Monsoon 2019 | Graduate Algorithms: End-sem | Max
Due: 28 September | | Duration: 1

Graduate Algorithms: End-sem

Name: **Roll No.:**

20% if you write **"I don't know"** for any (sub)question. Not applicable to YES-NO/true-false questions.
LAST PAGE is for ROUGH!

Q1 [5+2.5+2.5+5=15 points] 1. Fill in the blanks below to complete the algorithm that recursively compute size of the smallest vertex cover of a subtree of binary tree.

```
def VC(node r): // compute optimal vertex cover size for subtree rooted at r
    if r is null: return 0

    // vertex cover values that are computed are memoized
    if r.vc is not null: return r.vc // avoid recomputation of memoized values

    // compute optimal vertex cover size when r is part of the cover

    vc_with_r = $\boxed{1 + VC(r.left) + VC(r.right)}$
            (1 line recursive code to compute this value)

    // compute optimal vertex cover size when r is not part of the cover
    vc_wo_r = 0
    if r.left is not null:
        vc_wo_r += $\boxed{1 + VC(r.left.left) + VC(r.left.right)}$
                (1 line recursive code for left subtree of r)
    if r.right is not null:
        vc_wo_r += $\boxed{1 + VC(right.left) + VC(right.right)}$
                (1 line recursive code for right subtree of r)

    // optimal vertex cover size is the best of two cases
    vc = min (vc_with_r, vc_wo_r)
    // memoize the value
    r.vc = vc
    return r.vc
```

2. What is the asymptotic running time of `VC(root of tree T)`? Express the running time in terms of any/all of these: number of nodes n, smallest depth of any leaf s, largest depth of any leaf h, number of leaf nodes l. **Ans:** $\boxed{O(n)}$

Q2 [5+2.5+2.5=10 points] 1. Fill in the blanks in `toposort()` so that it returns all the nodes of a directed acyclic graph according to a topological ordering. `toposort` should run in linear-time.

```
def toposort(Graph G): // output vertices according to topological ordering

    G' = $\boxed{reverse(G)}$
    run DFS(G') and output vertices when they are finished

def hampath(Graph G):
    for each successive pair of vertices (u,v) output by toposort(G):
        if there is no edge from u to v:
            return false
    return true
```

2. Suppose a directed acyclic graph G has a Hamiltonian path. Does `hampath(G)` return true? $\boxed{\text{Yes}}$

3. Suppose a directed acyclic graph G has no Hamiltonian path. Does `hampath(G)` return false? $\boxed{\text{Yes}}$

Q3 [5+2.5+5+2.5+5=20 points] Consider the following "reduction" from 3SAT to Vertex Cover (VC).

```
def myRed(3SAT formula $F$):
    n = number of variables in $F$, m = number of clauses in $F$
    Construct empty graph $G$
    For every variable $X_i$ in $F$:
        add a vertex $v_i^T$ and $v_i^F$
        add edge between these two vertices
    For every clause $C_j = (l_1^j \vee l_2^j \vee l_3^j)$:// $l_k^j$ denotes literals in $C_j$
        add vertices $u_1^j, u_2^j, u_3^j$
        add edges between these three vertices
        for $k = 1,2,3$:
            if literal $u_k^j$ is a variable $X_i$:
                add edge between $u_k^j, v_i^T$
            if literal $u_k^j = \overline{X_i}$ (negation of a variable):
                add edge between $u_k^j, v_i^F$
    Set $k$ in some manner that will be derived below
    Return $G, k$
```

1. If F is satisfiable, then G has a vertex cover with $\leq$ $\boxed{n+2m}$ vertices

2. Any vertex cover of G must have $\geq$ $\boxed{n+2m}$ vertices

3. If F is not satisfiable, then any vertex cover of G must have $>$ $\boxed{n+2m}$ vertices

4. In the above reduction, set $k =$ $\boxed{n+2m}$

5. Write the algorithm for a reduction from 3SAT to VC that outputs a graph with at most $3m$ vertices (notice that the above reduction outputs a graph with $3m + 2n$ vertices).

```
def myBetterRed(3SAT formula $F$):
    n = number of variables in $F$, m = number of clauses in $F$
    Construct empty graph $G$
    For every variable $X_i$ in $F$:
        // add, if necessary (write in words, not code)


    For every clause $C_j = (l_1^j \vee l_2^j \vee l_3^j)$:// $l_k^j$ denotes literals in $C_j$
        // add, if necessary (write in words, not code)
        [create triangle, one node for each literal and storeliterals with nodes]


    Set $k$=[2m]

    // add, if necessary (write in words, not code)
        [add edges between nodes that correspond to a variable and its negation]


    Return $G, k$
```

