University of Glasgow

Friday 11 December 2020
1.00 pm – 4.00 pm
(3 hour)

DEGREES of MSci, MEng, BEng, BSc, MA and MA (Social Sciences)

ALGORITHMICS I (H)

Answer all 4 questions

This examination paper is worth a total of 60 marks.

1. **(a)** In at most **100 words** define what is meant by a border of a string. Give an example of a string of length 6 whose longest border is of length 4.

[3]

Solution:

A border of a string s is a proper substring of s that is both a prefix and a suffix of s. An example of a string of length 6 whose longest border is of length 4 is *ababab* (there are many other examples).

(b) Using your own words and in at most **300 words** describe the contents of the border table that is used in the Knuth-Morris-Pratt (KMP) string searching algorithm. Explain briefly how this table is used to determine the appropriate action when a mismatch is detected between the ith character of the text and the jth character of the string or pattern being searched for.

[7]

Solution:

The border table b is an array which is the same size as the string/pattern $s = s_0 s_1 \dots s_{n-1}$ that is being searched for. The element $b[j]$ is the length of the longest border of the prefix $s[0..j-1] = s_0 s_1 \dots s_{j-1}$ of s where if no border exists $b[j]$ equals 0.

When searching for the string s in the text t, if there is a mismatch between $s[j]$ and $t[i]$ the KMP algorithm decides which character in s should next be compared with $t[i]$. The approach is to 'move' s to the 'right' until the characters to the left of $t[i]$ match. To achieve this, the new value of j is chosen to equal length of a longest border of $s[0..j-1]$, i.e. $b[j]$. When no border exists, we have $b[j] = 0$, and hence we start from the beginning of the string.

(c) Construct the border table of the KMP algorithm when the string being searched for equals:

$$ACGACACGTACGAA$$

where the text is over the alphabet $\Sigma = \{A, C, G, T\}$. [5]

Solution:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
b[i]	0	0	0	0	1	2	1	2	3	0	1	2	3	4

2. Recall given a set of character frequencies a corresponding Huffman tree is constructed as follows:

- first add the leaves of Huffman tree which are characters with their frequencies labelling the nodes;
- second, while there is more than one parentless node, add a new parent to nodes of smallest weight, where the weight of the new parent node equals the sum of the weights of the child nodes.

The table below includes the number of occurrences of each character in the text:

"a simple string to be encoded using a minimal number of bits"

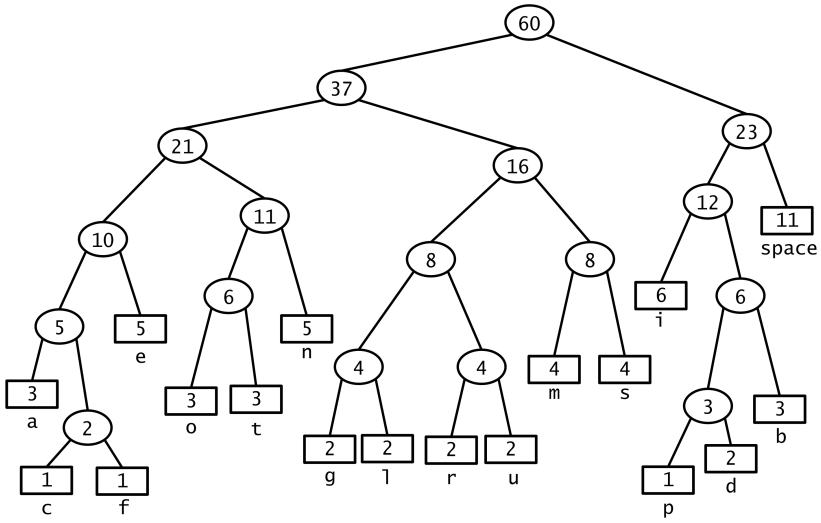
a	*b*	*c*	*d*	*e*	*f*	*g*	*i*	*l*	*m*	*n*	*o*	*p*	*r*	*s*	*t*	*u*	*space*
3	*3*	*1*	*2*	*5*	*1*	*2*	*6*	*2*	*4*	*5*	*3*	*1*	*2*	*4*	*3*	*2*	*11*

(a) Draw a Huffman tree for this text and write the subsequent code words for the characters '*f*' and '*i*'.

