KØBENHAVNS
UNIVERSITET

Diskret Matematik og Formelle Sprog: Exam April 1, 2020

Due: April 2 at 9 am.
Submission: Please submit your solutions via *Digital eksamen*. From the point of view of grading, we would prefer if you could typeset your solutions in LaTeX or some other math-aware typesetting system. This is *not* a requirement, however, and how you write your solutions will not affect the grading as long as you make sure that everything that you write is clearly legible (and leave reasonable margins on all sides). Please try to be precise and to the point in your solutions and refrain from vague statements. *Write your solutions in such a way that a fellow student of yours could read, understand, and verify your solutions.*
Collaboration: All problems should be solved individually. No communication or collaboration is allowed, and solutions will be checked for plagiarism.
Reference material: All exam aids and support material are allowed, including what you can find on the internet (except that, as stated above, no communication is allowed).
About the problems: Note that the problem are of quite different types, and are *not* sorted in increasing order of difficulty. You are not necessarily expected to be able to solve all of the problems. Please do not hesitate to send a private message on Absalon if any problem statement is unclear. Good luck!

1 (60 p) In the following snippet of code `A` is an array indexed from 1 to n containing elements that can be compared using the operator $<$.

```
stop := FALSE
i := 1
while (i <= n and not(stop))
    j := 1
    found := FALSE
    fail := FALSE
    while (j <= n and not(fail))
        if (A[i] < A[j])
            if (found)
                fail := TRUE
            else
                found := TRUE
        j := j+1
    if (found and not(fail))
        stop := TRUE
    else
        i := i+1
if (stop)
    return A[i]
else
    return "failed"
```

1a Explain in plain language what the algorithm above does.

1b Provide an asymptotic analysis of the running time as a function of the array size.

1c Improve the code to run faster while retaining the same functionality. How much faster can you get the algorithm to run? Analyse the time complexity of your new algorithm. Can you prove that it is asymptotically optimal?

2 (70+ p) In the following snippet of code `A` is an array indexed from 1 to n containing integers.

```
found := FALSE
i := 1
while (i <= n and not(found))
    j := 1
    while (j <= n and not(found))
        k := 1
        while (k <= n and not(found))
            if (i != j and j != k and i != k and A[i]+A[j]+A[k] == 0)
                found := TRUE
            else
                k := k+1
        if (not(found))
            j := j+1
    if (not(found))
        i := i+1
if (found)
    return (i, j, k)
else
    return "failed"
```

2a Explain in plain language what the algorithm above does.

2b Provide an asymptotic analysis of the running time as a function of the array size.

2c Improve the code to run faster while retaining the same functionality. How much faster can you get the algorithm to run? Analyse the time complexity of your new algorithm. Can you prove that it is asymptotically optimal?

2d *Bonus problem (worth 40 p extra):* This is in fact a well-known research problem. What information can you find about this on the internet? What are the best known upper and lower bounds for algorithms solving this problem? Explain how you dug up the information, and give references to where it can be found.

3 (40 p) For positive integers $a_1, a_2, \ldots, a_k$, define $\gcd(a_1, a_2, \ldots, a_k)$ to be the largest positive integer d such that d divides every a_i and any positive integer c that divides every a_i also has to divide d. Is it true that there are integers m_i, not necessarily positive, such that $d = \sum_{i=1}^{k} m_i a_i$? Prove this or give a simple counter-example.

4 (110 p) Suppose that we are given an array $A = [5, 6, 4, 7, 3, 8, 2, 9, 1, 10]$ to be sorted in increasing order.

4a Run merge sort by hand on this array, and show in detail in every step of the algorithm what recursive calls are made and how the results from these recursive calls are combined.

4b Run heap sort by hand on this array, and show in detail in every step of the algorithm what calls to the heap are made and how the array and the heap change.

4c Suppose that we are given another array B that is already sorted in increasing order. How fast do the merge sort and heap sort algorithms run in this case? Is any of them asymptotically faster than the other?

4d Suppose that we are given a third array C that is sorted in *decreasing* order, so that it needs to be reversed to be sorted in the order that we prefer, namely increasing. How fast do the merge sort and heap sort algorithms run in this case? Is any of them asymptotically faster?

