Quiz Template

Length: 15 min

Your name and **CCID**:

CMPUT 304
Department of Computing Science
University of Alberta

Quiz Structure. You will be given 2 questions and 15-20 minutes to fill the quiz.

Question Template. Here are all possible types of questions you *might* be asked in the actual quiz.

1. What is the result of running Huffman's algorithm on the following alphabet with characters $c_1, ..., c_X$ and frequencies given by

c_1	c_2	...	c_X
...	...	...	...

2. Is the following $\mathcal{M} = (U, \mathcal{I})$, defined by $U = ...$ and $\mathcal{I} = ...$ a matroid or not? Explain.
3. Show how the dynamic programming algorithm computes the edit distance between "...." and "..." where the costs are specified as...
4. Show how to find the optimal BST for the X keys $a_1 < a_2 < ... < a_X$ where the number of requests for each key is: $r_1 = ... \; r_X = ...$
5. Show how to compute all pairs shortest path distances on the following graph.
6. Show how to find the max-weight independent set on the following tree where the weight of each node is given in the diagram.

It is extremely unlikely that you'll be given a question that involves more than 8 items.

Guidelines. The purpose of this quiz isn't for you to actually solve the mini-problems on 4/5/6/8 elements. We do not care about specific values or specific solutions. The purpose is for you to illustrate us that you know how the different algorithms work. So do NOT spew onto the page: "here is the table we filled." Rather, show us you know how the algorithm operates; so key phrases are "we first fill the left-most / bottom-most cells in the array as they correspond to the base case" or "we now iterate of subarrays based on length, from those of length 2 to those of length 5, and fill each entry based on the cells below it and left of it" etc.

If you feel that you are stressed for time, it is ok to start skipping intermediate stages of the algorithm as long as you *provide us with an explanation of what you were doing.* "After we filled all cells corresponding to subsequences of length 2, we move to those of length 3, then length 4 and then length 5 and fill those in a similar fashion. Here is the result of the table after all iterations have been completed. You can see for example that we filled cell $[1, 4]$ using the best of the following options..."

Very few, if any, points will be deducted to arithmetic errors and/or stopping the algorithm prematurely. Just write down it down. "I am stressed for time, so rather then comparing the edit distance of the full two 7-characters long strings, I'm halting after the first 5 characters in each. Now, let me show how to find the instruction sequence that convert this first 5-character string ... to the second 5-character string...".

Preparation. After you go over your notes and make sure you understand how all the algorithms work, rather than testing yourselves on the examples provided below, you *should* devise your own examples over 6-8 items and solve yours.

Examples. Below we give a few specific examples of problems you may see in the quiz. Note that *not* all of them are solved.

Problem 1. What is the result of running Huffman's algorithm on the following alphabet with characters $c_1, c_2, c_3, c_4, c_5, c_6, c_7$ and frequencies given by

c_1	c_2	c_3	c_4	c_5	c_6	c_7
0.01	0.1	0.18	0.2	0.19	0.16	0.16

?

Problem 2. Is the following $\mathcal{M} = (U, \mathcal{I})$, defined by $U = \{1, 2, 3, 4, 5\}$ and

$$\mathcal{I} = \{\emptyset, \{1\}, \{2\}, \{3\}, \{4\}, \{5\}, \{1, 2\}, \{1, 4\}, \{1, 5\}, \{2, 3\}, \{2, 4\}, \{3, 4\}, \{3, 5\}, \{4, 5\}\}$$

a matroid or not? Explain.

Problem 3. Show how the dynamic programming algorithm computes the edit distance between "*accb*" and "*bccac*" where the costs are specified as:

	to a	to b	to c
from a	0	5	5
from b	6	0	7
from c	8	6	0

with adding costs of

a	b	c
7	8	7

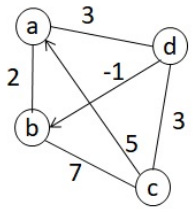
and omitting costs of

a	b	c
6	5	5

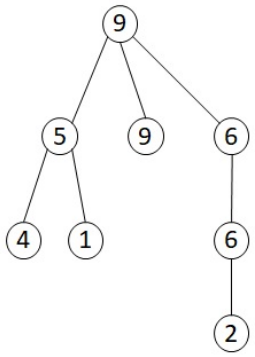
Problem 4. Show how the dynamic programming algorithm computes the edit distance between "*simplest*" and "*example*" where the cost of converting one character to another is always 1 and the cost for adding or removing a character is always 2.

Problem 5. Show how to find the optimal BST for the 6 keys $a_1 < a_2 < a_3 < a_4 < a_5 < a_6$ where the number of requests for each key is: $r_1 = 10$, $r_2 = 9$, $r_3 = 1$, $r_4 = 10$, $r_5 = 2$, $r_6 = 30$.

