COMPSCI 311: Introduction to Algorithms **Spring 2020**

Homework 6

Released 4/16/2020 Due 4/29/2020 11:59pm in Gradescope

Instructions. You may work in groups, but you must write solutions yourself. List collaborators on your submission. Also list any sources of help (including online sources) other than the textbook and course staff. If you are asked to design an algorithm, please provide: (a) the pseudocode or precise description in words of the algorithm, (b) an explanation of the intuition for the algorithm, (c) a proof of correctness, (d) the running time of your algorithm and (e) justification for your running time analysis.
Submissions. Please submit a PDF file. You may submit a scanned handwritten document, but a typed submission is preferred. Please assign pages to questions in Gradescope.

1. (20 points) **BoxDepth**

a) R as the set of rectangles.

Algorithm:

Function `BoxDepth(`R`):`
  $G \leftarrow \emptyset$ as (V, E) undirected
  for $i = 1$ *to* n **do**
    $V \leftarrow V \cup R_i$
  for $i = 1$ *to* n **do**
    for $j = 1$ *to* n **do**
      if $(i \neq j) \wedge$ `Intersects(`R_i, R_j`)` **then**
        $E \leftarrow E \cup (R_i, R_j)$
  return `MaxClique(`G`)`
Function `Intersects:`
  R_i, R_j
$(l_i, b_i) \leftarrow R_i.leftbottom$
$(l_j, b_j) \leftarrow R_j.leftbottom$
$(r_i, t_i) \leftarrow R_i.righttop$
$(r_j, t_j) \leftarrow R_j.righttop$
if $(l_i < r_j) \wedge (l_j < r_i) \wedge (t_i > b_j) \wedge (t_j > b_i)$ **then**
  return $False$
return $True$

Intuition: Take all the rectangles as the vertices, and intersects as edges. Find max clique in this graph.

Proof: Consider the definition of the problem, we want to find a common intersection point. Clearly, for all the rectangles that pass this point, they should pairly intersect with each other.

Take $S(A, B)$ as A intersects with B.

This is a symmetric and transitive relation, because A intersects B means B intersects A, and because they pairly intersect each other, for any this kind set of rectangles $R_1, ..., R_n$, we have $S(R_1, R_2), S(R_1, R_3), ...S(R_{n-1}, R_n)$, for any intersection link $S(R_a, R_b), S(R_a, R_c)$, we can always find $S(R_a, R_c)$. Therefore, for a set of rectangles that intersects each other, they must share some common intersection point.

In the algorithm, we take the rectangles as the vertices, and the intersections as edges. For the MaxClique, we find the max number of vertices that are pairly connected with each other, same as described above, means they share a common intersection point. Therefore, this algorithm is correct.

Complexity: This algorithm is $O(n^2)$ reduced to MaxClique, because the main part is a nested loop with each $O(n)$ with intersects function $O(1)$, so it is a polynomial-time reduction.

b) **Basic:**

Algorithm:

Function `BoxDepth`(R)**:**
- $max\_size \leftarrow 0$
- **for** $R_i \in R[1...n-1]$ **do**

Function `Helper`(R, R', i, c)**:**
- $c_{max} \leftarrow c$
- **for** $R_j \in R[i+1...n]$ **do**
  - $I \leftarrow$`ConstructSub`(R', R_j)
  - **if** `Area`$(I) > 0$ **then**
    - $c_{max} \leftarrow \max($`Helper`$(R, I, j, c+1), c_{max})$
- **return** c_{max}

Function `ConstructSub`(R_i, R_j)**:**
- $(l_i, b_i) \leftarrow R_i.leftbottom$
- $(l_j, b_j) \leftarrow R_j.leftbottom$
- $(r_i, t_i) \leftarrow R_i.righttop$
- $(r_j, t_j) \leftarrow R_j.righttop$
- **if** $(l_i < r_j) \wedge (l_j < r_i) \wedge (t_i > b_j) \wedge (t_j > b_i)$ **then**
  - $R \leftarrow ((0,0),(0,0))$
- **else**
  - $l \leftarrow max(l_i, l_j)$
  - $b \leftarrow max(b_i, b_j)$
  - $r \leftarrow min(r_i, r_j)$
  - $t \leftarrow min(t_i, t_j)$
  - **if** $l > r$ **then**
    - $(l, b), (r, t) \leftarrow (r, t), (l, b)$
  - $R \leftarrow ((l, b), (r, t))$
- **return** R

Function `Area`(R)**:**
- $(l, b) \leftarrow R.leftbottom$
- $(r, t) \leftarrow R.righttop$
- **return** $(r - l) * (b - t)$