4e Suppose that we build a max-heap from an array A. Is it true that once the array A has been converted to a correct max-heap, then considering the elements in reverse order (i.e., $A[n], A[n-1], \ldots, A[2], A[1]$) yields a correct min-heap? Prove this or give a simple counter-example.

5 (80 p) Decide for each of the propositional logic formulas below whether it is a tautology or a contradiction. If neither of these cases apply, then present a satisfying assignment for the formula. It is sufficient (and necessary) for a full score to justify all your answers by presenting truth tables for all the subformulas analogously to how we did it in class, but you are also encourage to *explain* why your answers are correct (and good explanations could compensate fully for minor slips in the truth tables).

(Note that logical not $\neg$ is assumed to bind harder than the binary connectives, but other than that all formulas are fully parenthesized for clarity. We write $\rightarrow$ for logical implication and $\leftrightarrow$ for equivalence.)

5a $((p \rightarrow q) \rightarrow r) \rightarrow ((p \wedge q) \rightarrow r)$

5b $((p \rightarrow q) \wedge (r \rightarrow s)) \leftrightarrow ((p \wedge r) \rightarrow (q \wedge s))$

5c $((p \wedge \neg r) \vee (q \wedge \neg r)) \rightarrow ((p \vee q) \rightarrow r)$

5d $(p \rightarrow (q \vee r)) \vee (\neg(\neg p \vee q) \wedge \neg r)$

6 (100 p) Provide formal proofs of the following claims using proof techniques that we have learned during the course.

6a For all $K \in \mathbb{Z}^+$ it holds that $\sum_{s=1}^{K} s \cdot s! = (K+1)! - 1$.

6b For all $s \in \mathbb{N}$ and all $k > 0$ it holds that $1 + sk \leq (1+k)^s$.

6c For all $q \in \mathbb{N}$ it holds that $2^{4q+2} + 3^{q+2}$ is a multiple of 13.

6d Let $T_1 = 0$, $T_2 = 1$, and $T_i = T_{i-1} + T_{i-2}$ for $i \geq 3$. Prove that for all $i \geq 2$ it holds that

$$1 \leq \frac{T_{i+1}}{T_i} \leq 2 \,.$$

7 (60 p) In this problem we focus on relations. In particular, suppose that $A = \{e_1, e_2, \ldots, e_9, e_{10}\}$ is a set of 10 elements and consider the relation R on A represented by the matrix

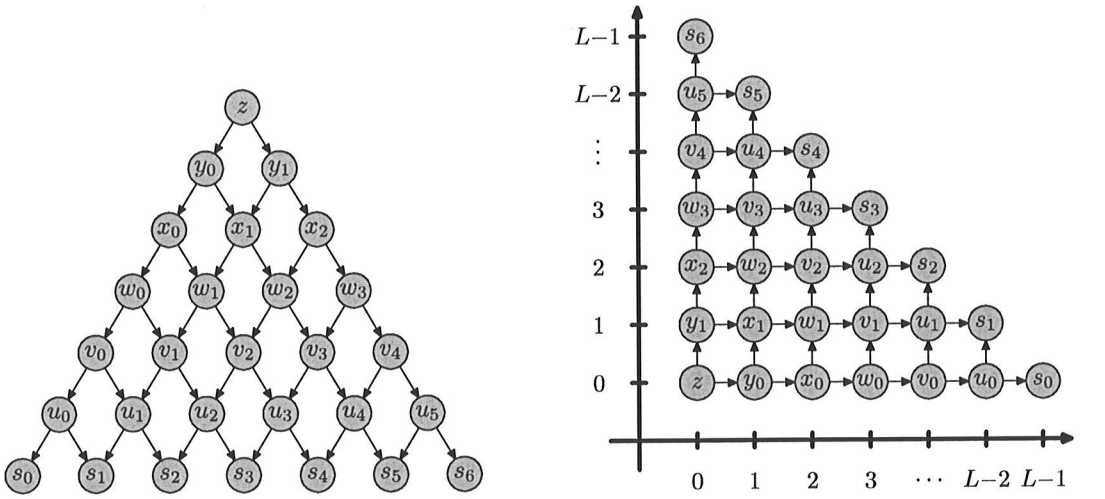
$$M_R = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

(where element e_i corresponds to row and column i).

7a Let us write T to denote the transitive closure of the relation R. What is the matrix representation of T in this case? Can you describe in words what the relation T is?