Problem 6. Show how to compute all pairs shortest path distances on the following graph.
(If an edge is undirected that means it has the same weight from u to v as from v to u.)



Problem 7. Show how to find the max-weight independent set on the following tree where the weight of each node is given in the diagram.



Answer. [**To Problem 2.**]

The given $\mathcal{M}$ is a matroid. We verify that indeed $\mathcal{I}$ satisfy the two properties.

Downward closure: Take any $A \in \mathcal{I}$. If $|A| = 0$ then it is the empty set, which is in $\mathcal{I}$. If $|A| = 1$ then any subset of it has size 0 or 1, and so we know that subset is in $\mathcal{I}$. If $|A| = 2$ then any subset of A has size 0,1 or 2 — the empty set belongs to $\mathcal{I}$, all singletons belong to $\mathcal{I}$ and clearly the subset of A of size 2 is A itself so it belongs to $\mathcal{I}$.

Augmentation property: Take any $A, B \in \mathcal{I}$ such that $|A| > |B|$. If $|B| = 0$ then $B = \emptyset$ and it is simple to see the augmentation property holds, especially since all possible singletons are in $\mathcal{I}$. If $|B| = 1$ then $|A| = 2$ (the cardinality of the largest sets in $\mathcal{I}$ is 2), and we need to check that on a case by case basis. We do it in the following table: one row per each possible A and one column for each possible B. Inside the table we specify an element in $A \setminus B$ with whom we can augment B.

	$\{1\}$	$\{2\}$	$\{3\}$	$\{4\}$	$\{5\}$
$\{1,2\}$	2	1	2	2	1
$\{1,4\}$	4	4	4	1	1
$\{1,5\}$	5	1	5	1	1
$\{2,3\}$	2	3	2	3	3
$\{2,4\}$	2	4	4	2	4
$\{3,4\}$	4	3	4	3	3
$\{3,5\}$	5	3	5	5	3
$\{4,5\}$	5	4	4	5	4

Note how when $B \subset A$, for example: when $A = \{2,3\}$ and $B = \{3\}$, we always choose the one element in $A \setminus B$ to augment B with (in the example: it is 2).

Note how in some cells there are multiple possible options. For example, for $A = \{4,5\}$ and $B = \{3\}$ we could either augment $\{3\}$ with 4 or with 5 (as both $\{3,4\}$ and $\{3,5\}$ belong to $\mathcal{I}$). In other cells, we only has a single option. For example, in the case of $A = \{1,2\}$ and $B = \{5\}$ we can only pick 1 to augment $\{5\}$ with, as $\{2,5\} \notin \mathcal{I}$. Hence.

Answer. [**To Problem 3.**] Show how the dynamic programming algorithm computes the edit distance between "*acccb*" and "*bccac*" where the costs are specified as:

	to a	to b	to c
from a	0	5	5
from b	6	0	7
from c	8	6	0

with adding costs of

a	b	c
7	8	7

and omitting costs of

a	b	c
6	5	5

The dynamic programming algorithm creates an array with a cell for every substring of the first string and any substring of the original second string. In our case, as the strings are of size 4 and 5, the array is 5×6 (as the empty string is a valid subarray).

The first cells we fill are those that correspond to either the first string begin empty (in which case the cost is just adding the characters in the second string one by one), or the second string being empty (in which case the cost is just removing the characters of the first string one by one).

convert to:	ϵ	"*b*"	"*bc*"	"*bcc*"	"*bcca*"	"*bccac*"
from ϵ:	0	8	15	22	29	36
from "*a*":	6					
from "*ac*":	11					
from "*acc*":	16					
from "*accb*":	21					

We now fill the array one row after another. In each time, we only examine 3 cells: $A[i,j]$ is the minimum of the three options: cost of replacing x_i with y_j plus $A[i-1,j-1]$; cost of omitting x_i + $A[i-1,j]$; cost of adding y_j plus $A[i,j-1]$. Thus $A[1,1] = 5+0$ as it is the cheapest option. We fill in the rest of the first row in a similar fashion.

convert to:	ϵ	"*b*"	"*bc*"	"*bcc*"	"*bcca*"	"*bccac*"
from ϵ:	0	8	15	22	29	36
from "*a*":	6	5	12	19	$\min\{0+22, 6+29, 7+19\} = 22$	$\min\{5+29, 7+36, 5+22\} = 27$
from "*ac*":	11					
from "*acc*":	16					
from "*accb*":	21					

We now fill in the second row, again, in a similar fashion. We now consider the cost of replacing c with the last character in the suitable target substring, omitting c, or adding the last character in the suitable target substring.

convert to:	ϵ	"*b*"	"*bc*"	"*bcc*"	"*bcca*"	"*bccac*"
from ϵ:	0	8	15	22	29	36
from "*a*":	6	5	12	19	22	27
from "*ac*":	11	10	5	12	19	22
from "*acc*":	16					
from "*accb*":	21					

