CS170–Fall 2014 — Solutions to Homework 11

Quoc Thai Nguyen Truong, SID 24547327, `cs170-ig`

November 26, 2014

Collaborators: Kiet, Aditya

1. Beyond Suspicion

(a) Suspected False

(b) True

(c) True

(d) Suspected False

(e) Suspected True

(f) True

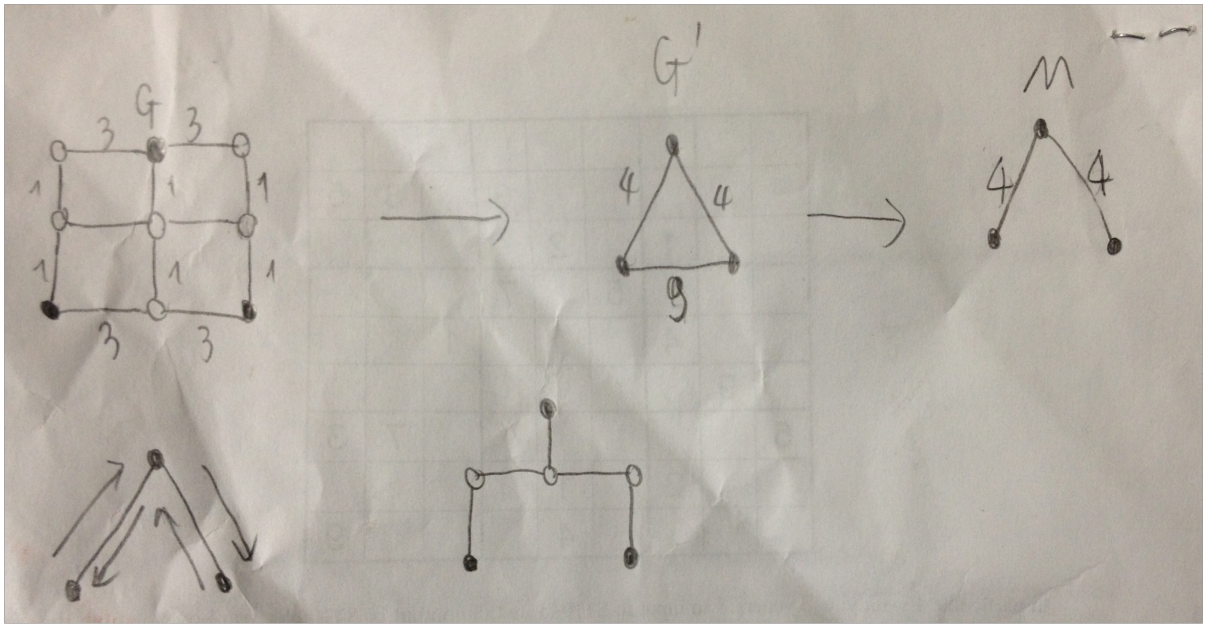
(g) True

(h) Suspected False

2. Approximation Algorithms

Proof.
Let w be the total weight of the minimum Steiner tree.
Let r be the weight in G′ that cost to visit every special nodes in S.
Let b be the total weight Steiner tree output from the approximation algorithm in the problem.
To prove that this algorithm achieves an approximation ratio of 2, we want to show that the ratio of b and w to be less than 2 which is $2w \geq b$

If we do DFS on the optimal minimum Steiner tree to create a tour ,and traverse all nodes and return to the source, it will give us a total weight of 2w.
If we apply the same on graph G′, then the total weight would be less than the optimal minimum Steiner tree.
The reason is not all the shortest distances between nodes u and v (u,v are in S) exist in graph G. We also remove any duplicate nodes in the tour in graph G′. Hence, the weight must be less than 2w.
In step 3, we add all edges that is also edges from minimum spanning tree. Hence, weight b can't be greater the weight r. Therefore, $b \leq r \leq 2w$



