CSCI 3104: Algorithms
Spring 2022
Recitation #7 - Midterm Review

SOLUTIONS

Midterm TA reviews. Please take take a few minutes to fill out TA FCQs! They're very helpful for us TAs. You can find the link in your email under the subject line "Computer Science Midterm TA FCQ's."

Problem 1

Prove the following via induction (strong or weak).

a. $\sum_{k=1}^{n} q^{k-1} = \frac{q^n-1}{q-1}$, for all $n \geq 1$, $q \neq 1$.

BC. For $n = 1$, $\sum_{k=1}^{1} q^{k-1} = 1 = \frac{q^1-1}{q-1}$, and the statement holds.

IH. Assume the statement holds for some $n \geq 1$.

IS. We show the statement holds for $n + 1$, as well:

$$\begin{aligned}
\sum_{k=1}^{n+1} q^{k-1} &= \sum_{k=1}^{n} q^{k-1} + q^n \\
&= \frac{q^n - 1}{q - 1} + q^n && \text{by IH.} \\
&= \frac{q^n - 1 + q^{n+1} - q^n}{q - 1} \\
&= \frac{q^{n+1} - 1}{q - 1}.
\end{aligned}$$

b. $\sum_{k=1}^{n}(2k - 1) = n^2$ for all $n \geq 1$.

BC. For $n = 1$, $\sum_{k=1}^{1}(2k - 1) = 1 = 1^2$, and the statement holds.

IH. Assume the statement holds for some $n \geq 1$.

IS. We show the statement holds for $n + 1$, as well:

$$\begin{aligned}
\sum_{k=1}^{n+1}(2k - 1) &= \sum_{k=1}^{n}(2k - 1) + (2(n + 1) - 1) \\
&= n^2 + 2n + 1 && \text{by IH.} \\
&= (n + 1)^2.
\end{aligned}$$

c. (Bonus.) Any integer $n \geq 2$ is a product of prime numbers.

BC. $n = 2$ is a prime number and thus the product of prime numbers.

IH. Assume the statement holds for all $2 \leq k \leq n$ for some $n \geq 2$.

IS. We show the statement holds for $n+1$, as well. If $n+1$ is prime, then it is trivially the product of one prime number. Otherwise, $n+1 = ab$ for some $1 < a < n$, $1 < b < n$. By the inductive hypothesis, a and b can both be written as the products of primes,

$$a = a_1 a_2 \dots a_n$$
$$b = b_1 b_2 \cdots_m$$

and we can write $n+1$ as a product of prime numbers: $n+1 = \prod_{i=1}^{n} a_i \prod_{j=1}^{m} b_j$.

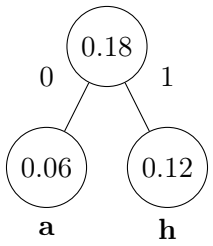
Problem 2

Construct a Huffman Code for the phrase "she sells sea shells". The frequencies for each letter are as follows (we ignore spaces):

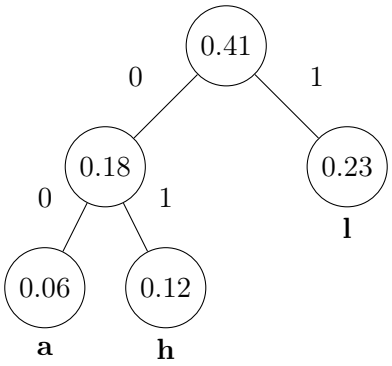
letter	a	e	h	l	s
number of ocurrences	1	4	2	4	6
frequency	0.06	0.24	0.12	0.23	0.35

(*Note:* "e" and "l" actually have the same frequency, but we round one up and one down to give each letter a distinct frequency in this example.)

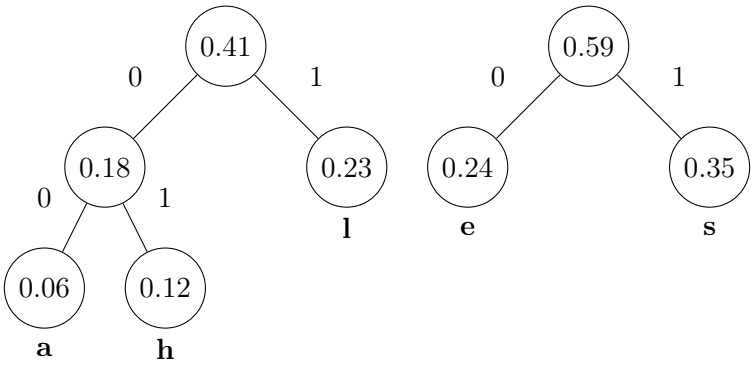
We order the letters in increasing freqency: {(a, 0.06), (h, 0.12), (l, 0.23), (e, 0.24), (s, 0.35)}. We take the two lowest-frequency letters and "combine" them. Note that it does not technically matter which goes on which side, but convention is that the smaller one is labelled "0" and goes on the left side:



