CS60007 Algorithm Design and Analysis 2018 Assignment 1

Palash Dey and Swagato Sanyal
Indian Institute of Technology, Kharagpur

Please submit the solutions of the problems 6, 11, 12 and 13 (written in red color). You are strongly encouraged to do all the problems for your own practice. The deadline is August 13, 2018.

This is a preliminary version and may contain errors. Please send an email to the instructors if you find any error. Thank you for your cooperation.

---

1. There is a sequence of n activities $a_1, a_2, \ldots, a_n$ with corresponding utilities $u_1, u_2, \ldots, u_n$. You wish to perform all these n activities according to this sequence within k days. If you perform the activities from a_i to a_j for some $1 \leqslant i \leqslant j \leqslant n$ on the j-th day, then your utility U_j for the j-th day is $\max\{u_\ell : i \leqslant \ell \leqslant j\}$. Your total utility is $\sum_{\ell=1}^{k} U_\ell$. Design a greedy algorithm to find the sequence of activities you will perform on every day which maximizes your total utility.

2. ~~There are n jobs $J_1, J_2, \ldots, J_n$ with duration $a_1, a_2, \ldots, a_n \in \mathbb{N}^+$ and deadlines $d_1, d_2, \ldots, d_n \in \mathbb{N}$ respectively. All the jobs are available at the beginning (that is at time 0). If a job J_ℓ is completed at time t_ℓ with $t_\ell > d_\ell$ then the utility u_ℓ incurred for job J_ℓ is -1 (which is a penalty); if $t_\ell \leqslant d_\ell$, then $u_\ell = 1$ (which is a profit). You have only one machine. So, you cannot process more than one jobs simultaneously. Every job, once started, must run till it finishes. Design a greedy algorithm to find the schedule for these n jobs which maximizes $\sum_{\ell=1}^{n} u_\ell$.~~

3. We are given a decimal string $s = (a_1, a_2, \ldots, a_n)$ of n digits (that is $a_i \in \mathbb{N}$ and $0 \leqslant a_i \leqslant 9$) and a positive integer k. Design an algorithm for find the lexicographic minimum string which can be obtained from s by performing at most k swaps. Note that we are allowed to swap consecutive elements only.

4. Suppose you have to give Re N to your friend. You have enough number of $500, 200, 100, 50, 20, 10$ rupee notes each at your disposal. Your goal is to give Re N to your friend with minimum number of notes. For example, Re 600 can be changed

using 3 Re 200 notes as well as using 1 Re 500 note and 1 Re 100 note. However, the later one uses minimum number of notes.

- ▷ Either prove correctness or provide counter example of the following greedy strategy: keep picking highest denomination as much as you can!
- ▷ Provide a set of denominations for which the above greedy strategy will fail.

5. An independent set of an undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is a subset $\mathcal{W} \subseteq \mathcal{V}$ such that there does not exist any edge $e = \{u, v\} \in \mathcal{E}$ with both end points in $\mathcal{W}$ (that is, $|e \cap \mathcal{W}| \leqslant 1$). An undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is called an interval graph if every vertex $v \in \mathcal{V}$ can be associated with some interval $\mathcal{I}_v = [a, b)$ in $\mathbb{R}$ such that there is an edge e between u and v if and only if $\mathcal{I}_u \cap \mathcal{I}_v \neq \emptyset$. Let $\mathcal{G}$ be an interval graph and $\{\mathcal{I}_v : v \in \mathcal{V}\}$ be the intervals associated with its vertices. Then design an algorithm for computing the size of a maximum independent set of $\mathcal{G}$.

6. **[15 marks]** Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be a connected, weighted graph. Let $\mathcal{T}$ and $\mathcal{T}'$ be two MSTs of $\mathcal{G}$ and $\alpha \in \mathbb{R}$. Then the number of edges in $\mathcal{T}$ of weight α is the same as the number of edges in $\mathcal{T}'$ of weight α.

7. Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be a connected, weighted graph, $v \in \mathcal{V}$ be any vertex, and e be an edge with minimum weight among all the edges that incident on v. Prove that there exists a MST which includes the edge e.

