1

Final Exam in Algorithms - CS-250
Exam date: 16 January 2021
Teacher: Ola Svensson

With Solutions

SCIPER : **111111**

Signature :

- You are only allowed to have a handwritten A4 page written on both sides.
- Communication, calculators, cell phones, computers, etc... are not allowed.
- Your explanations should be clear enough and in sufficient detail that a fellow student can understand them. In particular, do not only give pseudocode without explanations. A good guideline is that a description of an algorithm should be such that a fellow student can easily implement the algorithm following the description.
- You are allowed to refer to material covered in the lectures including algorithms and theorems (without reproving them).
- Please answer all questions within the designated boxes (otherwise your answer may not be accurately scanned). At the end of the exam there are five extra pages if you need additional space for your answers.
- **Do not touch until the start of the exam.**

Good luck!

Problem 1 / 20 points	Problem 2 / 15 points	Problem 3 / 17 points	Problem 4 / 17 points	Problem 5 / 15 points	Problem 6 / 16 points

Total / 100

Problem 1 (Basic questions) *This problem is worth 20 points. It consists of four subproblems **1a-1d** for which you **do not** need to motivate your answers.*

In each of these subproblems, the following array is used

$A[1\dots 8] =$

17	3	46	55	14	70	63	28

.

Moreover, in the pseudocode, PUSH, POP, ENQUEUE, DEQUEUE, BUILD-MAX-HEAP, and TREE-INSERT all refer to the procedures explained in class.

1a *(5 points)* Write down the array $B[1\dots 8]$ obtained by running the following pseudocode:

```
1. Let $S$ be an empty stack.
2. for $i = 1, 2, \dots, 8$
3.     PUSH($S, A[i]$)
4. for $i = 1, 2, \dots, 8$
5.     $B[i]$ = POP(S).
```

Please answer by entering the numbers in the empty cells.

The resulting array $B = [1\dots 8] =$

28	63	70	14	55	46	3	17

1b *(5 points)* Write down the array $C[1\dots 8]$ obtained by running the following pseudocode:

```
1. Let $Q$ be an empty queue.
2. for $i = 1, 2, \dots, 8$
3.     ENQUEUE($Q, A[i]$)
4. for $i = 1, 2, \dots, 8$
5.     $C[i]$ = DEQUEUE(Q).
```

Please answer by entering the numbers in the empty cells.

The resulting array $C[1\dots 8] =$

17	3	46	55	14	70	63	28

1c *(5 points)* Write down the array $D[1\dots 8]$ obtained by running the following pseudocode:

```
1. for $i = 1, 2, \dots, 8$
2.     $D[i] = A[i]$
3. BUILD-MAX-HEAP($D$, 8)
```

Please answer by entering the numbers in the empty cells.

The resulting array $D[1\dots 8] =$

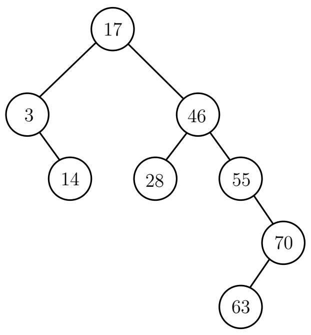
70	55	63	28	14	46	17	3

1d *(5 points)* Depict the binary search tree T obtained by running the following pseudocode:

```
1. Let $T$ be an empty binary search tree.
2. for $i = 1, 2, \ldots, 8$
3.     TREE-INSERT($T, A[i]$)
```

Please answer inside the box.

A drawing of the resulting binary search tree T is as follows:



Problem 2 (Divide-and-conquer analysis) *This problem is worth 15 points. It consists of two subproblems **2a-2b**.*

Consider the following algorithm Unknown that takes as input an integer n:

```
Unknown($n$):
1. if $n < 50$
2.     return
3. Unknown($\lfloor n/4 \rfloor$)
5. Unknown($\lfloor 3n/4 \rfloor$)
6. for $i = 1$ to $\lfloor \sqrt{n} \rfloor$
7.     print "I love algorithms!"
```

2a *(5 points)* Let $T(n)$ be the time it takes to execute Unknown(n). **Give the recurrence relation** for $T(n)$. To simplify notation, you may assume that $n/4, 3n/4$ and $\sqrt{n}$ always evaluate to integers.

