Zkouškový Test PAL B **Jméno**

V otázkách 13. a 14. vybírejte vždy jedinou správnou variantu odpovědi.
V ostatních otázkách může být správných variant odpovědí více.
Správně zodpovězená otázka je taková, která určí správně všechny varianty odpovědí a je hodnocena jedním bodem. Pokud některá varianta je určena chybně, otázka je hodnocena 0 body.

1.

Četnosti znaků v textu jsou dány tabulkou. Pro statický Huffmanův kód znaků a pro odpovídající Huffmanův strom T platí, že

a	*b*	*c*	*d*	*e*	*f*
16	5	4	10	2	8

a) kódy znaků d a f musí začínat stejným symbolem,
b) kódy znaků d a f musí končit stejným symbolem,
c) hloubka T je 4 (když hloubka kořene je 0),
d) dva znaky mají délku kódu rovnou hlubce stromu,
e) T má 1 list v hloubce 1.

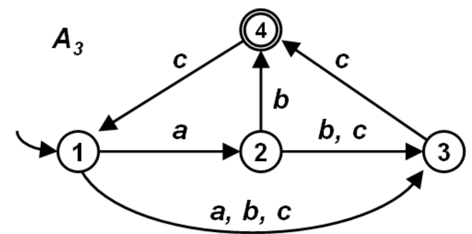
2.

Pro vzorek $P = bbbcc$ nad abecedou $\{a, b, c\}$je sestrojen nedeterministický automat A_1, pomocí nějž lze v textu vyhledávat řetězce, které mají od P Levenshteinovu vzdálenost rovnu nejvýše 2. Pro A_1 platí, že

a) obsahuje ε-přechody,
b) A_1 vyhledá řetězec *bbac*,
c) A_1 má méně než 14 stavů,
d) má alespoň 5 koncových stavů,
e) přechodový diagram A_1 (jakožto orientovaný graf) obsahuje cyklus délky 3 nebo více.

3.

K danému automatu A_3 je sestrojen s použitím standardního algoritmu převodu NKA na DKA odpovídající deterministický automat B_3. Přechodová tabulka B_3 obsahuje řádek



a)-	1	23	3	3	F
b)	23		34	34	
c)	23		34	4	F
d)	14	23	3	13	F
e)	13	23	3	34	

4.

Jsou dány dva jazyky R_1 a R_2 nad abecedou $A = \{x, y, z\}$. Jazyk R_1 se skládá ze všech takových slov, která obsahují podřetězec *xyz*. Jazyk R_2 se skládá ze všech slov, v nichž každý podřetězec délky 2 je prvkem množiny $\{xx, xy, yz, xz\}$.

a) $R_1 \cap R_2$ je konečný jazyk,
b) R_1 je regulární jazyk,
c) Doplněk R_2 není bezkontextový jazyk,
d) $R_1 \subseteq R_2$,
e) R_2 obsahuje slova délky menší než 3.

5.

Jazyk L nad abecedou $\{0,1\}$ je generován gramatikou G. G má pět pravidel:
$S \rightarrow SAS \mid SS \mid 1$;
$A \rightarrow A0 \mid 0$.
S je počáteční symbol. Platí

a) L je regulární jazyk,
b) L lze rozpoznat deterministickým konečným automatem,
c) L obsahuje slovo 1001101,
d) všechna slova L mají lichou délku,
e) v L existuje slovo $w = w_1w_2...w_n$ ($w_i \in \{0,1\}$, $n > 1$), takové, že $w_1 \neq w_n$.

6.

Pomocí Boyer-Mooreova algoritmu hledáme výskyt vzorku v textu, který se skládá pouze ze znaku *a* opakovaného *n* krát. (text: *aaaaaa...a*). Hledaný vzorek je *ababab*. Označme $f(n)$ počet porovnání dvojic znaků, které algoritmus provede nad těmito daty. Pak platí

a) $f(n) \in \Theta(6)$,
b) $f(n) \in O(n)$,
c) $f(n) \in \Omega(\log(n))$,
d) $f(n) \in O(n \cdot \log(n))$,
e) $f(n) \in \Omega(n^2)$.

7.

a) 1, b) 10, e) 11, c) 100, d) 11100.	$\delta(q, 1, S) = (q, BS)$ $\delta(q, 1, B) = (q, BB)$ $\delta(q, 0, B) = (r, \varepsilon)$ $\delta(r, 0, B) = (r, \varepsilon)$ $\delta(r, \varepsilon, B) = (t, B)$ $\delta(t, \varepsilon, B) = (t, \varepsilon)$ $\delta(t, \varepsilon, S) = (t, \varepsilon)$

Je dán zásobníkový automat P se stavy q, r, t, který přijímá slova prázdným zásobníkem. Vnější abeceda P je $\{0, 1\}$, zásobníková abeceda P je $\{S, A\}$. Na začátku práce je v zásobníku symbol S. Přechody P jsou dány uvedenými vztahy (vrchol zásobníku je vlevo). P přijme slovo

