University of Glasgow

Tuesday, 24 April 2018
9.30 am – 11.00 am
(1 hour 30 minutes)

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.

The use of calculators is not permitted in this examination.

INSTRUCTIONS TO INVIGILATORS: Please collect all exam question papers and exam answer scripts and retain for school to collect. Candidates must not remove exam question papers.

1. **(a)** Define what is meant by a border of a string. Give an example of a string of length at least 3 whose longest border is of length 1.

[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 3 whose longest border is of length 1 is *aba* (there are many other examples).

(b) Describe the contents of the border table that is used in the Knuth-Morris-Pratt (KMP) string searching algorithm for a string or pattern $s = s_0 s_1 \ldots s_{n-1}$. 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.

[9]

Solution:
The border table b is an array which is the same size as the string or pattern $s = s_0 s_1 \ldots s_{n-1}$ that is being searched for. The element $b[j]$ is the length of the longest border of the prefix $s[0..j-1] = s_0 s_1 \ldots s_{j-1}$ 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 KPM 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. In the case when $j = 0$ we instead increment i and j remains equal to 0.

(c) State two advantages that the KMP algorithm has over a naive 'brute-force' approach.

[3]

Solution:
First, the KMP algorithm has linear time worst-case complexity, i.e. the complexity equals $O(m+n)$ where m is the length of the string and n is the length of the text, while the brute-force approach has $O(mn)$ complexity. Second, it is an on-line algorithm, i.e. the text can be processed one character at a time and the character can be discarded as soon as it has been processed, so there is never any need to 'back up' in the text.

2. For a weighted graph $G = (V, E, wt)$ explain the following concepts:

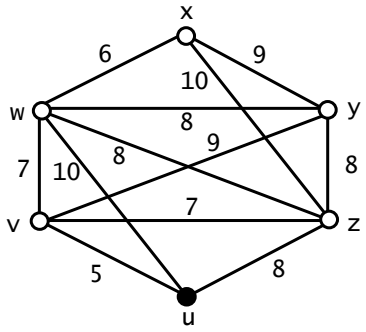
(a) a *spanning tree* of G; [2]

> **Solution:**
> A spanning tree is a subgraph of G which is both a tree and includes (spans) every vertex.

(b) the *weight* of a spanning tree of G; [1]

> **Solution:**
> The weight of a spanning tree is the sum of the weights of the edges appearing in the tree.

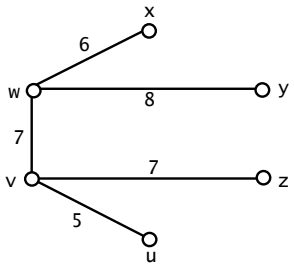
(c) Apply Dijkstra's refinement of the Prim-Jarnik algorithm to find a minimum weight spanning tree of the weighted graph G shown below, using vertex u as the starting tree-vertex.

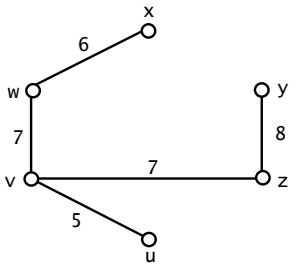


wt({u,v}) =5
wt({u,w})=10
wt({u,z})=8
wt({v,w})=7
wt({v,y})=9
wt({v,z})=7
wt({w,x})=6
wt({w,y})=8
wt({w,z})=8
wt({x,y})=9
wt({x,z})=10
wt({y,z})=8

Include in your answer the steps performed by the algorithm through the changes to the attribute "best tree vertex" for each vertex of the graph. [9]

> **Solution:**
> Two possible minimum spanning trees are given below (other solutions are possible).





Below gives the main steps of the algorithm including the changes to the attribute "best tree vertex" as required in the question. Notice this solution is not unique as during step 2 one could have chosen z instead of w.

- step 1:

q	$q.bestTV$	$wt(\{q.bestTV, q\})$
u	-	-
v	u	5
w	u	10
x	u	∞
y	u	∞
z	u	8

- step 2:

q	$q.bestTV$	$wt(\{q.bestTV, q\})$
u	-	-
v	-	-
w	v	7
x	u	∞
y	v	9
z	v	7

- step 3:

q	$q.bestTV$	$wt(\{q.bestTV, q\})$
u	-	-
v	-	-
w	-	-
x	w	6
y	w	8
z	v	7

- step 4:

q	$q.bestTV$	$wt(\{q.bestTV, q\})$
u	-	-
v	-	-
w	-	-
x	-	-
y	w	8
z	v	7

- step 5:

q	$q.bestTV$	$wt(\{q.bestTV, q\})$
u	-	-
v	-	-
w	-	-
x	-	-
y	w	8
z	-	-

- step 6:

q	$q.bestTV$	$wt(\{q.bestTV, q\})$
u	-	-
v	-	-
w	-	-
x	-	-
y	-	-
z	-	-

(d) Is the minimum weight spanning tree unique? Justify your answer. [3]

Solution:
The tree is not unique since when y is added to the tree there are two tree vertices which yield the minimum weight (w and z), and hence either edge ($\{w,y\}$ or $\{y,z\}$) could have been added to the tree at this point.

3. **(a)** What is meant by each of the following:

(i) the class NP; [1]

(ii) a polynomial-time reduction; [2]

(iii) the statement that a given decision problem Π is NP-complete. [3]

(b) Explain carefully the implications, from the algorithmic point of view, of proving that a decision problem is NP-complete. [2]

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.

(c) Consider the following two decision problems:

Name: Hamiltonian Path (HP)
Instance: undirected (unweighted) graph G;
Question: is there a path in G that visits every vertex exactly once?