In this problem you are asked to write down the recurrence relation for $T(n)$. You do not need to motivate your answer. Recall that you are allowed to refer to material covered in the lectures. Please answer inside the box.

$$T(n) = \begin{cases} \Theta(1) & \text{for } n < 50 \\ T(\frac{n}{4}) + T(\frac{3n}{4}) + O(\sqrt{n}) & \text{for } n \geq 50 \end{cases}$$

2b *(10 points)* **Prove** that $T(n) = O(n)$. You may simplify your calculations by assuming that $n/4, 3n/4$ and $\sqrt{n}$ always evaluate to integers.

In this problem you are asked to give a proof of $T(n) = O(n)$. Recall that you are allowed to refer to material covered in the lectures. Please answer inside the box on this and next page.

We will prove that $T(n) \leq an - c\sqrt{n}$, for some constants a and c (which we will choose). Also, there exists a constant b such that $O(\sqrt{n}) \leq b\sqrt{n}$. We prove our claim using strong induction. The base case is clear by the formula for $n < 50$. Now, suppose that for all $k < n$, $T(k) \leq ak - c\sqrt{k}$. Now, we prove that $T(n) \leq an - c\sqrt{n}$. Note that

$$\begin{aligned} T(n) &= T(\frac{n}{4}) + T(\frac{3n}{4}) + O(\sqrt{n}) \\ &\leq \left(a\frac{n}{4} - c\sqrt{\frac{n}{4}}\right) + \left(a\frac{3n}{4} - c\sqrt{\frac{3n}{4}}\right) + b\sqrt{n} \\ &= an - c\sqrt{n}\left(\frac{1}{2} + \frac{\sqrt{3}}{2}\right) + b\sqrt{n}. \end{aligned}$$

Now, if we choose

$$c \geq \frac{b}{\frac{\sqrt{3}}{2} - \frac{1}{2}},$$

then $T(n) \leq an - c\sqrt{n}$.

Problem 3 (Flow networks) *This problem is worth 17 points. It consists of subproblems **3a-3b** that you answer in the same boxes.*

You have just started your prestigious and important job as the Swiss Cheese Minister. For the the annual festival CHEESY you have ordered the following amount of cheese: 400 portions of Gruyere, 900 portions of Emmentaler, 300 Appenzeller, and 200 portions of Vacherin. Moreover, the attendees can be partitioned into four types according to their cheese preferences stated at registration:

- 200 attendees of type ALL — they are happy with a portion of any of the four cheeses.
- 600 attendees of type AG — they are happy with a portion of either Appenzeller or Gruyere.
- 400 attendees of type EG — they are happy with a portion of either Emmentaler or Gruyere.
- 400 attendees of type VG — they are happy with a portion of either Vacherin or Gruyere.