3. Algorithm Design Practice

Main idea.
The idea is to use Dynamic programming with memorization to solve this problem.
Let P(i) = largest index j, such that $j < i$, and interval j does not overlap with interval i, we use binary search in order to find j in the list of $([l_1, r_1], \cdots, [l_i, r_i])$
Let B(i) = the optimal solution which is the best weight attainable from non-overlap intervals $([l_1, r_1], \cdots, [l_i, r_i])$
There are 2 cases:
+ If the weight w_i of the current interval$[l_i, r_i]$ is in the optimal solution, then $B(i) = w_i + B(P[i])$.
+ Else : $B(j) = B(i-1)$.
In these 2 cases, we pick the one which give the higher total weight between intervals $([l_1, r_1], \cdots, [l_i, r_i])$ that don't overlap.
Therefore we have a recurrence relation:

$$B(i) = max(w_i + B(P[i]), B(i-1))$$

Pseudocode.
Line 0: M = [], M[0] = 0
Line 1: P = [], P[0] = 0
Line 2: Computer P using binary search in order to find largest index j value interval that does not overlap in the list for every index i in P
Line 3: BestWeigth($([l_1, r_1], \cdots, [l_k, r_k])$, $(w_1, \cdots, w_k)$):
Line 4: If $k == 0$: return 0
Line 5: for $i := 1$ to k:
Line 6: $M[i] = max(w_i + M[P[i]], M[i-1])$
Line 7: return M[k]

Proof of correctness.
Looking at the algorithm, we can see that it broke into sub-problems that are getting the maximum weights between non-overlap intervals. In the For loop, each time when it invokes M[i], it will calls $M[P[i]]$ $(P[i] < i)$ and $M[i-1]$, and we know for sure that $M[P[i]]$ stores the optimal solution which is the best weight attainable from non-overlap intervals $([l_1, r_1], \cdots, [l_{P[i]}, r_{P[i]}])$, and $M[i-1]$ stores the optimal solution which is the best weight attainable from non-overlap intervals $([l_1, r_1], \cdots, [l_{i-1}, r_{i-1}])$. Hence, we just update a new optimal solution which give the maximum value between $w_i + M[P[i]]$ and $M[i-1]$. We can see that the For loop iterate k times, so M[k] is the optimal solution which is the best weight attainable from non-overlap intervals $([l_1, r_1], \cdots, [l_k, r_k])$. Therefore, the algorithm return the best total weight of non-overlap intervals in k intervals.

Running time.

$$\boxed{T(k) = \Theta(klog(k))}$$

Justification of running time.
Let T(k) be the run time of the algorithm with k intervals (k weights).
At line 2: since the list is pre-sorted by the end value, using binary search in order to find largest index for array P take $\Theta(k \log(k)$. Call it k time and each time take $\Theta(\log(k))$
At line 5 and 6: each time when it invokes $M[i]$, it calls $M[P[i]]$ $(P[i] < i)$ and $M[i-1]$ which we already have, so it take constant time to lookup and find the maximum value. We know that it

iterate from 1 to k.
Therefore,

$$T(k) = \Theta(klog(k)) + \Theta(k)$$

$$\boxed{T(k) = \Theta(k \log(k))}$$

4. Learn to use a SAT solver

Final answer.
$\boxed{Yes}$. Alice could be a werewolf
List of variables and explanations.

- Let A be Alice is a knight
- Let B be Bob is a knight
- Let C be Carol is a knight
- Let !A be Alice is a Knave
- Let !B be Bob is a Knave
- Let !C be Carol is a Knave
- Let WA be A is a werewolf
- Let WB be B is a werewolf
- Let WC be C is a werewolf
- Let !WA be A is not a werewolf
- Let !WB be B is not a werewolf

Listing of STP file.

```
A => WA
!A => !WA
B => WB
!B => !WB
C => (!A and !B)
!C => (A and B)
```

STP Input:

```
a,b,c: BOOLEAN;
wa,wb,wc: BOOLEAN;
ASSERT(wa AND wa);
ASSERT(a => wa);
ASSERT(NOT(a) => NOT(wa));
ASSERT(b => wb);
ASSERT(NOT(b) => NOT(wb));
ASSERT(c => (NOT(a) AND NOT(b)));
ASSERT(NOT(c) => (a AND b));
```

STP OutPut:

```
hive5 [351] ~/HW11 # /home/ff/cs170/bin/easystp test.in
Satisfiable.
ASSERT( b  = TRUE  );
ASSERT( wa  = TRUE  );
ASSERT( c  = FALSE  );
ASSERT( a  = TRUE  );
ASSERT( wb  = TRUE  );
```

5. Sudoku

Main idea. We use 9 variables represent 9 number (1 to 9).
We solve this problem by create constraints . So for each of 9 variables, we create all the possible constraints for the row, columns, 3x3 blocks to make sure that number 1 to 9 are unique in each of row, columns, 3x3 blocks constraints.
Description of boolean variables.