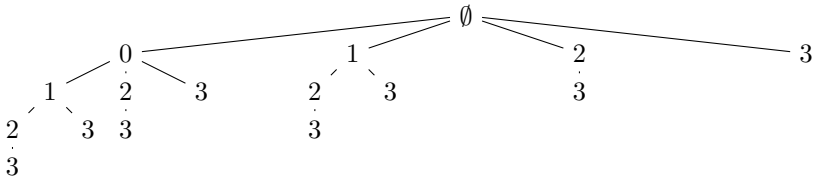
Intuition: For each rectangles, for all the rectangles that intersects with it, find the intersect subrectangle, use this new subrectangle to find the new overlap rectangles. Record the overlap count in the process, and finally select the maximum.

Proof: Very simple. It is just a dynamic programming. When there is only one rectangle, gives 1, correct.

For multiple rectangles share a common intersection area, if another rectangle intersects with this area, then it must share some common intersection area with all previous rectangles.

Therefore, the subproblem is correct, so does the algorithm.

Complexity: $O(n^3)$. Here, we can simply think the process as a tree. For example,



Consider total $n+1$ rectangles to make calculation easier.

We can see that total nodes are

$n + 1 + n + (n-1) + \cdots + 1 + (n-1) + (n-2) + \cdots + 1 + \cdots 1$

$= n + 1 + 1 + 2 + \cdots + n + 2 + 3 + \cdots + n + \cdots + n$

$= n + 1 + n^2 + (n-1)^2 + \cdots + 2^+1 = O(n^3)$

Because it is the whole recurrence tree, the iteration process is already calculated, so the total complexity is $O(n^3)$, polynomial.

Extra Credit:

Algorithm:

Function `BoxDepth(`R`):`

- $(O, C) \leftarrow$`ConstructSegment(`R`)` $L \leftarrow$ as list
- $maxOverlap \leftarrow 0$
- $T \leftarrow IntervalTree()$
- the tree takes the max common overlap as a index of the nodes
- duplicate intervals be treated as two different intervals
- **for** $s \in O \cup C$ **do**
  - $L \leftarrow L \cup s$
- $L.sortby(lambda\ elem:\ lem[0][0])$
- **for** $[(x, y_1), (x, y_2)] \in L$ **do**
  - **if** $[(x, y_1), (x, y_2)] \in O$ **then**
    - T.add((y_1, y_2))
    - $c \leftarrow$T.getMaxCommonOverlapCount((y_1, y_2))
    - **if** $maxOverlap < c$ **then**
      - $maxOverlap \leftarrow c$
  - **else**
    - T.remove((y_1, y_2))

Function `ConstructSegment(`R`):`

- $O \leftarrow \{\}$ as set
- $C \leftarrow \{\}$ as set
- **for** $i = 1$ *to* n **do**
  - $(l_i, b_i) \leftarrow R_i.leftbottom$
  - $(r_i, t_i) \leftarrow R_i.righttop$
  - $O \leftarrow O \cup [(l_i, b_i), (l_i, t_i)]$
  - $C \leftarrow C \cup [(r_i, b_i), (r_i, t_i)]$
- **return** O, C

Intuition: Use the sweep line method, sweep the line from left to right along the x axis, for each edge meets, if it is the left edge of the rectangle, then add it to the interval tree by its y coordinates, and check the current edges with it stored in the tree. If it is the right edge, just remove it from the tree. Track the max overlap during this process.

Proof: Simply a sweep line. Consider the line on a left edge, we have already stored some edges in the tree, that means for these edges, the line is inside these rectangles. So, we only need to find the maximum common overlap of this edge in the tree, because at that point, maximum number of rectangles that the line sweeps overlap on each other.

The line sweep until all the rectangles are scanned, so it is a global optimal

Complexity: $O(n \log n)$. It is dominated by the sort and operation in the tree in the loop. The sort is $O(n \log n)$. In the tree, the operation is $O(\log n)$, and because it is a $O(n)$ loop, in total $O(n \log n)$. The ConstructSegment method is only an $O(n)$ loop.

c) BoxDepth $\leq_P$ MaxClique$\wedge$BoxDepth $\in P$, cannot prove MaxClique$\in P$, so cannot prove $P = NP$

To prove MaxClique $\in P$, we need to find a algorithm that MaxClique $\leq_P$ BoxDepth

2. (10 points) **DNF-SAT**

a) **Algorithm:**

C as the set of clauses

```
Function DNF-SAT(C):
    for C_i ∈ C do
        S ← {} as set
        no_neg = True
        for l_i ∈ C_i do
            if ¬l_i ∈ S then
                no_neg = False
                break for
        if no_neg then
            return True
    return False
```

Intuition: Check that if any clause can be true then return true, else false

Proof: For DNF, because clauses are connected by or logic, so for any clause to be true, the whole proposition is true.

