CS3383, Winter 2019 Assignment # 4 Sample solutions

Rasoul Shahsavarifar
Faculty of Computer Science, UNB

Questions 1) The algorithm first fixes any variables appearing as a 1-literal clause and then chooses a value for a variable in a 2-clause. We prove that the required time is $O(n^2)$, where n is the number of variables. Given a formula φ with n variables, we can first fix any variables appearing in a clause with just one variable. This work takes $O(n)$ time. After this we pick a variable x_i appearing in a 2-clause and fix to value ν_i. We have the following cases:

1. This may create other clauses with 1 variable, which may then be fixed. Suppose the number of clauses remaining is $m'(m' \leq m)$. If this did not create an empty clause, the algorithm will proceed with m'. An important observation is that the formula φ' formed by these m' clauses is a subformula of φ. Since we have already satisfied all clauses in φ that are not in φ', φ is satisfiable if and only if φ' is satisfiable. Hence, we can simply proceed with φ' and will never need to backtrack to change the value of x_i.

2. If we derive an empty clause by fixing $x_i = \nu_i$, we know that x_i should be set to ν_i. If setting $x_i = \overline{\nu}_i$ does not derive the empty clause, we again get a subformula φ' and the previous argument applies.

3. If both $x_i = \nu_i$ and $x_i = \varphi v_i$ lead to the empty clause, the formula is unsatisfiable and we stop.

Notice that when we choose a value for a variable, either we derive an empty clause immediately (after fixing variables in the 1-clauses thus created) in time $O(n)$, or we simply carry on with the subformula derived and never backtrack. In the worst case, we derive an empty clause for one value of each variable and carry on with the other value. Hence the maximum time is $n * O(n) = O(n^2)$.

Questions 2) A sample pseudocode (recursive version) for this problem is as follows:

$3DM(S, X) : returns$ **boolean**
$done \leftarrow$ **true**;
$S_1 \leftarrow \emptyset$; $S_2 \leftarrow \emptyset$; $X_2 \leftarrow \emptyset$;
if $X = \emptyset$
    $return$ **true**
$done \leftarrow$ **false**
if $|S| > 0$
    $for\ each\ (x, y, z) \in S$
        $S_1 \leftarrow S \setminus \{(x, y, z)\}$
        $S_2 \leftarrow S \setminus \{(a, b, c) \mid a \in \{x, y, z\}\ or\ b \in \{x, y, z\}\ or\ c \in \{x, y, z\}\}$
        $X_2 \leftarrow X \setminus \{x, y, z\}$
        $done \leftarrow 3DM(S_2, X_2)$
    if not $done$
        $done \leftarrow 3DM(S_1, X)$
return $done$

Note: You are encouraged to think about returning the set S' (if it exists) instead of the YES/NO answer for this question.

Question 3) A sample pseudocode (recursive backtracking version) for this question is as follows:

$Ind - Set(G(V, E), k, V') : returns$ **boolean**
$done \leftarrow$ **true**;
Construct two graphs G_1 and G_2 with empty sets of vertices and edges
if $|V'| = k$
    $return$ **true**
$done \leftarrow$ **false**
if $|V| > 0$
    for $each$ $u \in V$
        Update G_1 by removing u and all neighbours of u from G
        $done \leftarrow Ind - Set(G_1, k, V' \cup \{u\})$
    if not $done$
        Update G_2 by removing u from G
        $done \leftarrow Ind - Set(G_2, k, V')$
return $done$

Note: Again, you are encouraged to think about returning the of k vertices (if it exists), instead of the YES/NO answer for this question.

Question 4)

a) A formula that helps to check the condition is sum of the integer numbers $1, 2, \ldots n^2$ which is equal to $n^2(n^2 + 1)/2$.

b) For this part, let assume that $1 \leq p, q \leq n$. We also assume that *NoDuplicates* is a sub-routine that takes an array of size n^2 and returns a boolean value representing whether the array has some duplicates elements or not. This sub-routine is a modified version of

the solution of Question 2 in assignment 3. So, we ignore writing its pseudocode again.

Initialize array $SumOverRows[1, ..., n^2]$ with zeros;
Initialize array $SumOverColumns[1, ..., n^2]$ with zeros;
Initialize array $SumInBox[1, ..., n^2]$ with zeros;
Initialize array $ndr[1, ..., n^2]$ of booleans with True;
Initialize array $ndc[1, ..., n^2]$ of booleans with True;
Initialize array $ndbox[1, ..., n^2]$ of booleans with True;
$Answer \leftarrow True$;
$for\ \ i = 1\ \ to\ \ n^2$
    $for\ \ j = 1\ \ to\ \ n^2$
        $SumOverRows[i] \leftarrow SumOverRows[i] + Sudoku[i][j]$;
        $SumOverColumns[i] \leftarrow SumOverColumns[i] + Sudoku[j][i]$;
    $ndr[i] \leftarrow NoDuplicates(row\ i)$;
    $ndc[i] \leftarrow NoDuplicates(column\ i)$;
$for\ \ each\ \ box[p][q]$
    $for\ \ i = (p-1)n + 1\ \ to\ \ pn$
        $for\ \ j = (q-1)n + 1\ \ to\ \ qn$
            $SumInBox[(p-1)n + q] \leftarrow SumInBox[(p-1)n + q] + Sudoku[i][j]$;
$ndbox[(p-1)n + q] \leftarrow NoDuplicates(box_{pq})$; // box_{pq} is a 1D array converted
                                                        // from 2D array box[p][q].
$for\ \ i = 1\ \ to\ \ n^2$
    if $(SumOverRows[i] == SumOverColumns[i] == SumInBox[i]$
                $== \frac{n^2(n^2+1)}{2}\ \ \&\&\ \ ndr[i]\ \ \&\&\ \ ndc[i]\ \ \&\&\ \ ndbox[i]$ **);**
        $Answer \leftarrow Answer\ \&\&\ True$;
    else
        $Answer \leftarrow Answer\ \&\&\ False$;
return $Answer$;

---

c) the total running time of this algorithm is $O(m^2)$, where m is the input size (i.e. $m = n^2$). So the Algorithm would be considered as a quadratic algorithm in terms of the input size.

Note: Try to analyze the algorithm by going through every single line of the pseudocode.