The next two lines are also filled in a similar fashion

convert to:	ϵ	"*b*"	"*bc*"	"*bcc*"	"*bcca*"	"*bccac*"
from ϵ:	0	8	15	22	29	36
from "*a*":	6	5	12	19	22	27
from "*ac*":	11	10	5	12	19	22
from "*acc*":	16	15	10	5	12	19
from "*accb*":	21	16	15	10	11	18

Now that the array has been filled, we can retrace the way we filled it, based on the formula. We start with the cell $[4, 5]$, whose value is 18. It is $18 = 7 + 11$ because that is the cost of adding c and $A[4, 4]$; so our instructions conclude with "add c".

We recurse on $A[4, 4] = 11$. This values is set because converting $b \to a$ costs 6 plus $A[3, 3]$. We thus know that our last instruction is "convert b to c and recurse on $A[3, 3]$.

Note how $A[3, 3] = A[2, 2] = A[1, 1] = 5$, as in the next two characters both are c in both substrings. On $A[1, 1] = 5$ the cost is set by replacing a with b. We then get to $A[0, 0]$ where we halt.

This gives the instructions: "convert a to b", "convert b to a", "add c".

Answer. [**To Problem 5.**]

Show how to find the optimal BST for the 6 keys $a_1 < a_2 < a_3 < a_4 < a_5 < a_6$ where the number of requests for each key is: $r_1 = 10$, $r_2 = 9$, $r_3 = 1$, $r_4 = 10$, $r_5 = 2$, $r_6 = 30$.

We are designating a 6×6-array A where $A[p, q]$ holds the cost of the optimal BST on the sequence of keys $a_p < ... < a_1$.

By definition, $A[p, q]$ for any p, q such that $q < p$ is the cost on an empty sequence, hence for these values $A[p, q] = 0$. These are the entries we fill first in our array.

upto:	a_1	a_2	a_3	a_4	a_5	a_6
starting at a_1						
starting at a_2	0					
starting at a_3	0	0				
starting at a_4	0	0	0			
starting at a_5	0	0	0	0		
starting at a_6	0	0	0	0	0	

We now turn to fill the diagonal entries of A. Those stand for a problem with a single key. Here the solution is simple, we put the only key as the root, costing us 1 per request. Thus the cost of each $A[p, p]$ is r_p.

upto:	a_1	a_2	a_3	a_4	a_5	a_6
starting at a_1	10					
starting at a_2	0	9				
starting at a_3	0	0	1			
starting at a_4	0	0	0	10		
starting at a_5	0	0	0	0	2	
starting at a_6	0	0	0	0	0	30

Now, we fill entries of the form $A[p, p + 1]$. We fill it using the formula that tries both elements as the root, and takes the minimum over the cost of the possible subtree. In this case, where there are only two keys, one subtree is empty (and costs 0) and the other costs is already filled and is written on the corresponding diagonal entry. This way, $A[4, 5]$ for example, is going the be the best of $r_4 + r_5 + A[5, 5]$(the root is a_4) and $r_4 + r_5 + A[4, 4]$(the root is a_5). As the smallest of the two is $A[5, 5]$, we choose to put a_4 as the root, and the chosen entry value is $10 + 2 + 2 = 14$.

upto:	a_1	a_2	a_3	a_4	a_5	a_6
starting at a_1	10	28				
starting at a_2	0	9	11			
starting at a_3	0	0	1	11		
starting at a_4	0	0	0	10	14	
starting at a_5	0	0	0	0	2	34
starting at a_6	0	0	0	0	0	30

Now, we fill entries of the form $A[p, p + 2]$. We fill it using the formula that tries all three elements as the root, and adds the sum of their requests to the min-cost way of splitting them: no keys on the left subtree and 2 keys in the right subtree; or one key in the left subtree and one key in the right subtree; or 2 keys in the left subtree and no keys in the right subtree. Thus, $A[2, 4]$ for example is $r_2 + r_3 + r_4 + \min\{0 + A[3, 4], A[2, 2] + A[4, 4], A[2, 3] + 0\} = 20 + \min\{11, 10 + 9, 11\} = 31$.

upto:	a_1	a_2	a_3	a_4	a_5	a_6
starting at a_1	10	28	31			
starting at a_2	0	9	11	31		
starting at a_3	0	0	1	11	16	
starting at a_4	0	0	0	10	14	56
starting at a_5	0	0	0	0	2	34
starting at a_6	0	0	0	0	0	30

Repeating this for cells $A[p, p+3]$ and the $A[p, p+4]$ we get

upto:	a_1	a_2	a_3	a_4	a_5	a_6
starting at a_1	10	28	31	51	58	
starting at a_2	0	9	11	31	35	87
starting at a_3	0	0	1	11	16	59
starting at a_4	0	0	0	10	14	56
starting at a_5	0	0	0	0	2	34
starting at a_6	0	0	0	0	0	30