8. Let $\mathcal{G}$ be a connected, weighted graph. Prove that, if all edge weights in $\mathcal{G}$ are distinct, then $\mathcal{G}$ has exactly one MST.

9. Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be a connected, weighted graph. Let $\mathcal{S} \subset \mathcal{V}$ with $\mathcal{S} \neq \emptyset$, $\mathcal{S} \neq \mathcal{V}$, and $e' = \text{argmin}_{e \in \mathcal{E} : e = \{u,v\}, u \in \mathcal{S}, v \in \mathcal{V} \setminus \mathcal{S}} \{w(e)\}$. Then there exists a MST T where $e' \in T$.

10. Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be a connected, weighted graph and e be an edge with minimum weight in $\mathcal{G}$. Then prove that there exists a MST which includes the edge e.

11. **[Independent Set of Binary Tree]** Refer to the definition of Independent Set of an undirected graph in problem 5. In this exercise you will compute a maximum weighted independent set of a binary tree. Your algorithm will be given as input a pointer to the root of a binary tree. Each node v has a weight $w(v)$ and two pointers $\texttt{left}(v)$ and $\texttt{right}(v)$ leading to the left and right children of v respectively; if a child is absent, the corresponding pointer is NULL. The weight of an independent set in a tree is the sum of the weights of the nodes in it. A maximum weight independent set is an independent set whose weight is maximum.

    (a) **[4 marks]** Derive a recursive relation for the weights of a maximum weight independent set of various subtrees.

    (b) **[4 marks]** Design an efficient algorithm to compute the weight of a maximum weight independent set of the input tree.

(c) **[4 marks]** Modify your algorithm to also compute a maximum weight independent set of the input tree, in addition to its weight.

(d) **[3 marks]** Discuss running times of your algorithms.

12. Suppose a trader has a container vehicle capable of shipping at most W amount of goods (W is an integer). Suppose that he has n items $\{1, \dots, n\}$. Each item i has a price v_i and an integral weight w_i. The trader wants to ship a set $S \subseteq \{1, \dots, n\}$ of items of highest possible total price subject to the constraint that the total weight of the items is at most W.

    (a) **[1.5 marks]** Mathematically formulate the constraint of the trader.

    (b) **[1.5 marks]** Mathematically formulate the objective that the trader wants to maximize subject to his constraint.

    (c) [4 marks] For an integer $\mathcal{W}$, let $c(i, \mathcal{W})$ be the maximum total price of a set of items in $\{1, \dots, i\}$ with total weight at most $\mathcal{W}$. Derive a recursive relation for $c(i, \mathcal{W})$.

    (d) [4 marks] Design an efficient algorithm for computing $c(n, W)$.

    (e) [3 marks] Modify your algorithm to also find a set of items that correspond to $c(n, W)$.

    (f) [1 marks] Discuss running times of your algorithms.

13. Consider the following properties of an undirected graph G with n vertices.

    (a) G is connected.

    (b) G is acyclic.

    (c) G has $n - 1$ edges.

    Prove that if G satisfies any two of the above three properties, then G also satisfies the third property, i.e.,

    (i) **[2 marks]** if G satisfies properties (a) and (b), then G satisfies property (c),

    (ii) **[2 marks]** if G satisfies properties (a) and (c), then G satisfies property (b),

    (iii) **[2 marks]** if G satisfies properties (b) and (c), then G satisfies property (a).

    As you must know, graphs satisfying these three properties are called trees.

14. We are given two English strings s_1 and s_2 of length n each. Design a dynamic programming based algorithm to find the minimum number of the following operations that one requires to perform on s_1 to convert it into s_2.

    ▷ Insert any symbol at any position of the string.

- $\triangleright$ Delete any symbol from the string.
- $\triangleright$ Replace any symbol in the string by another symbol.

15. A string s is called a sub-sequence of another string t if s can be obtained from t by deleting some symbols from s. Design a dynamic programming based algorithm which finds a longest common sub-sequence of two input strings.

16. Design a dynamic programming based algorithm which finds a subset of a set of integers (given as an array of integers as input) which has maximum sum of its elements.