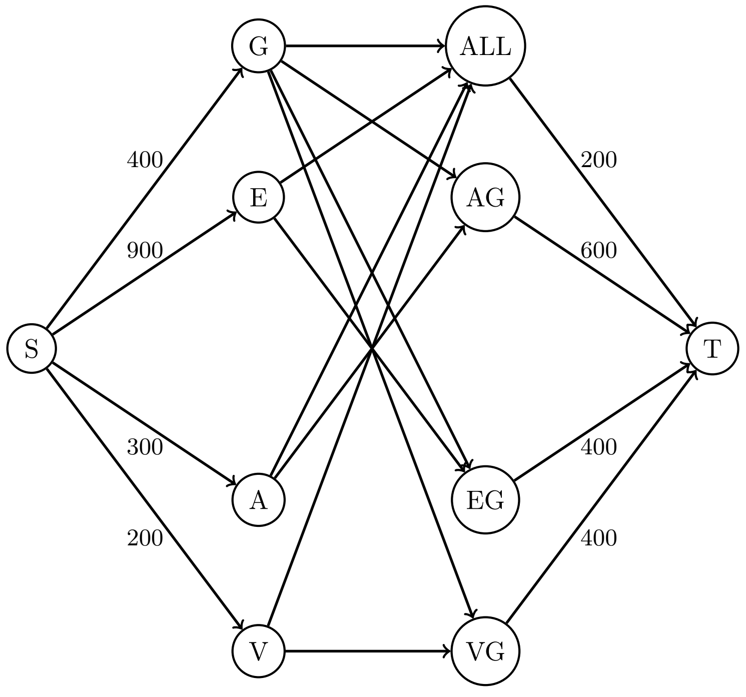
The problem is divided into the following two tasks:

3a *(11 points)* Give a max flow formulation that determines an assignment of cheese portions to attendees so that a maximum number of attendees are happy, i.e., receives a portion of their liking. You should draw a flow network such that an integral flow f corresponds to an assignment where $|f|$ attendees receive a cheese portion that respect their preference.

3b *(6 points)* Is it possible to satisfy all CHEESY attendees? If yes, then give a flow in your flow network that corresponds to an assignment where each attendee gets a portion of cheese of their liking. If no, then find a cut in your flow network whose capacity is less than the total number of attendees.

In this problem, you are first asked to formulate a flow network whose solutions correspond to assignment of cheese portions to attendees. Then you are asked to answer whether there is an assignment that satisfy all attendees by either giving a flow or a cut. Recall that you are allowed to refer to material covered in the lectures. Please answer inside the box on this and next page.

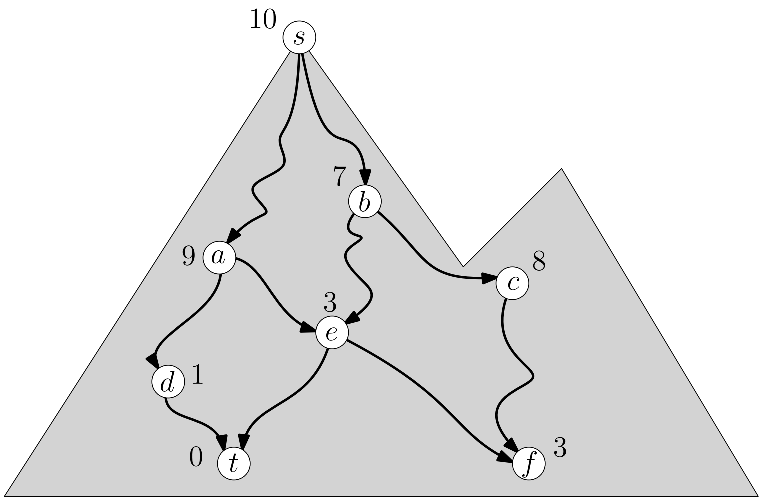
1. For the part (a), the network is described below. All the edges in the middle layer are of value ∞. The layer to the right of S (source) represents the different kinds of cheese, and the layer to the left of T (sink) represents the different kinds of attendees. Finding a maximum S-T flow in this network will help us satisfy the maximum number of attendees. The flow on the edges of capacity ∞ would give us an assignment of cheeses to the different kinds of attendees.



2. It is not possible to satisfy all the attendees. This is seen by considering the cut given by $\{S, E, ALL, EG\}$. The capacity of this cut is 1500 but the number of attendees is 1600.

Problem 4 (Maximizing the experience) *This problem is worth 17 points.*

Yippee, you are finally standing on the top of the mountain! As the climb was rather tough you want to carefully plan your skiing so as to maximize your experience. For your help you have a map with all the excellent viewpoints weighted by their beauty: the higher the number the better the view! As an example, consider the following map.