[8]

Solution:

The following is a possible Huffman tree for the given frequencies.



The (Boolean) encoding of a character is obtained via the path from root to the corresponding leaf, the the encoding of '*f*' is given by `000011` and the encoding of '*i*' is given by `100`.

(b) Explain in your own words and using no more that **300 words** how, given a set of character frequencies, a corresponding Huffman tree can be constructed with a complexity of $O(m \log m)$ where m is the number of characters appearing in the text. You do not need to describe in any detail standard operations based on any underlying data structure that you might use.

[7]

Solution:

Initially one needs to construct a heap where the nodes correspond to the characters labelled by their frequencies (i.e. the first step of the Huffman tree construction). Since there are m characters, it follows that it takes $O(m)$ time to initially build the heap.

Now considering each iteration, one must find and remove the two nodes of the Heap with minimum labels and then add a new node where the labelling equals the sum of the frequencies. We have that, both deleting a minimal element in a Heap and inserting a new element into a Heap have complexity $O(\log n)$ for a heap of size n. Using the fact that at each iteration we make the heap smaller (we remove two nodes and add a single node), it follows that the maximum size of the Heap is m, and hence the complexity of the operations at each iteration is bounded by $O(\log m)$.

To compute the overall complexity, we need to find the number of iterations required to build the Huffman tree. Since the constructed Huffman tree is a complete binary tree with m leaves, it follows that the tree has $2m-1$ nodes. Hence, we will require $(2m-1)-m = m-1$ iterations to add the remaining nodes of the Huffman tree. This yields an overall complexity of $O(m)+O(m\log m) = O(m\log m)$.

3. **(a)** Explain what it means for a problem to be NP-complete and then discuss in your own words (using at most **150 words**) the implications, from the algorithmic point of view, of proving that a decision problem is NP-complete.

[5]

Solution:

NP is the class of all decision problems that can be solved by a polynomial-time non-deterministic algorithm.

A polynomial-time reduction from a decision problem Π' to a decision problem Π is an polynomial-time algorithm that takes as input an arbitrary instance I' of Π' and produces as output an instance I of Π such that I is a 'yes'-instance of Π if and only if I' is a 'yes' instance of Π'.

A decision problem Π is NP-complete if

1. Π is a member of the class NP
2. for every problem Π' in NP there is a polynomial-time reduction from Π' to Π.

Proving that a problem Π is NP-complete means that it is very unlikely that an algorithm can be found for Π that has polynomial-time worst-case complexity. For the existence

of such an algorithm would imply P equals NP, and therefore that polynomial-time algorithms would exist for all problems in NP, thought to be extremely improbable. Hence, for even moderate sizes, there are at least some instances of the problem that we can expect to be unsolvable in practice.

(b) Consider the following problems:

- **Graph colouring (GC)**
  Instance: a graph G and a target integer k.
  Problem: can one of k colours be attached to each vertex of G so that adjacent vertices always have different colours?
- **Workshop Programming (WP)**
  Instance: given a set of workshops W and k dates, workshop attendees A, a mapping $f : W \to 2^A$ such that $f(w)$ returns the set of attendees of the workshop w and an integer k.
  Problem: Is there a workshop progamme over the k dates such that every attendee has at most one workshop on any date?

Under the assumption that **GC** is NP-complete, prove that **WP** is also NP-complete.

Hint: consider when each attendee attends exactly two workshops.

[10]

Solution:
First show **WP** is in NP. Consider the algorithm that first (non-deterministically) guesses one of k dates for each workshop and then checks that for each attendee there is no conflict. Clearly this algorithm runs in polynomial time and we have:

- for a 'yes'-instance I of **WP** there is some execution that returns 'yes';
- for a 'no'-instance I of **WP** there is no execution that returns 'yes'.