7b Suppose that we create a new relation S_1 from R by first taking the reflexive closure and then the symmetric closure. What does the matrix representation of S_1 look like? Can you describe in words what the relation S_1 is?

7c Suppose instead that we first take the symmetric closure and then the reflexive closure to derive another relation S_2 from R. Are S_1 and S_2 different relations, or are they the same relation in the end? If the latter case holds, is it true for any relation R on a set A that relations S_1 and S_2 derived in this way will be the same, or can they sometimes be different? Give a proof or present a simple counter-example.



(a) Lattice graph with 7 layers.

(b) Graph as connected points in 2D integer lattice.

Figure 1: Example graph for Problem 8.

8 (100 p) Consider a directed graph that consists of L layers numbered from 0 up to $L-1$, with $i+1$ vertices in every layer i numbered from 0 to i, and with outgoing edges from vertex j in layer i to vertices j and $j+1$ in layer $i+1$. See Figure 1a for an illustration of this graph with 7 layers. We can agree to call this graph a *lattice graph* (because it can be obtained from a fragment of the integer lattice in 2 dimensions as shown in Figure 1b, but this is actually completely irrelevant to this problem).

Suppose we start in the unique source vertex on level 0 and walk along edges in the graph, flipping a fair coin at every vertex to decide whether to go left or right, until we reach one of the sinks in the last layer. For instance, in Figure 1a the walk "left–left–right–left–right–right" would visit vertices z, y_0, x_0, w_1, v_1, u_2, and end in s_3.

8a For every vertex s_i in the lattice graph in Figure 1a, calculate the probability that such a walk ends in vertex s_i. Which vertex is the walk most likely to end up in?

8b For a lattice graph with L layers, where $L \geq 2$ is a positive integer, calculate the probability that a walk as described above ends in vertex j in layer $L-1$ for $j = 0, 1, \ldots, L-1$.

Hint: Use the fact that all walks are equally likely to turn this into a counting problem.

9 (80 p) Consider the graph in Figure 2. We assume that this graph is represented in adjacency list format, with the neighbours in each adjacency list sorted in lexicographic order (i.e., in the order $a, b, c, d, \ldots$).

9a Run the code for depth-first search by hand on this graph, starting in the vertex a. Describe in every recursive call which neighbours are being considered and how they are dealt with. Show the spanning tree produced by the search.

9b Run the code for breadth-first search by hand on this graph, also starting in the vertex a. Describe for every vertex which neighbours are being considered and how they are dealt with, and how the queue changes. Show the spanning tree produced by the search.

9c Suppose that the graph in Figure 2 is modified to give every edge weight 1, and that we run Dijkstra's algorithm starting in vertex a. How does the tree computed by Dijkstra's algorithm compare to that produced by DFS? What about BFS?

10 (110 p) Consider the graph in Figure 3. We assume that this graph is represented in adjacency list format, with the neighbours in each adjacency list sorted in lexicographic order.

10a Generate a minimum spanning tree by running Prim's algorithm by hand on this graph, starting in the vertex a. Use a heap for the priority queue implementation. Describe for every dequeueing operation which neighbours are being considered and how they are dealt with, and show how the heap changes. Also show the tree produced by the algorithm.

10b Generate a minimum spanning tree by running Kruskal's algorithm by hand on this graph. Assume that edges of the same weight are sorted in lexicographic order (so that we would have (a, b) coming before (a, d), which would in turn come before (b, a) for hypothetical edges of equal weight). Describe how the forest changes at each step (but you do not need to describe in detail how the set operations are implemented). Also show the final tree produced by the algorithm.

10c Suppose that some vertex v with several neighbours in a graph G has a unique neighbour u such that the edge (u, v) has strictly smaller weight than any other edge incident to v. Is it true that the edge (u, v) must be included in any minimum spanning tree? Prove this or give a simple counter-example.

10d Suppose that some vertex v with several neighbours in a graph G has a unique neighbour u such that the edge (u, v) has strictly larger weight than any other edge incident to v. Is it true that the edge (u, v) can never be included in any minimum spanning tree? Prove this or give a simple counter-example.

10e Suppose that T is a minimum spanning tree for a weighted, undirected graph G. Modify G by adding some constant $c \in \mathbb{R}^+$ to all edge weights. Is T still a minimum spanning tree? Prove this or give a simple counter-example.

