

Dynamic Algorithms Multiple Choice Test

Sample test: only 8 questions – 32 minutes

(Real test has 30 questions – 120 minutes)

Årskort

Name

Each of the following 8 questions has 4 possible answers of which exactly one is correct. For each question, you may select one or more answers by checking the corresponding boxes. Your test is graded as follows:

- If you select only the correct answer, you receive 2 points.
- If you select 2 answers, one of which is correct, you receive 1 point.
- If you select 3 answers, one of which is correct, you receive 0.41 points.
- if you select no answer or all answers, you receive 0 point.
- If you select only one answer, and it is wrong, you receive -0.67 points.
- If you select 2 answers, that are both wrong, you receive -1 point.
- If you select 3 answers, that are all wrong, you receive -1.25 points.

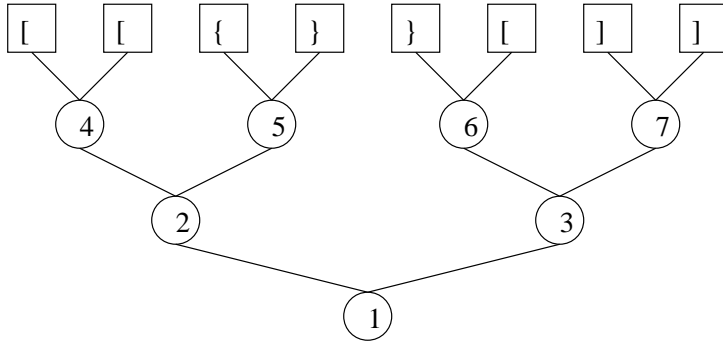
Note that perfect answers yield a score of 16 points and random guessing may give you a negative score. For the logical basis of this score system, see Frandsen and Schwartzbach, A singular choice for multiple choice [<http://doi.acm.org/10.1145/1189136.1189164>].

Example

Consider the following data type for maintaining balance information in a string of parentheses

- `init(s)`. Initialise input to the string s of length n .
- `change(i, c)`, $1 \leq i \leq n$. Replace the i 'th entry in s with parenthesis c .
- `query`. Decide whether s is perfectly balanced.

It is implemented by a binary tree data structure that is illustrated in the figure for the string $s = "[\{\}\]"$.



The data kept at the internal nodes $1, \dots, 7$ for the specific string $[\{\}\]$ is given by the table

node	subtree matches	matched at node	unmatched left	unmatched right
4, 5, 6, 7	true,true,true,true	"##", "{", "##", "##"	"[", "##", "#[", "##",	"##", "##", "}#", "]"
2, 3	true, true	"####", "#[]"	"[#]", "####"	"####", "}##"
1	false	"[#}]##"	"#####"	"#####"

Question 1

This question refers to the previous example describing a binary tree based data structure for maintaining parenthesis balancing information. Consider a change(5, '{') operation applied to the example resulting in the string "[[{}-{}[]]" What data should be kept at nodes 6,3, and 1 after the change?

a

node	subtree matches	matched at node	unmatched left	unmatched right
6,3,1	true, false, false	"##", "{[]}", "#####"	"{[", "####", "[#####"	"##", "####", "#####"

b

node	subtree matches	matched at node	unmatched left	unmatched right
6,3,1	true, true, true	"##", "#[]#", "######"	"{[", "{###", "###{###"	"##", "###", "#####"

c

node	subtree matches	matched at node	unmatched left	unmatched right
6,3,1	true, false, false	"##", "{[]}", "######"	"{[", "####", "###{###"	"##", "####", "#####"

d

node	subtree matches	matched at node	unmatched left	unmatched right
6,3,1	true, true, true	"##", "#[]#", "#####"	"{[", "{###", "[#####"	"##", "###", "#####"

Question 2

This question refers to the previous example describing a binary tree based data structure for maintaining parenthesis balancing information. In the example no details were given on how the data strings at the internal nodes of the binary tree are represented. One may map such a string to a matrix using a homomorphism h and making computations modulo a random prime. Which of the following suggestions for h is part of a *randomized* solution for the parenthesis balancing problem allowing change and balance query operations within time $O(\log^{O(1)} n)$?

a

x	'#'	'['	']'	'{'	'}'
$h(x)$	$\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$	$\begin{pmatrix} 1 & 2 \\ 0 & 1 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 \\ -2 & 1 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 \\ 2 & 1 \end{pmatrix}$	$\begin{pmatrix} 1 & -2 \\ 0 & 1 \end{pmatrix}$

b

x	'#'	'['	']'	'{'	'}'
$h(x)$	$\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$	$\begin{pmatrix} 1 & 2 \\ 0 & 1 \end{pmatrix}$	$\begin{pmatrix} 1 & -2 \\ 0 & 1 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 \\ 2 & 1 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 \\ -2 & 1 \end{pmatrix}$

c

x	'#'	'['	']'	'{'	'}'
$h(x)$	$\begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix}$	$\begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}$	$\begin{pmatrix} 1 & -1 \\ 0 & 1 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 \\ -1 & 1 \end{pmatrix}$

d

x	'#'	'['	']'	'{'	'}'
$h(x)$	$\begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}$	$\begin{pmatrix} 1 & -1 \\ 0 & 1 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 \\ -1 & 1 \end{pmatrix}$	$\begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}$

