B

Jméno ____________________________________

V následujících kvízových otázkách je správně někdy jen jedna, někdy více variant odpovědí. Správně zodpovězená otázka je taková, která určí správně všechny varianty odpovědí a je hodnocena uvedeným počtem bodů. Pokud některá varianta je určena chybně, otázka je hodnocena 0 body. V otázkách s tvořenou odpovědí je pouze správná odpověď hodnocena uvedeným počtem bodů. Než tvořenou odpověď vepíšete do archu, připravte si ji dobře nanečisto na jiném papíru.

1. (5 b.)

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.

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

2. (5 b.)

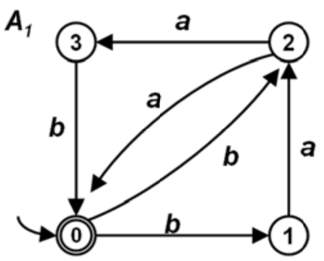
a) obsahuje ε-přechody
b) *A* vyhledá řetězec *babab*
c) *A* má méně než 14 stavů
d) má alespoň 5 koncových stavů
e) *A* vyhledá také některé řetězce, které mají od P Levenshteinovu vzdálenost rovnu nejvýše 2

Pro vzorek *P* = *bbbcc* nad abecedou {*a*, *b*, *c*} je standardním postupem sestrojen nedeterministický automat *A*, pomocí nějž lze v textu vyhledávat řetězce, které mají od *P* Hammingovu vzdálenost rovnu nejvýše 2. Pro *A* platí, že

3. (5 b.)

Je dán NKA A_1 pomocí přechodového diagramu na obrázku. Dále je na obrázku dán pomocí tabulky přechodů DKA A_2, který je s A_1 ekvivalentní a vznikl použitím standardního algoritmu převodu NKA na DKA. Žádné další úpravy se s A_2 neprováděly. V tabulce je jeden řádek poškozen a není čitelný. Určete, která tvrzení platí pro poškozený řádek.

a) Označení řádku je 02.
b) Označení řádku je 013.
c) Údaj ve sloupci *a* je prázdný.
d) Řádek představuje koncový stav.
e) Údaj ve sloupci *b* odkazuje na stav 023.



A_2

	a	b	
0		12	F
12	023		
023	03	012	F
poškozeno			
012	023	12	F

4. (5 b.)

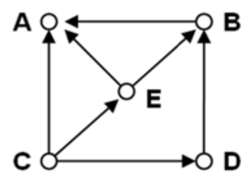
a) *B* může mít více stavů než *A*
b) *B* je deterministický
c) *B* může mít více koncových stavů než *A*
d) diagram *B* je vždy acyklický graf
e) *B* i *A* mají stejný počet koncových stavů

K danému nedeterministickému konečnému automatu A, který obsahuje alespoň jeden ε–přechod) konstruujeme pomocí standardní konstrukce k němu ekvivalentní automat *B* bez ε–přechodů. Platí:

5. (5 b.)

a) A ≤ B ≤ E ≤ C ≤ D
b) C ≤ A ≤ E ≤ D ≤ B
c) C ≤ D ≤ E ≤ B ≤ A
d) C ≤ E ≤ D ≤ B ≤ A
e) C ≤ E ≤ B ≤ A ≤ D

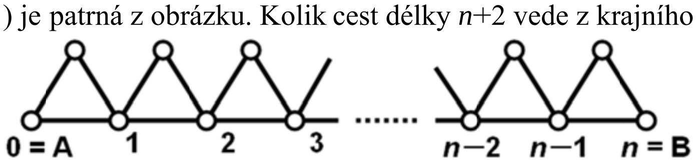
V topologickém uspořádání grafu na obrázku platí pro jednotlivé vrcholy (tranzitivní) relace



6. (10 b.)

a) n
b) $n-1$
c) $n(n-1)/2$
d) $n(n+1)/2$
e) $(n-1)^2$

Struktura grafu H_n ($n \geq 1$) je patrná z obrázku. Kolik cest délky $n+2$ vede z krajního uzlu *A* do krajního uzlu *B*?