Name: Degree-constrained spanning tree (DCST)
Instance: undirected (unweighted) graph G and target integer K;
Question: does G have a spanning tree in which all vertices have degree $\leq K$?
(i.e. the number of edges in the tree incident on any vertex is less than or equal to K)

Suppose that you have a proof that **HP** is NP-complete, present a formal proof

showing that **DCST** is NP-complete. [7]

Hint: Consider the correspondence between a spanning tree and a Hamiltonian path?

Solution:
First the **DCST** problem can be solved by the following nondeterministic polynomial time algorithm:

1. For each edge in the graph G, choose nondeterministically if it is to be included in a spanning tree T.
2. Check that T is indeed a spanning tree and that each vertex has degree less than or equal to K.

Since each step can be performed in polynomial time, it follows that **DCST** is in NP. Now if we can reduce any NP-complete problem to **DCST** it follows by definition of NP-completeness (in particular, using the fact that any NP-complete problem is polynomial time reducible from any problem in NP) and the transitivity of polynomial time reductions, that **DCST** is NP-complete.

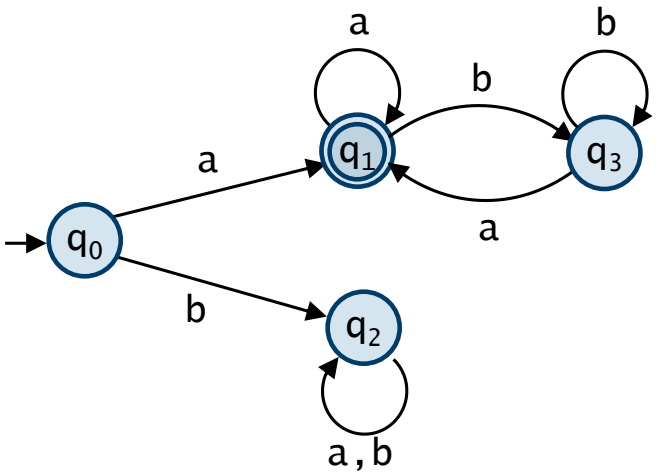
If we can show a graph G has a Hamiltonian path if and only if it has a spanning tree with vertex degree ≤ 2, then we can reduce the **HP** problem for a graph G to the **DCST** problem over the same graph G (with $K = 2$) and, therefore **DCST** is NP-complete.

It remains to show that a graph G has a Hamiltonian path if and only if it has a spanning tree with vertex degree ≤ 2. It is easy to see that such a spanning tree is a Hamiltonian path. In particular, as it spans all vertices, all vertices have degree greater than 0 and only two vertices can have degree equal to 1. So the spanning tree is a Hamiltonian path. If, on the other hand, G contains a Hamiltonian path, this path must be a spanning tree since the path visits every node and a path trivially is a tree.

4. **(a)** Explain the language represented by the regular expression $a|(a(a|b)^*a)$ and then give a deterministic finite state automaton over the alphabet $\Sigma = \{a, b\}$ that accepts this language. [6]

Solution:
The language can be described as all strings that start and end with an a (including a single a). A deterministic finite state automaton that represents this language is given

by (other solutions are possible):



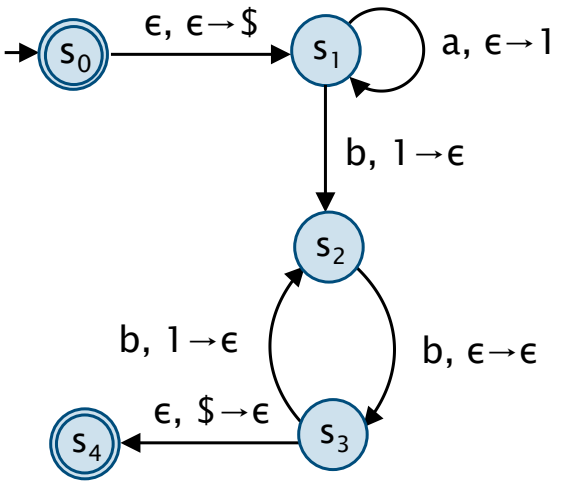
(b) Design a pushdown automaton that recognises the strings $\{a^n b^{2n} \mid n \geq 0\}$ over the alphabet $\Sigma = \{a, b\}$. Assume the stack symbols are 1 together with the special symbol \$. [9]

Solution:

A solution is the following PDA program (alphabet is $\{a, b\}$):

- s_0 is the start state and s_0 and s_4 are the only accepting states;
- $(s_0, \varepsilon, \varepsilon) \to (s_1, \$)$ move to s_1 and push \$ onto stack (\$ is the special symbol);
- $(s_1, a, \varepsilon) \to (s_1, 1)$ read a, then push 1 onto stack and remain in s_1;
- $(s_1, b, 1) \to (s_2, \varepsilon)$ read b and 1 is on top of stack, then pop stack and move to s_2;
- $(s_2, b, \varepsilon) \to (s_3, \varepsilon)$ read b and move to s_3;
- $(s_3, b, 1) \to (s_2, \varepsilon)$ read b and 1 is on top of stack, then pop stack and move to s_2;
- $(s_3, \varepsilon, \$) \to (s_4, \varepsilon)$ if \$ is the top of the stack, then pop stack and move to s_4.

Below is a diagrammatical representation of this automaton (marks will not be lost if only the transition rules or diagram is given or if only a textual description of the pushdown automaton is given).



Note: One could also pop a 1 for every even b rather than every odd (i.e. pop going from s_2 to s_3 rather than from s_1 to s_2 and s_3 to s_2). An alternative solution is to push two 1's to the stack for each a read and then pop the stack for each b read. Other solutions are also possible.

END OF QUESTION PAPER