Your starting position is labeled s and you must end your descent at t where you parked your car. This means that, in the depicted example, you cannot visit the viewpoints c or f although they are very nice. Instead you have three available options for descending from s to t:

- $s \to b \to e \to t$ with a total weight of $10 + 7 + 3 + 0 = 20$;
- $s \to a \to e \to t$ with a total weight of $10 + 9 + 3 + 0 = 22$;
- $s \to a \to d \to t$ with a total weight of $10 + 9 + 1 + 0 = 20$;

and so the second option is best with a total weight of 22.

More formally, the definition of the BEAUTIFULSKIING problem is follows:

Input: A directed acyclic graph $G = (V, E)$ with vertex-weights $w : V \to \mathbb{R}$, a source $s \in V$ and a sink $t \in V$

Output: The maximum weight of a path from s to t, where the weight of a path is the sum of the weights of its vertices.

So 22 is the correct output when given the depicted example as input. Your task is to **design and analyze** an algorithm for the BEAUTIFULSKIING problem that runs in time $O(|V| + |E|)$. Slower algorithms that run in time $\Theta(|V| \cdot |E|)$ will receive partial credits up to 10 points.

In this problem you are asked to design and analyze an algorithm for the BEAUTIFULSKIING *problem that runs in linear time $\Theta(|V| + |E|)$. Recall that you are allowed to refer to material covered in the lectures. Please answer inside the box on this and the following two pages.*

As a first step we will transform our node weighted graph into an edge weighted graph such that the cost of every path is preserved. To do that we define $\tilde{w} : E \to R$ such that $\tilde{w}(s, v) = w(s) + w(v)$ for all edges of the form (s, v) (outgoing from the source edges) and $\tilde{w}(u, v) = w(v)$ for every edge (u, v) where vertex u is different from the source.

1. $O(|V| + |E|)$ **solution**

   We first describe the recursive formula. To that end let $m(u)$ denote the maximum weight of a path which starts at the source and ends at node u. Then $m(s) = 0$ and

$$m(u) = \max_{(v,u)\in E}(m(v) + \tilde{w}(v, u)), u \neq s$$

   In order to calculate efficiently (only using as many comparisons as the number of ingoing edges) $m(u)$ we need to have already calculated all the $m(v)$ for vertexes with outgoing edges to u.

   To that end we continue by topologically sorting the vertexes. By doing that we associate each vertex with a number from 1 to n which describes the vertex position in the topological order. To make the description of the algorithm simple let $p : \{1, n\} :\to V$ be the function which describes this position. Now we are ready, using a bottom up approach, to calculate $m(t)$.

   By noting that topological sorting can be performed in linear time and that the inner for loop of the dynamic approach solution uses at most $|E|$ comparisons we get an overall linear time complexity.

Algorithm 1 DYNAMIC PROGRAMMING (DP) APPROACH

```
1. $m(s) = 0$
2. **for** $i = 2$ to $n$
3.     let $u = p(i)$
4.     $m(u) = -\infty$
5.     **for** all $v$ such that $(v, u) \in E$
6.         $m(u) \leftarrow \max\{m(u), m(v) + \tilde{w}(u, v)\}$
7. Output $m(t)$
```

2. $O(|V| \cdot |E|)$ **solution**

   Since we are trying to solve a weight maximization problem we first define a new edge weight function $w'(u, v) = -\tilde{w}(u, v)$ for all edges $(u, v) \in E$. Since the graph does not contain negative cycles (and more generally cycles) we run Bellman-Ford algorithm from the source node s using the new weight function w'. We finally output $-d(t)$ where $d(t)$ is the distance calculated by the Bellman-Ford algorithm. The overall complexity of the algorithm is linear (due to the initial change of the weight function) plus the complexity of the Bellman Ford algorithm. Thus the overall complexity (which is dominated by the complexity of the Bellman-Ford algorithm) is $O(|V| \cdot |E|)$.

