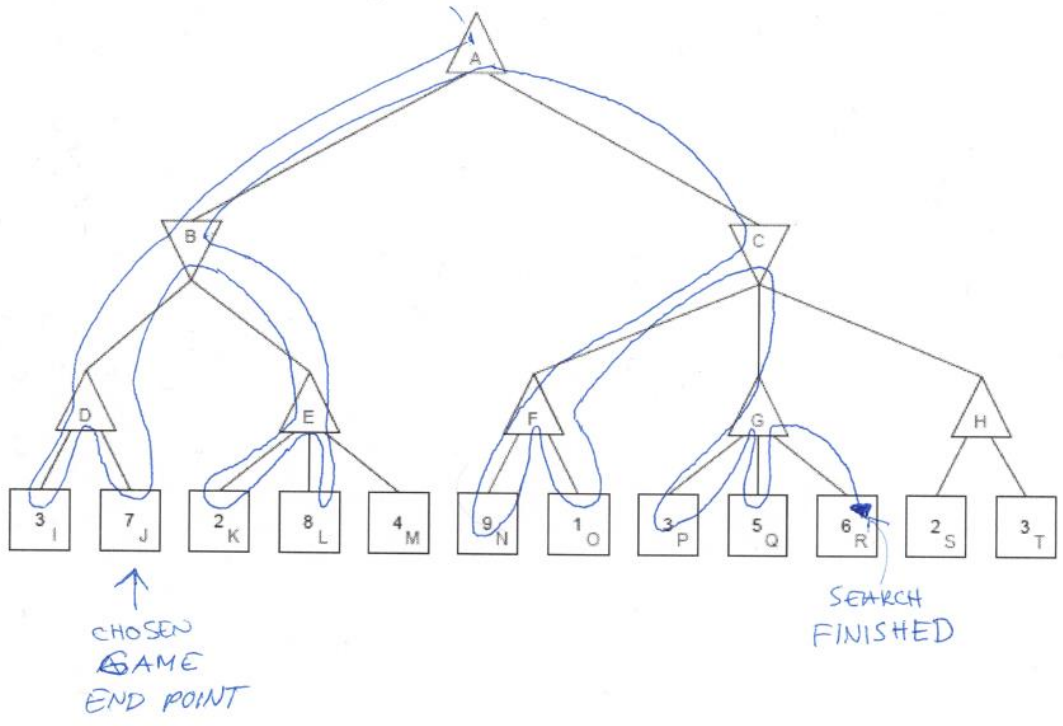
(a) In min max trees it is assumed that each level will choose the node that is the best for the outcome of the game. From node F the best result for the max level is 9 and for the nodes G and H the results are 6 and 3 respectively. Now for minimizing level the best result will be 3, which comes from node H. Thus, if the max level chooses node C in the beginning, the goodness of the outcome will be 3.

(b) The aim of alpha-beta-pruning is to discard irrelevant nodes from the search tree. In our case the maximizer starts to examine the tree as in depth-first (A, B, D, I and J). At this point it knows that if the minimizing level at node B chooses node D, then the maximizing level can score 7.

The search continues through nodes E, K, and L. At this point the maximizer knows that if the game would go to node E it would score at least 8 from the node L. However, it also realizes that the game will never go there as the minimizing level would never choose node E when it sees that the maximum score of the node D is less than what could come from node E. Therefore, the maximizer knows that the node M need not be checked, and the search continues with nodes C, F, N and O.

The maximizer can score 9 if minimizer chooses node F from node C. After checking the nodes G, P, Q and R, the maximizer realizes that the minimizer would choose node G rather than node F, because the node F would score 6. Now, since the score of the left branch (node A – node B) is 7, and the minimizer can force a score of 6 by choosing the node G in the right branch, the maximizer can determine that examining node H is not useful since it can only help the minimizer (high score would prevent minimizer from choosing H, and a low score would be in minimizers favor). Therefore, the search ends at node R. The search is illustrated in the image below.

(c) The game will end at node J.



(d) MIN nodes substituted with EXP nodes: Let us assume that at EXP nodes actions are chosen with equal probability. The value of nodes D and E is 7 and 8, so value of the node B is 7.5. The value of nodes F, G, and H is 9, 6, and 3, respectively, so the value of the node C is 6. So, the root node will choose the move leading to the node B.

Problem 3. The solution is to add to inner loops some restrictions to the values considered. The code below generates only 3938 nodes.

```
function crypta_solution

    C1s = [0,1];
    C2s = [0,1];
    Fs = [1];
    Rs = [0,2,4,6,8];
    Os = [1,2,3,4,5,6,7,8,9];
    Us = [0,1,2,3,4,5,6,7,8,9];
    Ws = [0,1,2,3,4,5,6,7,8,9];
    Ts = [0,1,2,3,4,5,6,7,8,9];

    count = 0;
    for F = Fs
        for C1 = C1s
            for C2 = C2s
                % Carry C1 limits the choice of Os and Rs
                if (C1 == 1)
                    Os = [5,6,7,8,9];
                    Rs = [0,2,4,6,8];
                else
                    Os = [1,2,3,4];
                    Rs = [2,4,6,8];
                end
                for R = Rs
                    for O = Os
                        if (C2 == 1)
                            Ws = [5,6,7,8,9];
                        else
                            Ws = [0,1,2,3,4];
                        end
                        if (C1 == 1)
                            Us = [1,3,5,7,9];
                        else
                            Us = [0,2,4,6,8];
                        end
                        for U = Us
                            for W = Ws
                                for T = Ts
                                    count = count+1;
                                    if goal_test(T,W,O,F,U,R,C1,C2)
                                        fprintf('T=%d,W=%d,O=%d,F=%d,O=%d,U=%d,R=%d\n',…
                                                T,W,O,F,O,U,R);
                                        fprintf('%d leaf nodes visited.\n',count);
                                        return;
                                    end
                                end
                            end
                        end
                    end
                end
            end
        end
    end
end
```