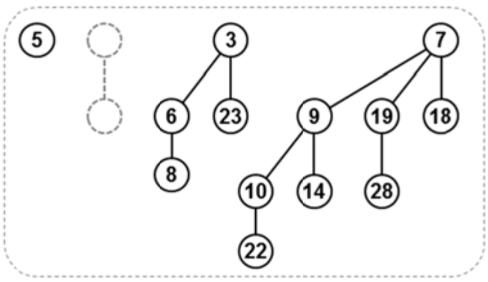
B **Jméno** ______________________________

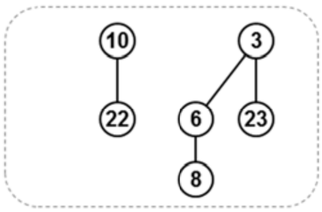
7. (5 b.)
V jaké třídě složitosti je funkce `f` v následujícím programu vzhledem k velikosti pole `p` (budeme ji značit n), když víme, že v proměnné N je vždy velikost pole `p`?

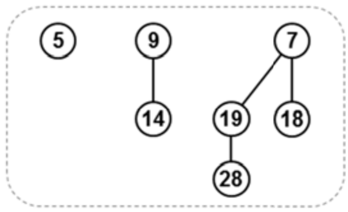
a) $O(1)$
b) $O(n \cdot \log(n))$
c) $O(n^2)$
d) $\Omega(1)$
e) $\Omega(\log(n))$
f) $\Omega(n^2)$
g) $\Theta(n^2)$
h) $\Theta(1)$

```
int f (int p[], int N, int e) {
      int i,s,o;
      for( i=N>>1, s=i, o=N<<1;
            (o!=0) && (i<=N);
            (p[i]>e)?(i-=s):(i+=s), o>>1 ) {
               s = (s==1)?1:(s>>1);
               if (p[i] == e) return i;
      }
      return -1;
}
```

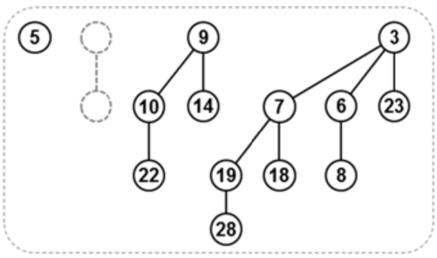
8. (10 b.)
Když spojíme pomocí operace Merge dvě binomiální haldy na obrázku získáme jedinou výslednou binomiální haldu. Nakreslete ji.

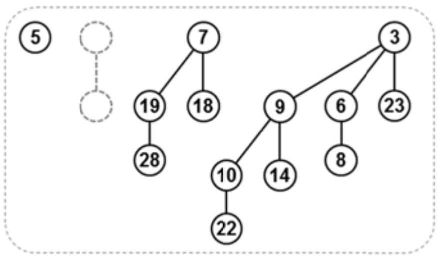






Další dvě možné varianty řešení, stačí kterákoli z uvedených tří:





9. **(5 b.)**

$\boldsymbol{T}(\boldsymbol{x}) =$

$$\begin{pmatrix} 1/4 & 0 \\ 0 & 1/4 \end{pmatrix} \boldsymbol{x} + \begin{pmatrix} 1/4 \\ 1/2 \end{pmatrix}$$

Určete transformaci roviny, která je složená z následujících zobrazení v uvedeném pořadí. Nejprve je vzor $\boldsymbol{x}$ posunut o jednotku doprava (v kladném směru osy x) a o dvě jednotky nahoru (v kladném směru osy y). Potom je výsledek zmenšen na čtvrtinovou velikost. Transformaci zapište ve tvaru $\boldsymbol{T}(\boldsymbol{x}) = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{z}$, kde A je matice a $\boldsymbol{z}$ je sloupcový vektor.