Inside each clause, the literals are connected by and logic. Because we just want to find a satisfiable situation, we can set the literals to be true or false. So, the only situation that a clause is false is that it can never be true, means there is $a \wedge \neg a$ case.

Complexity: $O(n)$. Because the two loops iterates through the literals once which is $O(n)$, and the check in set takes $O(1)$.

b) However, at least convert the CNF to DNF is a NP-Hard problem. $CNF - SAT \leq_P DNF - SAT$ requires that $CNF \rightarrow DNF$ to be P which has not been proved. Therefore, because the prepare for new input is not P, this reduction is not P, therefore cannot prove $P = NP$

3. (30 points) **Magic Boxes**

a) **Algorithm:**

Consider that G has a Hamiltonian cycle, BlackBox return INF if there is no such a cycle.

```
Function HamiltoniaCycle(G):
    (V, E) ← G
    min_length = BlackBox(G)
    for e ∈ E do
        E ← E \ e
        length = BlackBox(G)
        if length>min_length then
            E ← E ∪ e
    return G
```

Intuition: Find the length of the shortest hamiltonian cycle of G and record it as *min_length*. Then, for each edge in G, remove it. If the new graph has shortest hamiltonian cycle larger than ml, add the edge back. After the first iteration, the graph is just the shortest cycle.

Proof: Consider m as the minimum length of hamiltonian cycle in the original G.

In the algorithm, we try to remove each edge e to see if the new length $m' > m$.

Suppose e not on any shortest hamiltonian cycle, remove it will not affect the cycle, safe to remove.

Suppose e on a shortest hamiltonian cycle. Remove it will break at least one such cycle. If there is no other such cycle in the new graph, the shortest length would definitely increase (if no any cycle, length is ∞), means delete it is not safe. So, the final graph would contain at least one hamiltonian cycle and would be the shortest.

Consider for G' as the final graph G, which is not a single hamiltonian cycle. As described above, all the edges in this graph must be in some shortest hamiltonian cycle, and will increase the cycle length if deleted. That is not possible, because in this G, by definition of hamiltonian cycle, there must be at least one vertex that has degree larger than 2, and all edges to it are in some shortest hamiltonian cycle, but for one such cycle, it can only pass 2 of them, while all others are not used. So, delete them will not affect the shortest length. Also, because of this, those edges must have not been met by the algorithm, and can be deleted. As

a result, it is contradictory to the assumption.

So, the algorithm is correct.

Complexity: Consider black box is $O(N^c)$, we have $O(mN^c)$, because the loop is $O(m)$ by itself, and in each iteration BlackBox is executed once, so finally $O(mN^c)$ still polynomial, as it is a polynomial multiples a polynomial.

b) **Algorithm:**

Function `MaxClique(G)`:
- $(V, E) \leftarrow G$
- $max\_size =$ `BlackBox(G)`
- **for** $v \in V$ **do**
  - $V \leftarrow V \setminus v$
  - $size =$ `BlackBox(G)`
  - **if** $size < max\_size$ **then**
    - $V \leftarrow V \cup v$
- **return** G

Intuition: Find the maximum size of the complete subgraph of the original graph. For each vertex in V, remove it from it to see if that size decreases. If decreases, add that vertex back to V.

Proof: Suppose that some vertex v not in the largest complete subgraph, safe to remove clearly.

Consider v in some largest complete subgraph. If there are multiple such subgraph, remove it is safe, because if one such subgraph exists, the max possible clique will not change. This will lead to the situation that only one largest complete subgraph left in the graph.

Consider v in the only largest complete subgraph. Remove it will reduce the max size, not safe as described in the algorithm.

In this case, by iterating through all the vertices, only one of the largest complete subgraph will be left there. So, this algorithm is correct.

Complexity: Consider black box is $O(N^c)$, we have $O(nN^c)$, because the loop is $O(n)$ by itself, and in each iteration BlackBox is executed once, so finally $O(nN^c)$ still polynomial, as it is a polynomial multiples a polynomial.

4. (20 points) **Student Preferences**

We can transfer this problem into a proposition problem.

For each student, we know if that one of the five strong preferences is satisfied, the student is satisfied. Therefore, we can consider a student to be a clause in which the literals are connected by or logic.