Scanned by CamScanner

Q4 [2.5+10+2.5=15 points] Consider the following algorithm to return a hitting set of a set system $\{U, T = \{S_1, S_2, S_3, \dots S_k\}\}$ in which U denotes an universe of elements and the S_i's are subsets over U.
Do not ask what the Hitting Set problem is about.

```
def myHS($U, T = \{S_1, S_2, \dots S_k\}$):
    $h = \{\}$
    while $T$ is not-empty:
        arbitrarily choose some subset $S$ from $T$
        add elements of $S$ to $h$
        remove all subsets in $T$ that intersect with $S$
    return $h$
```

1. Does `myHS` return some hitting set of (U, T)? Say YES/NO. $\boxed{\text{Yes}}$
2. If yes, <u>state</u> and <u>derive</u> the relative-approximation ratio of `myHS`. The ratio can be stated in terms of all/any of k (the number of subsets in T), n (the number of elements in U), s (size of the largest subset in T), m (size of the smallest subset in T), b (the largest number of subsets any element belongs to).
If no, give a counter-example and explain why the output of `myHS` is not a proper hitting set.

Let $S_{i_1}, S_{i_p}, \dots S_{i_p}$ be the subsets chosen by myHS. Since these subsets do not have any overlap, therefore, OPT must contain at least one element from each of them; therefore, $OPT \geq p$.
Let HS denote the size of the hitting set returned by myHS. Since $HS \leq p \times s$, therefore, $HS \leq OPT \times s$, i.e., myHS has a relative approximation ratio of s (the size of the largest subset in T).

3. Suppose the subset S in `myHS` is always chosen to be the smallest subset in T. In that case will `myHS` return the smallest hitting set? Say YES/NO. $\boxed{\text{No}}$

Q5 [5+5=10 points] Suppose we want to obtain 1000/999-relative approximation to the solution of a Knapsack instance and we run the scaling-based approximation algorithm that was studied in lecture.

1. In that algorithm, the original value, say v_i of item i, is modified, to v_i'. Explain how v_i' should be computed from v_i using n,v_{max} and constants. Keep the specific approximation ratio in mind.

$v_i' = \lfloor \frac{1000 n v_i}{v_{max}} \rfloor$

2. After scaling, the Knapsack instance with the modified values is solved using a dynamic programming algorithm. What would be the running time of this algorithm in terms of the number of items n? Explain.

$T = O(n \sum_i v_i)$. Now, $\sum_i v_i \leq \sum_i \frac{1000 n v_i}{v_{max}} \leq 1000n^3$. Therefore, $T = O(n^3)$.

Q8 [5 points] Let $\langle a_0 \ldots a_{n-1}\rangle$ denote the coefficients of the degree-$(n-1)$ polynomial $A(x)$. Fill in the blanks to give us a $DFT_n(A_n)$ computation algorithm that runs in $O(n \log n)$ time.

```
def $DFT_n(\langle a_0 \ldots a_{n-1}\rangle)$:
    if n=1: return $\langle a_0\rangle$
    else:
        $\langle y_0^{even}, \ldots y_{n/2-1}^{even}\rangle \leftarrow DFT_{n/2}(\langle a_0, a_2, \ldots a_{n-2}\rangle)$
        $\langle y_0^{odd}, \ldots y_{n/2-1}^{odd}\rangle \leftarrow DFT_{n/2}(\langle a_1, a_3, \ldots a_{n-1}\rangle)$
        for k=0 to n−1:

            $y_k = \boxed{y^{even}_{k \bmod n/2} + e^{2\pi i k/n} y^{odd}_{k \bmod n/2}}$
    return $\langle y_0, \ldots y_{n-1}\rangle$
```

Q9 [5 points] This question is regarding Hirshberg's technique for computing the optimal edit sequence from $A[1\ldots m]$ to $B[1\ldots n]$. Recall that $H(i,j)$ is a value such that some optimal edit sequence from $A[1\ldots i]$ contains an optimal edit sequence from $A[1\ldots m/2]$ to $B[1\ldots H(i,j)]$. Give recursive expressions to compute $T(m,n)$ which is the running time of the edit-sequence dynamic programming algorithm using Hirshberg's space optimization.

$T(m,n) =$

$$T(m,n) = \begin{cases} O(n) & \text{if } m \le 1 \\ O(m) & \text{if } n \le 1 \\ O(mn) + T(m/2, Half(m,n)) + T(m/2, n - Half(m,n)) & \text{otherwise} \end{cases}$$

Q10 [2.5 points] Select the time complexity to construct a 1D balanced interval tree from n intervals.