8.

a) $M[S][\varepsilon] = \varepsilon, 2$ b) $M[S][c]$ = není definováno (= chyba v analyzovaném slově) c) $M[A][b] = bA, 4$ d) $M[B][c] = cC, 7$ e) $M[C][c] = cC, 7$.	(1) $S \to aAbBbS$ (2) $S \to \varepsilon$ (3) $A \to aBC$ (4) $A \to bA$ (5) $B \to aB$ (6) $B \to \varepsilon$ (7) $C \to cC$ (8) $C \to \varepsilon$

Gramatika G má neterminální symboly S, A, B, C, terminální symboly a, b, c, startovní symbol S a uvedená pravidla. Označme $M[X][y]$, resp $M[X][\varepsilon]$ prvek rozkladové tabulky gramatiky G, který odpovídá neterminálu X a terminálu y, resp. neterminálu X a prázdnému řetězci ε. Která tvrzení o tabulce M platí?

9.

a) terminální symbol e b) neterminální symbol S. c) terminální symbol d. d) neterminální symbol B. e) řetězec 'cBS'.	(1) $S \to aAB$ (2) $S \to bBS$ (3) $S \to \varepsilon$ (4) $A \to cBS$ (5) $A \to \varepsilon$ (6) $B \to dB$ (7) $B \to e$

Gramatika G má neterminální symboly S, A, B, terminální symboly a, b, c, d, e, startovní symbol S a uvedená pravidla. Množina FOLLOW(A) obsahuje

10.

V jaké třídě složitosti je funkce `b` v následujícím programu vzhledem k velikosti pole `p` (budeme ji značit n)? Předpokládejte, že funkce `malloc` (funkce `malloc` naalokuje datovou strukturu velikosti argumentu a vrací na tuto strukturu ukazatel) a `free` (funkce `free` uvolní z paměti naalokovanou datovou strukturu, kterou dostane v argumentu) pracují v konstantním čase vzhledem k velikosti pole `p`. Dále předpokládejte, že velikost proměnné `N` vždy před a po provedení funkce `a` koresponduje s velikosti pole `p`.

a) $O(1)$
b) $O(n*\log(n))$
c) $O(n^2)$
d) $\Omega(1)$
e) $\Omega(\log(n))$
f) $\Omega(n^2)$
g) $\Theta(n^2)$
h) $\Theta(n)$

```
int N = 0;
int * p;
void b (void)
{
  int i=0;
  while ((i < N) && (p[i] == 1)) {
        p[i] = 0;
        i++;
  }
  if (i==N) {
        int m = N+1;
        int j;
        if (N != 0) free(p);
        p = malloc(m*sizeof(int));
        for (j = 0; j<m; j++) p[j]=0;
        N = m;
  }
  p[i] = 1;
}
```

11.

V jaké amortizované třídě složitosti je funkce `b` z předchozího programu vzhledem k velikosti pole `p` (budeme ji značit n)? Předpokládejte, že funkce `malloc` a `free` pracují v konstantním čase vzhledem k velikosti pole `p`.

a) **$O(1)$**
b) **$O(n*\log(n))$**
c) **$O(n^2)$**
d) **$\Omega(1)$**
e) $\Omega(\log(n))$
f) $\Omega(n^2)$
g) $\Theta(n^2)$
h) $\Theta(1)$

12.
Mějme libovolnou čtvercovou matici A racionálních čísel s rozměry n×n. Rozhodněte, zda platí některá z následujících tvrzení:

a) Matici A lze rozložit pomocí *LUP dekompozice* na matice L, U a P, kde platí, že PA=LU.
b) Pokud má matice A hodnost n, lze k ní najít inverzní matici v čase $O(n^3)$.
c) Pokud je matice A diagonální s hodností n, je po provedení *LUP dekompozice* na A matice P jednotková.
d) Pro matici A takovou, že k ní existuje inverzní matice, lze *LUP dekompozici* provést v čase $O(n^2)$.
e) Předpokládejme, že bychom reprezentovali matici A jako pole čísel s pohyblivou řádovou čárkou dle IEEE 754. Přesnost nalezení inverzní matice k A pomocí *LUP dekompozice* nelze zvýšit permutací některých řádků v A před provedením *LUP dekompozice*.

13.

Kolik hran má úplný neorientovaný graf na 12 vrcholech?

a) 32
b) 56
c) 66
d) 78

14.