11 (60 p) Consider again the graph in Figure 3 with the same assumptions.

11a Run Dijkstra's algorithm by hand on this graph, starting in the vertex a. Use a heap for the priority queue implementation. Describe for every vertex which neighbours are being considered and how they are dealt with, and show how the heap changes. Also show the tree produced by the algorithm.

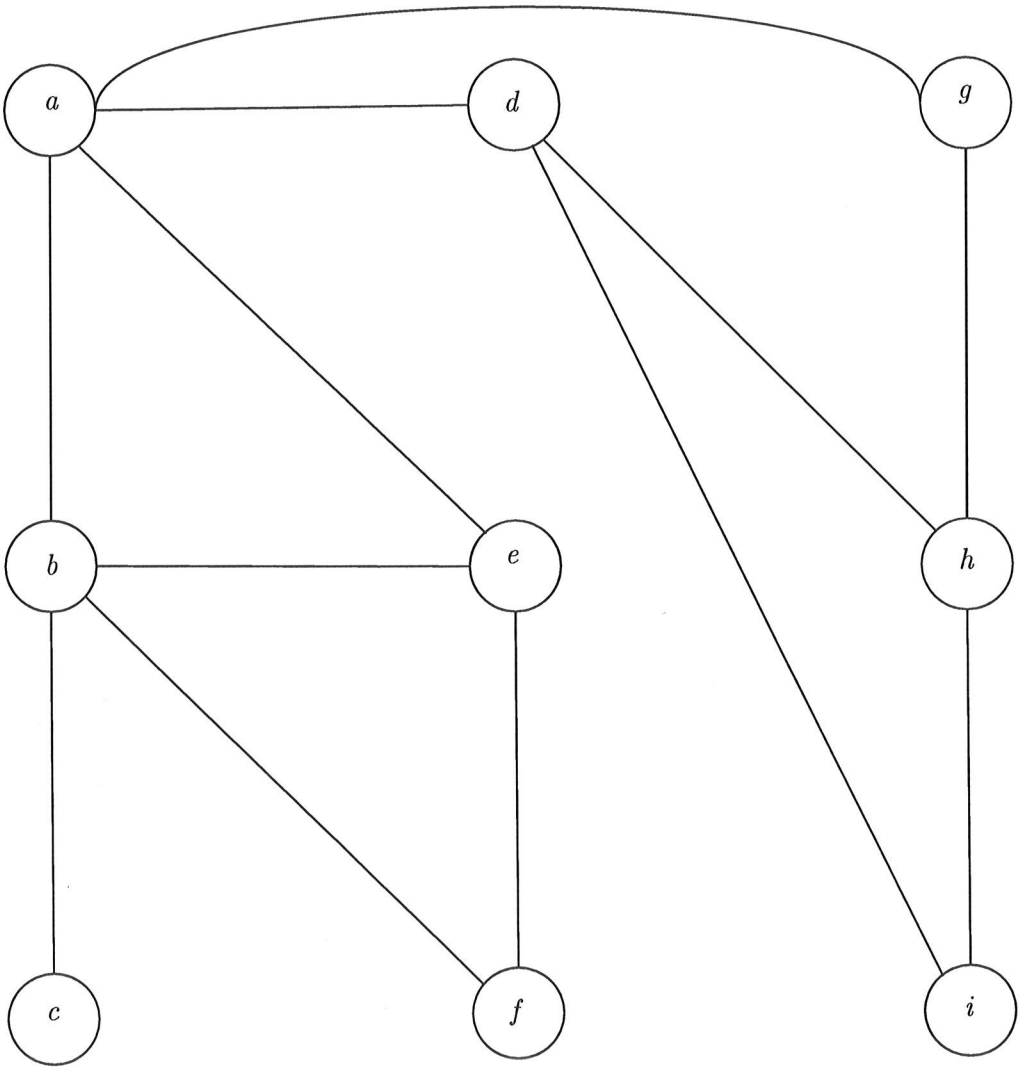


Figure 2: Undirected unweighted graph for graph traversals in Problem 9.