$O(\log n)$ [✓ $O(n)$] $O(n \log n)$ $O(n^2)$ $O(n^2 \log n)$

Q11 [2.5 points] Let n_1 be the number of binary search trees (BSTs) that can be created using the values 54, 12, 91, 100, 98, 7, 29, 88, 16. Let n_2 be the number of BSTs that can be created using the values 4, 12, 91, 100, 98, 7, 29, 88, 116, and n_3 be the number of BSTs that can be created using the values 54, 12, 9, 10, 8, 7, 29, 88, 16. Relate the values n_1, n_2, n_3 (using $\le, <, =, >, \ge$). $\boxed{n_1 = n_2 = n_3}$

Q12 [2.5 points] Let $J(n)$ be the total number of nodes in a 1D range tree on n values that is created in a dynamic manner (so you cannot assume any fixed structure of the tree or any ordering of the values that were inserted in the tree). Give an expression for $J(n)$. Write an exact expression or tight upper and/or lower bounds with/without asymptotic notation (basically, the tightest that you can derive). $\boxed{J(n) = 2n - 1}$

Q13 [5 points] Consider a disjoint-set implemented using "shallow threaded trees" (discussed in class as shallow reversed trees + threading + wtd. union) — don't ask what this is. State the *total* complexity of making n MakeSet and n^2 Union calls. There could be duplicate Union calls also or in-effectual calls (e.g., Union(x,y) where x and y already belong to the same set — Union(x,y) will not change anything in this case).

$\boxed{O(n + n\log n + (n^2 - n)) = O(n^2 + n \log n) = O(n^2)}$

Q14 [1+2+2=5 points] Determine (i) running time, (ii) additive and (iii) relative approximation ratio of this.

```
def PlanarChromaticBetter(planar G):
    if G has no edge: return 1
    if G is bipartite : return 2
    else return 4
```

(i) running time = $\boxed{O(V+E)}$ (ii) additive approx. ratio = $\boxed{1}$ (iii) relative approx. ratio = $\boxed{4/3}$

Q6 [5+5+2.5+2.5+2.5+2.5=20 points] The 3WAYPARTITION problem takes as input an array A of positive integers (duplicates allowed) and returns YES if A can be partitioned into three disjoint subsets B, C, D such that $B \cup C \cup D = A$ and $\sum_{x \in B} x = \sum_{x \in C} x = \sum_{x \in D} x$. Answer these questions below to give a dynamic programming based algorithm for solving 3WAYPARTITION. *Hint: Think of SUBSET-SUM.*

(a) Define a suitable subproblem $M[\ldots]$. *Hint: These could be Boolean values.*

Boolean variable $M[i, s, t]$ = True iff there are three partitions of $\{a_1, \ldots, a_i\}$ with sums s, t and $\sum_{j=1}^{i} a_j - s - t$, respectively.

(b,c) Write a recursive formula to compute the values of $M[\ldots]$ including its base case(s).

Base case would be: $M[3, s, t]$ = True iff $s \in \{a_1, a_2, a_3\}$ and $t \in \{a_1, a_2, a_3\} \setminus \{s\}$.
Recurrence would be: $M[i, s, t]$ = True iff $M[i-1, s-A_i, t] \vee M[i-1, s, t-A_i] \vee M[i-1, s, t]$ is True.

(d) Explain a suitable memoization data-structure and how to fill this data-structure.

Let $S = \sum_i A_i / 3$.
Use a three-dimensional array of dimension $n \times S \times S$ to store the memoized values. First, compute $M[3, s, t]$ for all s and t starting from $s = 1 \ldots S$ and $t = 1 \ldots S$. Then, compute $A[i, s = 1 \ldots S, t = 1 \ldots S]$ in increasing manner of i.

(e) How to solve the 3WAYPARTITION problem using the memoization data structure?

Look at $M[n, S, S]$.

(f) Discuss the time-complexity of your algorithm, including any possible optimization.

The entire table needs to be computed and for computing each entry requires constant time. So the time-complexity is $O(nS^2)$.

(g) Discuss the space-complexity of your algorithm, including any possible optimization.

$A[i, \ldots]$ can be computed solely from $A[i-1, \ldots]$. Therefore, only two values of the first index needs to remain in memory. Thus space complexity is $O(S^2)$.

Q7 [2.5 points] Suppose we want to compute the diameter of a tree (not necessarily binary). Let *diam*(v) denote the diameter of the subtree rooted at v. For any internal node v, *diam*(v) can be calculated solely from the *diam*$(v.child)$ values correspondong to all children of v. YES/NO $\boxed{\text{No}}$