17. Design a dynamic programming based algorithm for problem 4 which works for any set of denominations.

18. There are n jobs $J_1, J_2, \ldots, J_n$ with start and end times $s_1, s_2, \ldots, s_n \in \mathbb{N}$ and deadlines $t_1, t_2, \ldots, t_n \in \mathbb{N}$ respectively with $t_\ell > s_\ell$ for every $\ell \in [n]$. The weight of the job J_ℓ is $w_\ell \in \mathbb{N}$ for $\ell \in [n]$. You have only one machine. So, you cannot process more than one jobs simultaneously. Every job, once started, must run till it finishes. Design a dynamic programming based algorithm for the problem of scheduling a subset $\mathcal{S} \subseteq \{J_\ell : \ell \in [n]\}$ of jobs which maximizes $\sum_{J_\ell \in \mathcal{S}} w_\ell$.

19. Suppose we have a stick of length $n \in \mathbb{N}^+$. Design a dynamic programming based algorithm to break the stick into at least 2 pieces of integral lengths which maximizes the product of the lengths of the pieces.

20. Suppose we have a stick of length $n \in \mathbb{N}^+$. Let $p : \mathbb{N}^+ \longrightarrow \mathbb{N}^+$ be a function which maps every positive integer ℓ to the utility of a stick of length ℓ. Design a dynamic programming based algorithm to break the stick into pieces which maximizes the total utility (the sum of utilities of the pieces).

Solutions

Problem 6: For the sake of contradiction, let us assume that there exist two minimum spanning trees $\mathcal{T}_1$ and $\mathcal{T}_2$ of a graph $\mathcal{G}$ and $\alpha \in \mathbb{R}$ such that the number of edges of weight α in $\mathcal{T}_1$ and $\mathcal{T}_2$ are not the same. Let us consider the set $\mathcal{S} = \{\mathcal{T} : \mathcal{T}$ is a minimum spanning tree of $\mathcal{G}$ and there exists $\alpha \in \mathbb{R}$ such that the number of edges of weight α in $\mathcal{T}_1$ and $\mathcal{T}$ are not the same$\}$. By our assumption $\mathcal{S}$ is not an empty set since we have $\mathcal{T}_2 \in \mathcal{S}$. Also since the input graph $\mathcal{G}$ is finite, $\mathcal{S}$ is also a finite set. Let $\mathcal{T}'$ be a minimum spanning tree in $\mathcal{S}$ for which $|\mathcal{T}_1 \Delta \mathcal{T}'|$ is minimum possible where $\mathcal{T}_1 \Delta \mathcal{T}'$ denotes symmetric difference of the set of edges of $\mathcal{T}_1$ and $\mathcal{T}'$. Such a $\mathcal{T}'$ always exists since $\mathcal{S}$ is a non-empty and finite set. Let $e \in \mathcal{T}_1 \setminus \mathcal{T}'$ be any edge in $\mathcal{T}_1 \setminus \mathcal{T}'$. Let us consider the subgraph $\mathcal{F} = \mathcal{T}' \cup \{e\}$ of $\mathcal{G}$. Since $\mathcal{T}'$ is a tree, there exists a unique cycle $\mathcal{C}$ in $\mathcal{F}$. Moreover, the $\mathcal{C}$ contains the edge $\{e\}$. Let the two components of $\mathcal{T}_1 \setminus \{e\}$ be $\mathcal{H}_1$ and $\mathcal{H}_2$. Since the cycle