Jaký je maximální počet hran u libovolného stromu s 2414894 vrcholy?

a) 2414893
b) 402596
c) 1236995
d) 69893623

15.
Uvažujte následující program napsaný v pseudokódu:

Poznámka: metoda `G.neighbors_of(y)` vrací seznam všech vrcholů, které jsou spojeny hranou s vrcholem `y` v grafu `G`.

```
 1) Vertices visited = empty;
 2) procedure search(Graph G, Vertex start_vertex )
 3) {
 4)       while (to_visit.number_of_elements() != 0) {
 5)             if (v not in visited) then {
 6)                   visited.add(v);
 7)                   for all Vertex x in G.neighbors_of(v) {
 8)                   }
 9)             }
10)       }
11) }
```

Rozhodněte, která z následujících tvrzení platí:

a) **Přidáním řádku „`Queue to_visit = empty;`" za řádek 1),**
řádku „`to_visit.push(start_vertex);`" za řádek 3),
řádku „`Vertex v = to_visit.pop();`" za řádek 4 a
řádku „`to_visit.push(x);`" za řádek 7)
vznikne následující procedura, která prohledává graf do šířky.

```
Vertices visited = empty;
Queue to_visit = empty;
procedure search(Graph G, Vertex start_vertex )
{
            to_visit.push(start_vertex);
            while (to_visit.number_of_elements() != 0) {
                  Vertex v = to_visit.pop();
                  if (v not in visited) then {
                        visited.add(v);
                        for all Vertex x in G.neighbors_of(v) {
                              to_visit.push(x);
                        }
                  }
      }
}
```

b) Přidáním řádku „`Stack to_visit = empty;`" za řádek 1),
řádku „`to_visit.push(start_vertex);`" za řádek 3),
řádku „`Vertex v = to_visit.pop();`" za řádek 4 a
řádku „`to_visit.push(x);`" za řádek 7)
vznikne následující procedura, která prohledává graf do šířky.

```
Vertices visited = empty;
Stack to_visit = empty;
procedure search(Graph G, Vertex start_vertex )
{
            to_visit.push(start_vertex);
            while (to_visit.number_of_elements() != 0) {
                  Vertex v = to_visit.pop();
                  if (v not in visited) then {
                        visited.add(v);
                        for all Vertex x in G.neighbors_of(v) {
                              to_visit.push(x);
                        }
                  }
      }
}
```

c) Přidáním řádku „`search(G,x);`" za řádek 7) a
smazáním řádků 4) a 9) vznikne následující procedura, která prohledává graf do šířky.

```
Vertices visited = empty;
procedure search(Graph G, Vertex start_vertex )
{
                  if (v not in visited) then {
                        visited.add(v);
                        for all Vertex x in G.neighbors_of(v) {
                              search(G,x);
                        }
                  }
}
```

d) **Přidáním řádku „`int a = 0;`“ za řádek 1),**
řádku „`Priority_Queue to_visit = empty;`“ za řádek 1),
řádku „`to_visit.push(a++, start_vertex);`“ za řádek 3),
řádku „`Vertex v = to_visit.pop();`“ za řádek 4 a
řádku „`to_visit.push(x);`“ za řádek 7)
vznikne následující procedura, která prohledává celý graf do šířky.
Poznámka: Metoda push má v tomto případě dva argumenty. První je hodnota klíče a druhý jsou data. Metoda pop vrací vždy ty data, která byla vložena do prioritní fronty s nejnižší hodnotou klíče. Po provedení operace pop je tento klíč z prioritní fronty odstraněn.

```
Vertices visited = empty;
int a = 0;
Priority_Queue to_visit = empty;
procedure search(Graph G, Vertex start_vertex )
{
            to_visit.push(a++,start_vertex);
            while (to_visit.number_of_elements() != 0) {
                  Vertex v = to_visit.pop();
                  if (v not in visited) then {
                        visited.add(v);
                        for all Vertex x in G.neighbors_of(v) {
                              to_visit.push(x);
                        }
                  }
      }
}
```

e) Přidáním řádku „`int a = 0;`“ za řádek 1),
řádku „`Priority_Queue to_visit = empty;`“ za řádek 1),
řádku „`to_visit.push(a--, start_vertex);`“ za řádek 3),
řádku „`Vertex v = to_visit.pop();`“ za řádek 4 a
řádku „`to_visit.push(x);`“ za řádek 7)
vznikne procedura, která prohledává celý graf do hloubky.
Poznámka: Metoda push má v tomto případě dva argumenty. První je hodnota klíče a druhý jsou data. Metoda pop vrací vždy ty data, která byla vložena do prioritní fronty s nejnižší hodnotou klíče. Po provedení operace pop je tento klíč z prioritní fronty odstraněn.

```
Vertices visited = empty;
int a = 0;
Priority_Queue to_visit = empty;
procedure search(Graph G, Vertex start_vertex )
{
            to_visit.push(a--,start_vertex);
            while (to_visit.number_of_elements() != 0) {
                  Vertex v = to_visit.pop();
                  if (v not in visited) then {
                        visited.add(v);
                        for all Vertex x in G.neighbors_of(v) {
                              to_visit.push(x);
                        }
                  }
      }
}
```