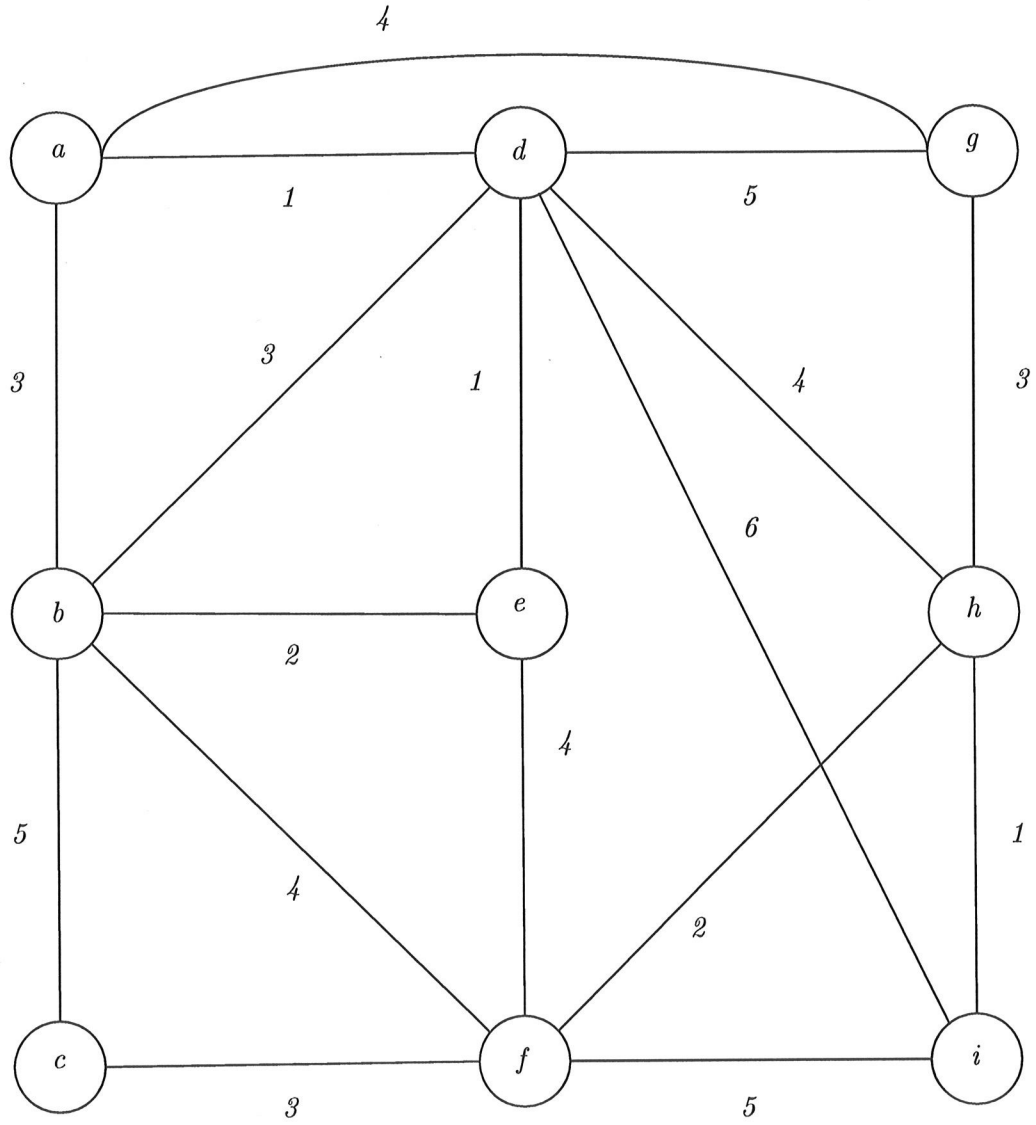


Figure 3: Undirected weighted graph for minimum spanning trees in Problem 10 and shortest paths in Problem 11.

11b If Dijkstra's algorithm is run on an undirected graph, is it true that the spanning tree produced is a minimum spanning tree? Prove this or give a simple counter-example.

12 (120 p) In this problem, we want to understand the languages generated by specified regular expressions and context-free grammars.

12a Which of the words below belong to the language generated by the regular expression $(b(ab)^* a) \mid (b^* ab^* ab^*)$? Motivate briefly your answers.