To show **WP** is NP-complete it remains to give a polynomial reduction from **GC** to **WP**. For an instance $G = (V,E)$ and k of **GC** we construct an instance of **WP** where:

- the set of workshops W is the set of vertices V of the graph G;
- the set of attendees A is the set of edges E of the graph;
- the function $f(w) = \{e \mid e \in E \wedge w \in e\}$, i.e. $f : V \to E$ returns the edges that a vertex is incident to;
- the integer k remains unchanged from the **GC** instance.

We now show there exists a programme for the workshops over k dates dates with no conflict if and only if G can be coloured with k colours.

- Suppose G can be coloured by k colours. Consider a programme where each date corresponds to a distinct colour and a workshop $w \in V$ is scheduled on the date it is coloured. For an arbitrary attendee a and two workshops w_1 and w_2 that are attended by a, i.e. $a \in f(w_1)$ and $a \in f(w_2)$, by construction a is an edge between w_1 and w_2, and hence colours associated with w_1 and w_2 are different. Since the workshops w_1 and w_2 were arbitrary it follows that there will be no conflicts for this attendee. Furthermore, since the attendee was arbitrary, it follows that there a workshop proggramme consisting of k dates such that no attendee has a conflict.
- Suppose there exists an workshop programme of k dates without conflicts. We associate with each date a different colour and for each vertex we assign the colour of the date of the corresponding workshop. Now, for an arbitrary edge $e = \{w_1, w_2\}$, by construction there exists an attendee such that $e \in f(w_1)$ and $e \in f(w_2)$. Therefore, since the attendee does not have any conflicts, the workshops w_1 and w_2 must be on different dates, and hence w_1 and w_2 have different colours. Since the edge was arbitrary, it follows the graph is k colourable.

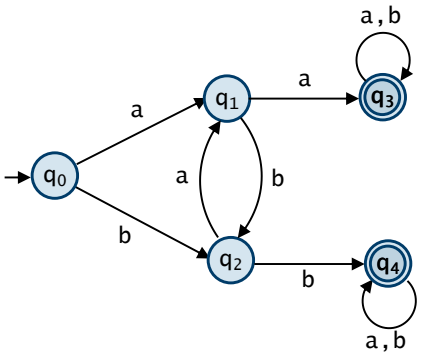
It remains to show the reduction can be performed in polynomial time. This follows, from the fact that to build the edges one must go through each pair of workshops and corresponding sets of attendees once adding edges when necessary.

4. **(a)** Describe, using a diagram or otherwise, a deterministic finite state automaton to recognize the language L consisting of all strings over the alphabet $\{a, b\}$ that contain two consecutive a's or two consecutive b's.

[4]

Solution:

A possible solution for a DFA to recognise all strings over the alphabet $\{a, b\}$ that start and end with the same character:



(b) Give a regular expression for the language L.

[2]

Solution:

Two possible a regular expression for this language are:

$$((a|b)^*aa(a|b)^*)|((a|b)^*bb(a|b)^*)$$

$$((a|b)^*(aa|bb)(a|b)^*)$$

other solutions are possible.

(c) Describe, using a suitable form of pseudocode, or otherwise, a Turing Machine that recognizes the language consisting of all strings over the alphabet $\{a,b\}$ that contain more a's than b's.

[9]

Solution:

The idea is to delete one a and one b, until the tape is empty (reject) or contains only a's (accept) or only b's (reject). In the initial state of the Turing machine we assume an equal number of a's and b's have been deleted and therefore only return to this state (restart) when this is indeed the case.

The pseudo-code is as follows:

```
if (blank) reject; // equal number of a's and b's
else // tape not empty
    read symbol in current square and remember it;
    delete symbol and move right;
    if (blank)
        if (remembered symbol was a) accept;
        else reject; // i.e. remembered symbol was b
    else if (symbol found not equal to remembered symbol)
        delete symbol and restart;
    else // symbol equals remembered symbol
        delete symbol and move right searching for other symbol;
        if (none found) // i.e. reach a blank
            if (remembered symbol was a) accept;
            else reject; // i.e. remembered symbol was b
        else // other symbol found
            delete symbol and write remembered symbol;
            move left over non-blanks;
            move right and restart;
```


END OF QUESTION PAPER