Question 3

This question refers to the previous example describing a binary tree based data structure for maintaining parenthesis balancing information. In the example no details were given on how the data strings at the internal nodes of the binary tree are represented. Which of the following representations of the strings "matched at node" are part of a solution allowing change and balance query operations within time $O(\log^{O(1)} n)$?

a

A string $s = "[\#\#\#\#]"$ is divided in the middle into 2 strings $s_1 = "[\#\#]"$ and $s_2 = \text{reverse}("}\#\#\#")$. Then blanks '#' are removed and the two strings are kept in a Mehlhorn-Sundar-Uhrig data structure for dynamic sequences.

b

A string $s = "[\#\#\#\#]"$ is transformed into a string consisting of just one kind of parentheses $t = "((\#\#\#\#))"$. Then blanks '#' are removed and the resulting string is kept in a data structure specialized to keep balancing information for just 1 kind of parenthesis.

c

A string $s = "[\#\#\#\#]"$ is divided in the middle into 2 strings $s_1 = "[\#\#]"$ and $s_2 = \text{reverse}("}\#\#\#")$. Without removing blanks the two strings are kept in a Mehlhorn-Sundar-Uhrig data structure for dynamic sequences.

d

A string $s = "[\#\#\#\#]"$ is transformed into a string consisting of just one kind of parentheses $t = "((\#\#\#\#))"$. Without removing blanks '#' the resulting string is kept in a data structure specialized to keep balancing information for just 1 kind of parenthesis.

Question 4

The Frandsen-Miltersen-Skyum implementation of the dynamic word problem for group-free monoids makes use of van Emde Boas Trees. How many van Emde Boas trees are needed for a monoid M satisfying the second case of the Krohn-Rhodes decomposition theorem, i.e.

$$M \setminus \{1\} = \langle a \rangle = \{a, a^2, a^3, \dots, a^k = a^{k+1}\}$$

- a 0
- b 1
- c 2
- d k

Question 5

What is the union-split-find data structure

- a A fully dynamic algorithm for maintaining connected components in an undirected graph
- b An extension to the union-find data structure consisting in a split operation that allows an "undo" of the latest union operation
- c An extension to the union-find data structure that makes it equivalent to a van Emde Boas tree.
- d An extension to the union-find data structure consisting in a split operation that allows the removal of a single element from one of the sets

Question 6

What is the time needed per operation in Eppstein et al.s algorithm for maintaining a minimum spanning forest in a dynamic plane graph?

- a $O(\log n)$
- b $O(\sqrt{n})$
- c $O(\log^2 n \log^* n)$
- d $O(\log^2 n)$

Question 7

What is the cut-value technique for constructing dynamic algorithms for algebraic problems such as prefix computation?

- a* One considers a prefix circuit and defines a subset S of the circuit nodes to be a cut if the output is determined by the values at the cut nodes. Suppose there is a cut S such that the value of each cut node can be computed from at most $d(n)$ input nodes and each cut node influences at most $d(n)$ output nodes then one may derive a solution for dynamic prefix computation of complexity $O(d(n))$.
- b* One considers a prefix circuit and defines a subset S of the circuit nodes to be a cut if the output is determined by the values at the cut nodes. Suppose there is a cut S such that each output node depends on at most $d(n)$ nodes in the cut and each input node influences at most $d(n)$ nodes in the cut then one may derive a solution for dynamic prefix computation of complexity $O(d(n))$.
- c* One interprets a prefix circuit as a flow graph and defines a cut to be a subset V of the circuit nodes such that V contains all the output nodes and no input nodes. If the value of each node in V can be computed from $d(n)$ nodes outside V and if the values of each node in V influences the values of at most $d(n)$ other nodes then we may derive a solution for dynamic prefix computation of complexity $O(d(n))$.
- d* One interprets a prefix circuit as a flow graph and defines a cut to be a subset V of the circuit nodes such that V contains all the input nodes and no output nodes. If the value of each node in V can be computed from $d(n)$ other nodes and if the values of each node in V influences the values of at most $d(n)$ nodes outside V then we may derive a solution for dynamic prefix computation of complexity $O(d(n))$.

Question 8

Consider the six-coloring scheme from Mehlhorn-Sundar-Uhrig applied to the sequence of numbers $13 = 1101_2$, $22 = 10110_2$, $6 = 110_2$, $22 = 10110_2$, $27 = 11011_2$. In one coloring step one reaches

initial	after one step
1101	1
10110	0
110	1000
10110	1001
11010	100

What coloring is reached after an additional step?

a

initial	after one step	after two steps
1101	1	1
10110	0	110
110	1000	0
10110	1001	1
11010	100	0

b

initial	after one step	after two steps
1101	1	1
10110	0	0
110	1000	1
10110	1001	111
11010	100	110

c

initial	after one step	after two steps
1101	1	1
10110	0	0
110	1000	110
10110	1001	111
11010	100	1

d

initial	after one step	after two steps
1101	1	1
10110	0	0
110	1000	111
10110	1001	1
11010	100	0