1. *babababa*
2. *babbabbb*
3. *ba*
4. *babaabaaab*
5. *babbaabbbba*
6. *babba*

12b Consider the following context-free grammars, where a, b, c, d are terminals, A, B, S are non-terminals, and S is the starting symbol.

Grammar 1:

$$S \to abS \quad (1a)$$
$$S \to B \quad (1b)$$
$$B \to aB \quad (1c)$$
$$B \to cB \quad (1d)$$
$$B \to dB \quad (1e)$$
$$B \to \quad (1f)$$

Grammar 2:

$$S \to AbS \quad (2a)$$
$$S \to \quad (2b)$$
$$A \to aAa \quad (2c)$$
$$A \to \quad (2d)$$

Grammar 3:

$$S \to aSa \quad (3a)$$
$$S \to bS \quad (3b)$$
$$S \to \quad (3c)$$

Which of these grammars generate regular languages? For each grammar, write a regular expression that generates the same language, or argue why the language generated by the grammar is not regular.

13 (120 p) In this problem, we want to write regular expressions and context-free grammars generating specified languages.

13a Write a regular expression for the language consisting of all finite (possibly empty) bit strings (i.e., over the alphabet $\{0, 1\}$) that contain an even number of 0s. Examples of such strings are ε (the empty string), 00, 1010001, and 111, whereas 10 and 01010 do not qualify for membership.

13b Write a regular expression for the language consisting of all finite, non-empty bit strings that contain no two consecutive 1s. Examples of strings in this language are 0000, 01010, and 1001, while 111, 00110, and ε do not qualify. For partial credit (if your regular expression is wrong or missing), argue why this language is clearly regular.

13c Give a context-free grammar for the language $\left\{a^m b^n c^{m+n} \,\middle|\, m, n \in N\right\}$. Examples of strings in this language are *aacc* and *abbccc*, whereas *abc* and *abccc* do not make the cut.

14 (120+ p) Consider the regular expression $b^*(ab^*(aab^*|\varepsilon)cb^*)^*$

14a Translate this regular expression to a nondeterministic finite automaton using the method from Mogensen's notes that we learned in class. Make sure to explain which part of the regular expression corresponds to which part of the NFA.

14b Translate the nondeterministic finite automaton to a deterministic finite automaton using the method from Mogensen's notes that we learned in class. Make sure to explain how you perform the subset construction, so that it is possible to follow your line of reasoning.

14c *Bonus problem (worth 20 p extra):* How small a DFA can you produce that accepts precisely the language generated by the regular expression above? (Note that that you can solve this problem even if you did not solve the other subproblems above.)

15 (170 p) In this problem we want to construct parsers for context-free languages. We assume that (,), [,], +, *, and **num** are tokens return by a lexer (i.e., , from the point of view of the parser these are terminals in the alphabet).

15a Consider the following simplified version of the grammar for arithmetic expressions with precedence that we saw in class, where E is the starting symbol:

$$E \to E + E_2 \tag{4a}$$
$$E \to E_2 \tag{4b}$$
$$E_2 \to E_2 * E_3 \tag{4c}$$
$$E_2 \to E_3 \tag{4d}$$
$$E_3 \to \textbf{num} \tag{4e}$$
$$E_3 \to (\, E\,) \tag{4f}$$

Is this an LL(1) grammar?

If your answer is yes, then construct *FIRST*, *FOLLOW*, and *Nullable*, and give pseudo-code for a recursive descent parser.

If your answer is no, then explain why the grammar fails to be LL(1). Is it possible to build a predictive parser for the language in some other way by using more characters of look-ahead?

15b Consider a grammar for parenthesized lists of **num** tokens as follows, with S as the starting symbol:

$$S \to P\ S \tag{5a}$$
$$S \to B\ S \tag{5b}$$
$$S \to N \tag{5c}$$
$$P \to (\, S\,) \tag{5d}$$
$$B \to [\, S\,] \tag{5e}$$
$$N \to \textbf{num}\ N \tag{5f}$$
$$N \to \tag{5g}$$

Is this an LL(1) grammar?

If your answer is yes, then construct *FIRST*, *FOLLOW*, and *Nullable*, and give pseudo-code for a recursive descent parser.

If your answer is no, then explain why the grammar fails to be LL(1). Is it possible to build a predictive parser for the language in some other way by using more characters of look-ahead?