10. (5 b.)

a) $\Theta(n)$
b) $\Omega(n^2)$
c) $\Theta(n^2)$
d) $O(n^2 \cdot \log(n))$
e) $O(n \cdot \log(n))$

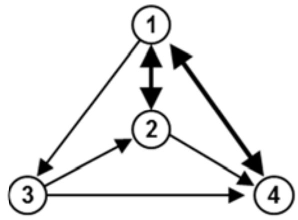
Do binární haldy obsahující $n^{1.5}$ prvků, jejíž kořen obsahuje nejmenší hodnotu z celé haldy, přidáme n prvků. Asymptotická složitost této akce je

11. (10 b.)
Najděte přechodovou matici pro návštěvnost webu se čtyřmi adresami daného obrázkem. Šipky znázorňují možnosti přechodů mezi jednotlivými adresami. Najděte přechodovou matici $\boldsymbol{P}$ pro algoritmus Page Rank pro tento web (5 b.). Určete sloupcový $\boldsymbol{w}$ vektor popisující stacionární režim přechodové matice tohoto webu (5 b.).

B

Jméno ______________________________

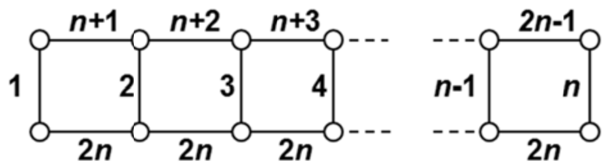
Matice *P*:	vektor *w*:
$\begin{pmatrix} 0 & 1/2 & 0 & 1 \\ 1/3 & 0 & 1/2 & 0 \\ 1/3 & 0 & 0 & 0 \\ 1/3 & 1/2 & 1/2 & 0 \end{pmatrix}$	$\begin{pmatrix} 12/31 \\ 6/31 \\ 4/31 \\ 9/31 \end{pmatrix}$



12. (5 b.)
V Kruskalově algoritmu reprezentujeme datovou strukturu Union-Find pomocí orientovaných stromů. Předpokládáme, že při sjednocování stromů zařazujeme vždy menší strom pod kořen většího a nepoužíváme žádné další heuristiky pro zvýšení efektivity. Určete, jaká je hloubka konečného stromu v této struktuře po nalezení minimální kostry daného daného (kořen má hloubku 0), a nakreslete strom pro $n = 5$.

Hloubka pro obecné $n \geq 3$:
2

Strom:

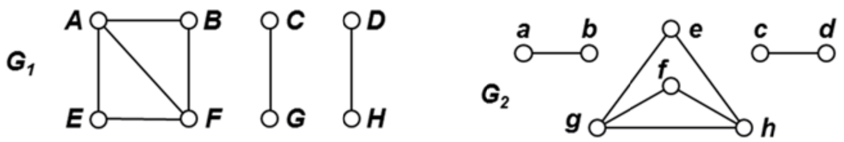


13. (5 b.)

Počet izomorfizmů:

32

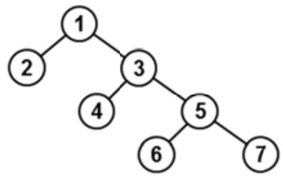
Určete počet izomorfizmů mezi danými dvěma grafy. Pozor, nejsou souvislé.



14. (5 b.)

Černé: 1 2 4 5
Červené: 3 6 7

Napište, jak je nutno obarvit uzly daného stromu, pokud to má být červenočerný strom.



15. (15 b.)
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.

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

a) .	b)	c)	d) .	e) .

Rozhodněte, která z následujících tvrzení platí:

B

Jméno ______________________________

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);
 }
}
```

B

Jméno ______________________________

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(a++,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(a++,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(a++,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(a++,x);
    }
  }
}
```