- $a11, a12, \cdots, a99 = 1$
- $b11, b12, \cdots, b99 = 2$
- $c11, c12, \cdots, c99 = 3$
- $d11, d12, \cdots, d99 = 4$
- $e11, e12, \cdots, e99 = 5$
- $f11, f12, \cdots, f99 = 6$
- $g11, g12, \cdots, g99 = 7$
- $h11, h12, \cdots, h99 = 8$
- $i11, i12, \cdots, i99 = 9$

Solution to Sudoku puzzle.

9	8	7	6	5	4	3	2	1
2	4	6	1	7	3	9	8	5
3	5	1	9	2	8	7	4	6
1	2	8	5	3	7	6	9	4
6	3	4	8	9	2	1	5	7
7	9	5	4	6	1	8	3	2
5	1	9	2	8	6	4	7	3
4	7	2	3	1	9	5	6	8
8	6	3	7	4	5	2	1	9

Program used to generate input to STP.
PLEASE LOOK AT NEXT PAGE

```
sodoku = open('sudoku.in','w')
numbers = ['a','b','c','d','e','f','g','h','i']

for n in numbers:
    for i in range(1,10):
        for j in range(1,10):
            sodoku.write(n + str(i) + str(j)+ ',')
sodoku.write('p:BOOLEAN;' + '\n')

#constraints for Row
for j in range(1,10):
    for n in numbers:
        for i in range(1,10):
            temp1 = n + str(i) + str(j)
            temp2 = n + str(1) + str(j)
            if (temp1) == (not temp2):
                sodoku.write('ASSERT(' + n + str(j) + str(1) + '=>' + 'NOT' + '(' + n + str(j) + str(i) + ')' + ')' + ';' + '\n')
#constraints for Column
for i in range(1,10):
    for n in numbers:
        for j in range(1,10):
            temp1 = n + str(i) + str(j)
            temp2 = n + str(i) + str(1)
            if (temp1) == (not temp2):
                sodoku.write('ASSERT(' + n + str(1) + str(i) + '=>' + 'NOT' + '(' + n + str(i) + str(j) + ')' + ')' + ';' + '\n' )

#constraints for 3x3 block
def block(top, down, left, right):
    for n in numbers:
        for t in top:
            for b in down:
                for i in left:
                    for j in right:
                        temp1 = n + str(i) + str(j)
                        temp2 = n + str(t) + str(b)
                        if (temp1) == (not temp2):
                            sodoku.write('ASSERT(' + n + str(t) + str(b) + '=>' + 'NOT' + '(' + n + str(i) + str(j) + ')' + ')' + ';' + '\n')

block(range(1,4),range(1,4),range(1,4),range(1,4))
block(range(1,4),range(4,7),range(1,4),range(4,7))
block(range(1,4),range(7,10),range(1,4),range(7,10))
block(range(4,7),range(1,4),range(4,7),range(1,4))
block(range(4,7),range(4,7),range(4,7),range(4,7))
block(range(4,7),range(7,10),range(4,7),range(7,10))
block(range(7,10),range(1,4),range(7,10),range(1,4))
block(range(7,10),range(4,7),range(7,10),range(4,7))
block(range(7,10),range(7,10),range(7,10),range(7,10))
```

```
sodoku.write('ASSERT(a33 AND a33);\n')
sodoku.write('ASSERT(a57 AND a57);\n')
sodoku.write('ASSERT(a85 AND a85);\n')

sodoku.write('ASSERT(b35 AND b35);\n')
sodoku.write('ASSERT(b83 AND b83);\n')

sodoku.write('ASSERT(c26 AND c26);\n')
sodoku.write('ASSERT(c79 AND c79);\n')

sodoku.write('ASSERT(d53 AND d53);\n')
sodoku.write('ASSERT(d95 AND d95);\n')

sodoku.write('ASSERT(e29 AND e29);\n')
sodoku.write('ASSERT(e44 AND e44);\n')
sodoku.write('ASSERT(e71 AND e71);\n')

sodoku.write('ASSERT(g46 AND g46);\n')
sodoku.write('ASSERT(g78 AND g78);\n')

sodoku.write('ASSERT(h28 AND h28);\n')

sodoku.write('ASSERT(i62 AND i62);\n')
sodoku.write('ASSERT(i99 AND i99);\n')
```