$\mathcal{C}$ includes e, there exists an edge $f \in \mathcal{C} \setminus \{e\}$ with one end point in $\mathcal{H}_1$ and the other end point in $\mathcal{H}_2$. Clearly $f \notin \mathcal{T}_1$ since otherwise there would be an edge (namely f) between $\mathcal{H}_1$ and $\mathcal{H}_2$ in $\mathcal{T}_1 \setminus \{e\}$. We claim that $w(f) \leqslant w(e)$. Indeed, otherwise, $\mathcal{F}_1 = \mathcal{F} \setminus \{f\}$ is a spanning tree of $\mathcal{G}$ of weight less than $\mathcal{T}'$ which contradicts our assumption that $\mathcal{T}'$ is an minimum spanning tree of $\mathcal{G}$. So we have $w(f) \leqslant w(e)$. We also have $w(f) \geqslant w(e)$ since otherwise $(\mathcal{T}_1 \setminus \{e\}) \cup \{f\}$ would be a spanning tree of $\mathcal{G}$ of weight less than $\mathcal{T}_1$ contradicting our assumption that $\mathcal{T}_1$ is an minimum spanning tree of $\mathcal{G}$. Hence we have $w(f) = w(e)$. However, then $\mathcal{F}_1$ is a spanning tree of $\mathcal{G}$ of weight same as $\mathcal{T}'$ and thus itself an minimum spanning tree of $\mathcal{G}$. Moreover, for every positive integer α, the number of edges of weight α in $\mathcal{F}_1$ and $\mathcal{T}'$ are the same. Hence we have $\mathcal{F}_1 \in \mathcal{S}$. However, we have $|\mathcal{T}_1 \Delta \mathcal{F}_1| < |\mathcal{T}_1 \Delta \mathcal{T}'|$ which contradicts our choice of $\mathcal{T}'$. This concludes the proof of the statement. □

Problem 11

(a) Let v be either NULL, or a pointer to a vertex of the tree. Let $\mathsf{MIND}(v)$ be defined as follows:

$$\mathsf{MIND}(v) = \begin{cases} 0 & \text{if } v \text{ is NULL,} \\ \text{the size of a maximum independent set of the subtree rooted at } v & \text{otherwise.} \end{cases}$$

Clearly,

$$\mathsf{MIND}(v) = 1 \quad \text{if } v \text{ is a leaf.}$$

Now, let v be an internal node, and let v_ℓ and v_r be its left and right children respectively. If v is NULL, assume that v_ℓ and v_r are both NULL.

case 1: A maximum independent set of the subtree rooted at v does not include v. Then, it is the union of maximum independent sets of the subtrees rooted at v_ℓ and v_r.

$$\mathsf{MIND}(v) = \mathsf{MIND}(v_\ell) + \mathsf{MIND}(v_r).$$

case 2: A maximum independent set of the subtree rooted at v includes v. Then, it is the union of $\{v\}$ and maximum independent sets of the subtrees rooted at the grandchildren of v. Note that some of the grandchildren may be NULL.

$$\mathsf{MIND}(v) = 1 + \mathsf{MIND}((v_\ell)_\ell) + \mathsf{MIND}((v_\ell)_r) + \mathsf{MIND}((v_r)_\ell) + \mathsf{MIND}((v_r)_r).$$

Combining the two subcases above, we get the following recurrence relation for $\mathsf{MIND}(v)$.

$$\mathsf{MIND}(v) = \begin{cases} 0 & \text{if } v \text{ is NULL,} \\ 1 & \text{if } v \text{ is a leaf,} \\ \max\{\mathsf{MIND}(v_\ell) + \mathsf{MIND}(v_r), \\ 1 + \mathsf{MIND}((v_\ell)_\ell) + \mathsf{MIND}((v_\ell)_r) + \\ \mathsf{MIND}((v_r)_\ell) + \mathsf{MIND}((v_r)_r)\} & \text{otherwise} \end{cases}$$

Notice that in the recursive expression of $\mathrm{MIND}(v)$ only vertices whose heights are strictly less than that of v are used.

(b) The above recurrence reveals an optimal substructure property. Notice that a recursive algorithm based on part (a) will suffer from the "overlapping subproblems". So, how about trying dynamic programming? We are given the root pointer. So it may be convenient to retain the recursive structure of our algorithm, and do memoization. Let r be the input root pointer. Let $\mathrm{MIND}(v)$ be a global table, with one entry for each vertex. FINDMIND(r) returns the size of a maximum independent set of the input tree.