For example, student A strongly prefers policy a, b and reject c, the clause would be $a \vee b \vee \neg c$. If Jeff accepts some policy, it should be true in the proposition.

In this case, we can construct the students into a list of n clauses in each has most 5 literals. The goal is, find a combination of the true/false of each literals that make the number of the true clauses in the list to be the maximum.

As described above, it is the exact simulation of the problem, so no more proof needed to make sure that this algorithm gives the correct answer.

Define this problem as a MAX-5-SAT problem.

It is clear that MAX-5-SAT can be reduced from MAX-2-SAT in at least polynomial time, mean that if we prove MAX-2-SAT is NP-hard, then MAX-5-SAT is NP-hard.

Consider a 3SAT instance ϕ with n clauses, each clause $C_i = (x \vee y \vee z) \in \phi$. Add another new variable w_i different from all other variables in ϕ, and replace C_i to

$$x, y, z, w_i (1)$$

$$\neg x \vee \neg y, \neg x \vee \neg z, \neg y \vee \neg z (2)$$

$$x \vee \neg w_i, y \vee \neg w_i, z \vee \neg w_i (3)$$

clauses.

So, $\Phi' = \bigcap_{i=1}^{n} \{x_i, y_i, z_i, w_i, (\neg x_i \vee \neg y_i), (\neg x_i \vee \neg z_i), (\neg y_i \vee \neg z_i), (x_i \vee \neg w_i), (y_i \vee \neg w_i), (z_i \vee \neg w_i)\}$

Here we have two theorems: **Theorem 1:** If some assignment satisfies $(x \vee y \vee z)$, then there is some w_i that satisfies at most 7 of the 10 clauses.

Theorem 2: If some assignment falsifies $(x \vee y \vee z)$, then there is some w_i that satisfies at most 6 of the 10 clauses.

Proof: It is clear that the order of x, y, z does not affect the result, So there are at most 4 cases.

Case 1: $x = y = z = True, x \vee y \vee z = True$

By set $w_i = True$, we have $(1), (2)$ be true, 7 in total.

Case 2: $x = y = True, z = False, x \vee y \vee z = True$

By set $w_i = True$, we have 3 in (1), 2 in (2) and 2 in (3) are true, exactly 7 in total.

Case 3: $x = True, y = z = False, x \vee y \vee z = True.$

By set $w_i = False$, we have 1 in (1) is true, and $(2), (3)$ are true, 7 in total.

Case 4: $x = y = z = False, x \vee y \vee z = False.$

If $w_i = False$, we have true for $(2), (3)$, total 6.

Therefore, 3-SAT can be extended to MAX-2-SAT, as a satisfying assignment for Φ can be extended to a satisfying assignment for Φ' in which exactly 7 clauses in each groups are satisfied.

If Φ is not satisfiable, then there are some $C_i = x, y, z$ in Φ that $x = y = z = False$, in which at most 6 clauses in C_i' can be true.

Therefore, 3-SAT is at least as complex as MAX-2-SAT, mean MAX-2-SAT is NP-hard, so we can prove that MAX-5-SAT is NP-hard, so no polynomial time algorithm can solve this problem.

5. (20 points) **Concatenations**

Suppose $m \leq n$, and L is the length of the longest string in $A \cup B$. Consider that there exists such concatenation u in A, B that $l(u)$ has the minimum length among all possible concatenations.

Claim that $l(u) \leq n^2 L^2$. It comes from: first extract all possible pairs (a_i, b_j), the number of pairs is $mn \leq n^2$. We align them together to create string $u_a = a_{i_1} a_{i_2} ... a_{i_p}, u_b = b_{j_1} b_{j_2} ... b_{j_p}$ which $u_a = u_b = u$. There are up to L different ways to align them, consider there are elemental strings. Also, $l(a_i) \leq L$, so the $l(u) \leq n^2 * L * L = n^2 L^2$

Suppose that this claim is incorrect. As assumption, $l(u) > n^2 L^2$. As described above, this length exceeds the maximum possible alignment combinations, Consider p as the p^{th} string concatenated in u_a, u_b. There must exist positions p, p' that $p < p'$, and by deleting $p, p+1, ..., p'-1$, we still have $u_a' = u_b' = u'$, and $l(u') < l(u)$, gives that u is not the shortest common concatenation. Therefore, the assumption $l(u) > n^2 L^2$ a false assumption, gives that $l(u) \leq n^2 L^2$, which is a polynomial.

6. **(0 points).** How long did it take you to complete this assignment?