Problem 5 (Probabilistic Analysis) *This problem is worth 15 points.*

Recall the hiring problem seen in class: we have n basketball players arriving in a uniformly at random order and we hire a player if she is taller than those that arrived beforehand. Surprisingly, we proved that in expectation, over the random arrival order of the players, the number of hires we make is only $\Theta(\log n)$. However, while $\log(n)$ is small, it is a growing function of n.

In this problem we will slightly change the hiring process so that the number of hires is *independent* of the number n. To this end, consider the following hiring process:

- The n players arrive in a uniformly at random order.
- None of the first $n/10$ players are hired.
- Each of the remaining $9n/10$ players is hired if taller than all players that arrived beforehand.

Hence, the only change with respect to the hiring process analyzed in class is that we never hire any of the first 10% of the players. Note that, this may result in us not hiring any of the players. This happens if the tallest player happens to arrive among the first $n/10$ players. However, the advantage with the updated hiring process is that we only hire a few players (independent of n) in expectation. Specifically, your task is to **prove that we make at most 10 hires in expectation over the random arrival of players**.

In this problem you are asked to prove that the updated hiring process makes at most 10 hires in expectation over the random arrival of the players (no matter n). Recall that you are allowed to refer to material covered in the lectures. Please answer inside the box on this and next page.

Solution 1:
Let $X_i = \chi[i\text{th player to arrive is hired}]$ be the indicator random variable for the event that the ith player to arrive is hired. Then $X = X_1 + X_2 + \cdots + X_n$ is the random variable for the number of players hired. We have

$$\begin{aligned}
E[X] &= E\left[\sum_{i=1}^{n} X_i\right] \\
&= \sum_{i=1}^{n} E[X_i] \text{ by linearity of expectation} \\
&= \sum_{i=1}^{n} \Pr[i\text{th arriving player is hired}] \\
&= \sum_{i=n/10+1}^{n} \Pr[i\text{th arriving player is hired}] \text{ since first } n/10 \text{ are ignored} \\
&= \sum_{i=n/10+1}^{n} \frac{1}{i} \\
&\leq \int_{i=n/10}^{n} \frac{1}{i} \\
&= \log n - \log(n/10) \\
&= \log 10 \leq 10.
\end{aligned}$$

Solution 2:
Let $X_i = \chi[i\text{th tallest player is hired}]$ be the indicator random variable for the event that the ith tallest player is hired. Then $X = X_1 + X_2 + \cdots + X_n$ is the random variable for the number of players that are hired. We have

$$\begin{aligned}
E[X] &= E\left[\sum_{i=1}^{n} X_i\right] \\
&= \sum_{i=1}^{n} E[X_i] \text{ by linearity of expectation} \\
&= \sum_{i=1}^{n} \Pr[i\text{th tallest player is hired}] \\
&= \sum_{i=1}^{n} \Pr[i\text{th player is in last } 9/10 \text{ to arrive, and taller than all previous players}] \\
&= \sum_{i=1}^{n} \Pr[i\text{th is in last } 9/10] \Pr[i\text{th taller than all previous} \mid \text{in last } 9/10 \text{ to arrive}] \\
&\le \sum_{i=1}^{n} \left(\frac{9}{10}\right)\left(\frac{9}{10}\right)^{i-1} \cdot \frac{1}{i} \\
&\le \frac{9/10(1-9/10)^n}{1-9/10} \text{ because } i \ge 1 \text{, and by geometric series formula} \\
&\le \frac{9/10}{1/10} \\
&\le 10.
\end{aligned}$$

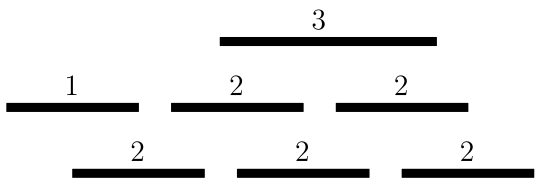
The first inequality comes from the following reasoning. First, the probability that any given player arrives in the last $9/10$ fraction of all the players happens with probability $9/10$, and then for each of the $i-1$ players that are taller than the ith, it must be the case that they also arrive in the last $9/10$ fraction (or else they cannot be arriving after the ith player). Finally, there are i possible positions for the ith player to be arranged in, compared to the $i-1$ tallest players (first, second, ..., last), and in only 1 of them the ith player is first. This gives $(9/10)(9/10)^{i-1}(1/i)$. Hence, no more than 10 players are hired in expectation.