Algorithm 1 FINDMIND(r)

```
1: Do a breadth-first traversal on the tree. For each vertex $v$, $\mathrm{MIND}(v) \leftarrow 0$.
2: return IND(r).
```

Algorithm 2 $\mathrm{IND}(v)$

```
1: if $v = \mathrm{NULL}$ then
2:     return 0.
3: end if
4: if $\mathrm{MIND}(v) > 0$ then
5:     return $\mathrm{MIND}(v)$.
6: end if
7: $\mathrm{MIND}(v) \leftarrow \max\{\mathrm{IND}(v_\ell) + \mathrm{IND}(v_r), 1 + \mathrm{IND}((v_\ell)_\ell) + \mathrm{IND}((v_\ell)_r) + \mathrm{IND}((v_r)_\ell) + \mathrm{IND}((v_r)_r)\}$.
8: return $\mathrm{MIND}(v)$.
```

(c) For each v maintan the information of whether or not v is included in the maximum independent set of the subtree rooted at v that your algorithm computes. Let it be stored in the global table ININD. $\mathrm{IND}_{\mathrm{mod}}$ is the modified IND function. $\mathrm{IND}_{\mathrm{mod}}$ fills up the table ININD. Finally, output the set of vertices v for which $\mathrm{ININD}[v] = 1$.

(d) Running time is $O(|V|)$.

Problem 12

(a) $\sum_{i \in S} w_i \leqslant W$.

(b) $\sum_{i \in S} v_i$.

(c) Let T be a set of items from $\{1, \ldots, i\}$ with total weight at most W of maximum total price, i.e., $\sum_{j \in T} v_j = c(i, W)$.

Algorithm 3 $\text{IND}_{\text{mod}}(v)$

```
1: if $v = \text{NULL}$ then
2:     return 0.
3: end if
4: if $\text{MIND}(v) > 0$ then
5:     return $\text{MIND}(v)$.
6: end if
7: $val1 \leftarrow \text{IND}(v_\ell) + \text{IND}(v_r)$.
8: $val2 \leftarrow 1 + \text{IND}_{\text{mod}}((v_\ell)_\ell) + \text{IND}_{\text{mod}}((v_\ell)_r) + \text{IND}_{\text{mod}}((v_r)_\ell) + \text{IND}_{\text{mod}}((v_r)_r)$.
9: if $val1 \geqslant val2$ then
10:     $\text{ININD}[v] \leftarrow 0$.
11:     $\text{MIND}(v) \leftarrow val1$.
12:     return $\text{MIND}(v)$.
13: end if
14: $\text{ININD}[v] \leftarrow 1$.
15: $\text{MIND}(v) \leftarrow val2$.
16: return $\text{MIND}(v)$.
```

case1: $i \notin T$: In this case T is also a set of items from $\{1, \ldots, i-1\}$ with total weight at most W of maximum total price. Thus $c(i, W) = c(i-1, W)$.

case 2: $i \in T$: In this case, $w_i \leqslant W$ and $T \setminus \{i\}$ is a set of items from $\{1, \ldots, i-1\}$ with total weight at most $W - w_i$ of maximum total price. Thus, $c(i, W) = v_i + c(i-1, W - w_i)$.

From the above two cases we have that

$$c(i, W) = \begin{cases} 0 & \text{if } i = 1 \text{ and } w_i > W, \\ v_i & \text{if } i = 1 \text{ and } w_i \leqslant W, \\ c(i-1, W) & \text{if } i > 1 \text{and } w_i > W, \\ \max\{c(i-1, W), v_i + c(i-1, W - w_i)\} & \text{if } i > 1 \text{and } w_i \leqslant W. \end{cases}$$

(d) Use dynamic programming. A recursive program will suffer from overlapping subproblems. Recall that w_i's are all integers. Let the global table T stores the solutions of various $c(i, W)'s$. The first step is initializing all its entries to -1. Then update it using the algorithm KNAPSACK. Finally return $T(n, W)$. We use memoization here; but you can do iterative table-filling also.