Finally we are left with the last cell $A[1, 6]$ which is filled by:

$$r_1+...+r_6+\min\{0+A[2,6],\ A[1,1]+A[3,6],\ A[1,2]+A[4,6],\ A[1,3]+A[5,6],\ A[1,4]+A[6,6],\ A[1,5]+0\}$$

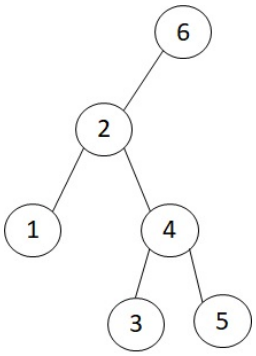
i.e., by

$$62 + \min\{87, 10+59, 28+56, 31+34, 51+30, 58\} = 62 + 58 = 120$$

upto:	a_1	a_2	a_3	a_4	a_5	a_6
starting at a_1	10	28	31	51	58	120
starting at a_2	0	9	11	31	35	87
starting at a_3	0	0	1	11	16	59
starting at a_4	0	0	0	10	14	56
starting at a_5	0	0	0	0	2	34
starting at a_6	0	0	0	0	0	30

Now that the table has been filled, we recurse over the filled array and ask in each case why we filled that particular value. As we just saw, $A[1, 6]$ was obtained by $62 + A[1, 5]$ so a_6 is the overall key, with an empty right subtree and a left subtree whose root is how we filled $A[1, 5]$.

$A[1, 5]$ was filled by $32 + A[1, 1] + A[3, 5] = 32 + 10 + 16$. That means that its root is a_2. Clearly its left subtree holds a single element a_1 and its right subtree holds the optimal tree for key a_3, a_4, a_5. As $A[3, 5] = 13 + A[3, 3] + A[5, 5]$, the best subtree there is a_4 as root with a_3 as left subtree and a_5 are the right subtree. We get the following result:



The optimal tree for these 6 values.

Answer. [**to Problem 6.**]

We begin with $d_0(u, v)$ for all nodes. This is a 4×4-matrix holding the immediate edges between any pair of nodes. Note that here only two pairs aren't connected: (a, c) and (b, d). Those would be the only entries filled with ∞.

to:	a	b	c	d
from a	0	2	∞	3
from b	2	0	7	∞
from c	5	7	0	3
from d	3	-1	3	0

We now fill $d_a(u, v)$, where the paths are allowed to either use the direct edge or go through a. Remember: $d_a(u, v) = \min\{d_0(u, v), d_0(u, a) + d_0(a, v)\}$. So we fill up this array:

to:	a	b	c	d
from a	0	2	∞	3
from b	2	0	7	5
from c	5	7	0	3
from d	3	-1	3	0

Note how we did not chance (c, a). It was 7 and now it is the best of the two options: 7 or $5+2 = 7$. We did however change (b, d): it is now $2 + 3 = 5$.

Now we fill $d_{a,b}(u, v)$ were both a and b are permissible for our paths.

to:	a	b	c	d
from a	0	2	9	3
from b	2	0	7	5
from c	5	7	0	3
from d	1	-1	3	0

Note how now the distance from d to a is 1: instead of 3, we replace it with $-1 + 2 = 1$.

Now we fill $d_{a,b,c}(u, v)$ were a, b and c are permissible for our paths. The formula is the same as before: $d_{a,b,c}(u, v) = \min\{d_{a,b}(u, v), \quad d_{a,b}(u, c) + d_{a,b}(c, v)\}$.

to:	a	b	c	d
from a	0	2	9	3
from b	2	0	7	5
from c	5	7	0	3
from d	1	-1	3	0

Finally we fill the shortest paths distances by allowing all vertices as intermediate vertices.

to:	a	b	c	d
from a	0	2	9	3
from b	2	0	7	5
from c	4	2	0	3
from d	1	-1	3	0

Suppose we now wish to find the shortest path from c to a. We recurse on the last table's (c, a)-entry. It is 4, which is strictly smaller than its entry in the previous table (5), so we know the $c \to a$ path uses d. This means we recurse on (c, d) and (d, a) in the previous table.

To find the best $c \to d$ path we recurse over all the tables until we reach the first one (in all of them the (c, d)-entry is the same, 3) so finally we print (c, d).

To find the best $d \to a$ path, we recurse over the tables, and see that it only changed once we allowed the use of b. So we recurse on (d, b) and on (b, a). Those however stayed the same throughout all tables, so again — each recursion reached the initial table and so we print (d, b) and then (b, a).

The result is that we print the path $(c, d), (d, b), (b, a)$.