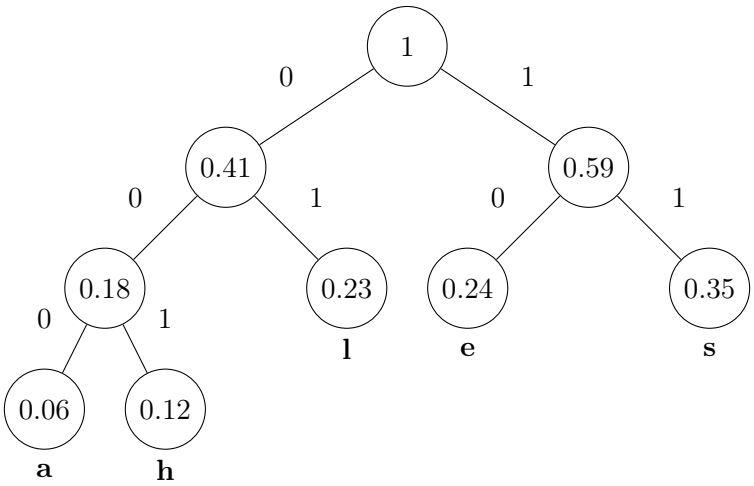
We then have the list {(ah, 0.18), (l, 0.23), (e, 0.24), (s, 0.35)}, and again combine the smallest two:



We now have {(e, 0.24), (s, 0.35), (ahl, 0.41)}. Note that the subtree we have been working with has smaller total frequency than other elements of the list and its place in the list has moved. We combine "e" and "s":



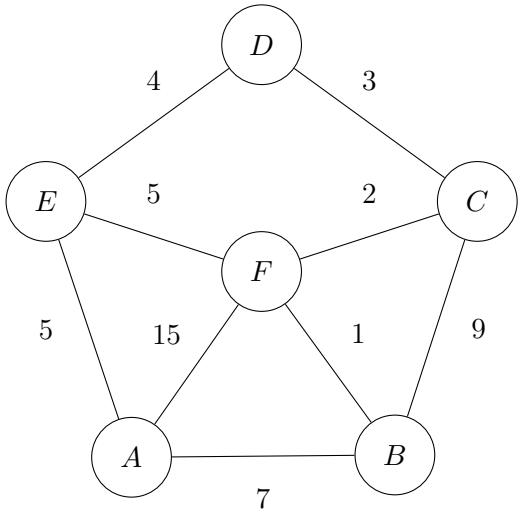
Finally, we have only the vertices {(ahl, 0.41),(es, 0.59)} and combine them to form the root:



You can write the phrase "she sells sea shells" as "11 001 10 11 10 011 011 11 11 10 000 11 001 10 011 011 11."

Problem 3

Run Dijkstra's algorithm on the following graph to find a SSSP tree, starting from vertex A.



- We initialize the distances and previous vertices.

vertex	A	B	C	D	E	F
current distance	0	∞	∞	∞	∞	∞
current previous	–	–	–	–	–	–

  We also initialize our priority queue, $Q = \{(A, 0)\}$.

- We poll $(A, 0)$ from the queue, and update A's neighbors, adding them to the priority queue.

vertex	A	B	C	D	E	F
current distance	0	7	∞	∞	5	15
current previous	–	A	–	–	A	A

  We update our priority queue with the newly-visited vertices in order of increasing distance from A: $Q = \{(E, 5), (B, 7), (F, 15)\}$.

- We poll $(E, 5)$ from the queue, and update E's neighbors.

vertex	A	B	C	D	E	F
current distance	0	7	∞	9	5	10
current previous	–	A	–	E	A	E

  We add D to our priority queue and update F: $Q = \{(B, 7), (D, 9), (F, 10)\}$.

- We poll $(B, 7)$ from the queue, and update B's neighbors.

vertex	A	B	C	D	E	F
current distance	0	7	16	9	5	8
current previous	–	A	B	E	A	B

  We add C to our priority queue: $Q = \{(F, 8), (D, 9), (C, 16)\}$.