(e) Easy. Modify the KNAPSACK algorithm to also store whether or not i is included in the solution. Let this information be kept in the 2-dimensional global array A; $A(i, w)$ stores whether or not the element i is included in the optimal solution when the set of items is $\{1, \ldots, i\}$ and the weight limit is W. The algorithm KNAPSACK-MODIFIED (see pseudocode) fills up the table A. The set of items can be retrieved from the table A as follows: First, check if $A(n, W) = 1$. If $A(n, W) = 1$, include n in the set,

Algorithm 4 KNAPSACK (i, W)

```
1:  if i=0 then
2:      return 0.
3:  end if
4:  if $T(i, W) > -1$ then
5:      return $T(i, W)$.
6:  end if
7:  $val1 \leftarrow$ KNAPSACK$(i-1, W)$.
8:  if $w_i > W$ then
9:      $T(i, W) \leftarrow val1$.
10:     return $T(i, W)$.
11: end if
12: $val2 \leftarrow$ KNAPSACK$(i-1, W-w_i) + v_i$.
13: if $val1 \geqslant val2$ then
14:     $T(i, W) \leftarrow val1$.
15: else
16:     $T(i, W) \leftarrow val2$.
17: end if
18: return $T(i, W)$.
```

and then check if $A(n-1, W-w_n) = 1$ to decide whether or not to include $n-1$. If $A(n, W) = 0$, then do not include n in the set, and check if $A(n-1, W) = 1$ to decide whether or not to include $n-1$. Continue similarly. The algorithm FINDSET (see pseudocode) does this.

(f) Same order as the number of subproblems. $O(nW)$.

Problem 13 Let $G = (V, E)$.

(i) The proof is by induction on n.
Base case: $n = 1$. Number of edges is $0 = n - 1$.
Induction step: Assume that the hypothesis is true for all graphs of at most $n-1$ vertices, for $n \geqslant 1$. Now consider a graph G with n vertices that satisfies properties (a) and (b). Now consider any maximal path $v_1, \dots, v_k$ in G[1]. Since the graph is acyclic, from the maximality of the path it follows that the degree of vertex v_1 (also v_k) is 1[2]. Consider the subgraph G' of G induced by the vertex set $V \setminus \{v_1\}$. Note that the number of edges in G' is $|E| - 1$. Also, observe that G' is connected and acyclic. Since G' has $n-1$ vertices, by inductive hypothesis, we have that the number of edges $|E| - 1$ in G' is equal to $(n-1) - 1 = n - 2$, which implies that $|E| = n - 1$ and the proof is complete.

[1]a path is maximal if it cannot be extended any more in either direction.
[2]such vertices are called leaves.

Algorithm 5 KNAPSACK-MODIFIED (i, W)

```
 1: if i=0 then
 2:     return 0.
 3: end if
 4: if $T(i, W) > -1$ then
 5:     return $T(i, W)$.
 6: end if
 7: $val1 \leftarrow$ KNAPSACK$(i-1, W)$.
 8: if $w_i > W$ then
 9:     $A(i, W) \leftarrow 0$.
10:     $T(i, W) \leftarrow val1$.
11:     return $T(i, W)$.
12: end if
13: $val2 \leftarrow$ KNAPSACK$(i-1, W-w_i) + v_i$.
14: if $val1 \geqslant val2$ then
15:     $A(i, W) \leftarrow 0$.
16:     $T(i, W) \leftarrow val1$.
17: else
18:     $A(i, W) \leftarrow 1$.
19:     $T(i, W) \leftarrow val2$.
20: end if
21: return $T(i, W)$.
```

(ii) Proof by contradiction. Assume that G satisfies properties (a) and (c), and G has a cycle. Then we modify G in the following way: As long as G has a cycle, remove an edge from that cycle. Notice that these operations preserve the connectivity of G. Thus at the end we get a graph which is connected and acyclic, but has strictly less than $n-1$ edges. This contradicts (i).

(iii) Proof by contradiction. Assume that G satisfies properties (b) and (c), and G is not connected. Thus G has at least two connected components. Now modify G as follows: As long as G is not connected, add an edge between two vertices in two different connected components. Note that these operations preserve acyclicity of G. At the end, we get a graph which is connected, acyclic and has strictly more than $n-1$ vertices. This contradicts (i).

Algorithm 6 FINDSET

```
1: weight $\leftarrow W$.
2: $S \leftarrow \emptyset$.
3: for $i = n$ to 1 do
4:     if $A(i, \text{weight}) = 1$ then
5:         $S \leftarrow S \cup \{i\}$.
6:         weight $\leftarrow$ weight $- w_i$.
7:     end if
8: end for
9: return S.
```