Problem 6 (Cardinality constrained disjoint intervals) *This problem is worth 16 points.*

Design and analyze an efficient algorithm for the following problem:

> **Input:** An integer $k \geq 0$, n intervals $I_1 = [a_1, b_1], I_2 = [a_2, b_2], \ldots, I_n = [a_n, b_n]$, and a value $v(I_i) \geq 0$ for each interval $i = 1, 2, \ldots, n$.
>
> **Output:** The maximum value $\sum_{I_i \in S} v(I_i)$ achievable by a subset $S \subseteq \{I_1, I_2, \ldots, I_n\}$ of intervals satisfying that
>
> - S contains up to k intervals: $|S| \leq k$; and
> - the intervals in S are mutually disjoint: for every two intervals $I_i, I_j \in S$ with $i \neq j$ we have $I_i \cap I_j = \emptyset$.

The running time of your algorithm should be polynomial in n and k. For simplicity, you may assume that all endpoints (the a_i's and b_i's) are unique so that there is no ambiguity when two intervals are overlapping.

> **Example** An example of seven intervals can be depicted as follows:
>
> 
>
>
> The value of an interval is given by the depicted number above it, so there is one interval of weight 1, five intervals of weight 2, and one interval of weight 3. For $k = 3$, the maximum value equals 6 which is achieved by the the three bottom intervals. For $k = 2$, the maximum value equals 5 which is achieved by the bottom left interval and the topmost interval.

In this problem you are asked to design and analyze an algorithm for the above defined problem whose running time is polynomial in n and k. Recall that you are allowed to refer to material covered in the lectures. Please answer inside the box on this and the following two pages.

We derive a dynamic programming solution using the following idea: Suppose we *know* the right-most interval in the solution. Then the remaining intervals in the solution must be the optimum solutions that we can get using $k-1$ intervals from the set of intervals that are strictly to the left of the picked solution.
Thus we can derive a recursive formulation as the following: First, we assume all the intervals are sorted according to b_i values (i.e., the right end-points). Otherwise, we sort them and re-label them in this sorted order.
Let $d(i, j)$ be the maximum value obtained for a set of at most j disjoint intervals out of the first i intervals. The goal is to find $d(n, k)$. For the i-th interval, there are two choices: either we add the i-th interval or we do not add it. If we add it, we can select the best solution with at most $j-1$ disjoint intervals out of the interval that do not overlap with the i-th interval. If we do not add it, we can select the best solution with at most j disjoint intervals $\{I_1, \ldots, I_{i-1}\}$. Let $f(i)$ be the maximum index such that $b_{f(i)} < a_i$, and note that all intervals from $I_1, \ldots, I_{f(i)}$ are disjoint from I_i. With this notation, we get the following recursive formula:

$$d(i,j) = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0 \\ \max\{v(I_i) + d(f(i), j-1), d(i-1, j)\} & \text{otherwise} \end{cases}$$

We can use the bottom-up or top-down with memoization to solve this. In either case, there are at most $n \cdot k$ values to find, and the running time to find a single $d(i, j)$ value is dominated by the the time to compute $f(i)$ which is $O(\log n)$ when the intervals are sorted. Note that sorting the intervals take $O(n \log n)$ time so the total running time would be $O(n \cdot k \cdot \log n)$.
Note that we can also precompute all $f(i)$ values in time $O(n)$ once the intervals are sorted, and in this case, the running time is $\underbrace{O(n \log n)}_{\text{for sorting}} + O(n \cdot k)$.