- We poll $(F, 8)$ from the queue, and update F's neighbors.

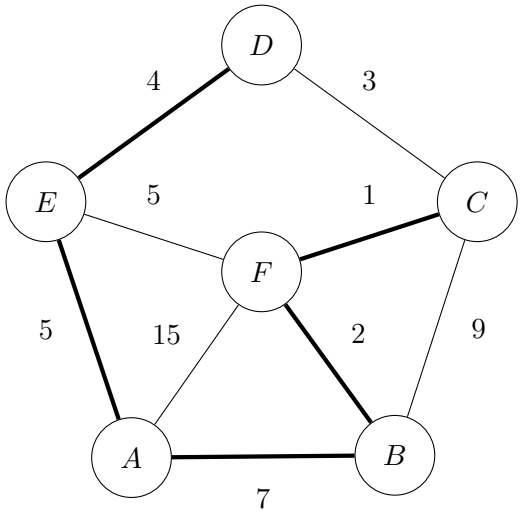
vertex	A	B	C	D	E	F
current distance	0	7	10	9	5	8
current previous	–	A	F	E	A	B

  We now have $Q = \{(D, 9), (C, 16)\}$.

- We poll $(D, 9)$ from the queue, and find no updates to make on D's neighbors. We now have $Q = \{(C, 16)\}$.

- We poll the last item C from the queue, and find no updates to make on C's neighbors, giving us a completed SSSP from A.

Our completed tree is



Problem 4

You've decided to go on a backpacking trip in the beautiful Rocky Mountains. Your itinerary is pretty ambitious, and you can't waste time stopping to dig through your bag to get things out. Luckily, you can use your algorithms expertise to pack your bag as efficiently as possible, to minimize the amount of time you spend rooting around for something at the bottom.

To be concrete, you have n items to pack. Item i has length $\ell_i \geq 0$ and probability p_i of you wanting to access it. Assuming you can only pack items on top of each other, and that the time to access the i^{th} item down your pack is directly proportional to the combined lengths of the top i items, determine which order to place the items such that the *expected* access time T is minimized. In other words, minimize

$$T = \sum_{i=1}^{n} p_i L_i,$$

where L_i is the combined length of items above and including i in the pack.

a. Devise an algorithm to order your gear in a way that minimizes the total expected access time.

> We use the greedy by largest p_i/ℓ_i ratio algorithm, with items deeper in the bag having smaller ratio.

b. Prove your algorithm is correct.

> We prove this via a greedy exchange argument. Let $\mathcal{A}$ be the ordering produced by our greedy algorithm, and $\mathcal{O}$ be an alternative ordering. Without loss of generality, we may assume items are numbered according to their ordering in $\mathcal{O}$, with item 1 being on the top in $\mathcal{O}$ and item n at the bottom.
>
> If $\mathcal{O} \neq \mathcal{A}$, then there exists some *inversion* from $\mathcal{A}$'s ordering, where $j < i$ and $p_j/\ell_j \leq p_i/\ell_i$. More specifically, there must exist two adjacent items in our pack which are inverted, $p_i/\ell_i \leq p_{i+1}/\ell_{i+1}$. We can exchange i and $i+1$ without affecting the access cost on items above or below i and $i+1$, since the total length to access them remains unchanged.
>
> We can rewrite our inversion as $p_i\ell_{i+1} \geq p_{i+1}\ell_i$. We write the cost for access to i and $i+1$ in $\mathcal{O}$:
>
> $$\begin{aligned} p_i L_i + p_{i+1} L_{i+1} &= (L_{i-1} + \ell_i)p_{i+1} + (L_{i-1} + \ell_i + \ell_{i+1})p_i \\ &= L_{i-1}p_{i+1} + \ell_i p_{i+1} + L_{i-1}p_i + \ell_i p_i + \ell_{i+1}p_i \\ &\geq L_{i-1}p_{i+1} + \ell_i p_{i+1} + L_{i-1}p_i + \ell_i p_i + \ell_i p_{i+1} \\ &= L_i p_{i+1} + (L_i + \ell_{i+1})p_i, \end{aligned}$$
>
> the total expected access cost for i and $i+1$ if we exchange their places in $\mathcal{O}$. Thus, by exchanging i and $i+1$ in $\mathcal{O}$, we can only improve our solution and eventually convert $\mathcal{O}$ into our solution, $\mathcal{A}$.