Dátové štruktúry a algoritmy (ZS 2016/2017)

Skúška – 16. 1. 2017 (Variant B – Riešenie)

MENO A PRIEZVISKO (PALIČKOVÝM PÍSMOM):

MIESTNOSŤ (zaškrtni):

U120	U80	Veľká aula	CPU

Vypíšte sľub a podpíšte ho:
„Sľubujem na svoju česť, že som túto písomku vypracoval(a) samostatne.“

Podpis:

Inštrukcie

1. Všetko odložte z lavice.
2. Na lavicu si vyberte jeden preukaz s fotografiou!
   (študentský preukaz, občiansky preukaz, cestovný pas)
3. Na lavicu si pripravte písacie potreby!
   (najlepšie aspoň TRI perá)
4. Pred sebou máte obojstranne vytlačené zadanie skúšky,
   do ktorého vypĺňate odpovede, a ktoré po skončení odovzdáte.
   Dostanete papier na pracovné výpočty, ktorý sa neodovzdáva.
   Vlastné papiere nie sú povolené! Nepíšte za okraj.

   Skúška obsahuje 14 úloh, za ktoré môžete získať najviac 35 bodov.
   (povinné minimum 15 bodov)
5. Počkajte na zahájenie skúšky.
   Skúška trvá 180 minút.

Veľa úspechov!

A (max. 2b): Uvažujte úlohu vyhľadania prvku podľa kľúča v spájanom zozname. Uveďte (nejaké) dolné ohraničenie zložitosti tohto problému a zdôvodnite ho.

Odpoveď: Hľadaný prvok sa môže nachádzať na ľubovoľnom mieste v zozname, je preto potrebné skontrolovať každý prvok zoznamu. Dolné ohraničenie teda je O(N), pretože každý algoritmus riešiaci túto úlohu musí prejsť každý prvok v zozname: vykonať aspoň O(N) krokov, kde N je počet prvkov zoznamu.

B (max. 1b): Zistite, či dĺžka behu (časová zložitosť) usporadúvania vkladaním (Insertion Sort) závisí od vstupnej postupnosti. Odpoveď zdôvodnite.

Odpoveď: Časová zložitosť usporadúvania vkladaním závisí od vstupnej postupnosti. Ak je vstupná postupnosť usporiadaná, tak pri vkladaní po jednom prvku nemusíme robiť výmeny, zložitosť je preto O(N). Ak je postupnosť neusporiadaná, resp. opačne usporiadaná, tak vloženie každého prvku znamená presunúť prvok na začiatok, a teda zložitosť $O(N^2)$

C (max. 2b): Uvažujte max haldu nezáporných celých čísel so štandardnými operáciami (insert a extractMax), ktorá je implementovaná ako úplný binárny strom reprezentovaný vektorom. Implementujte novú operáciu extractSecondMax, ktorá vráti a odstráni hodnotu druhého najväčšieho prvku v halde, ak takýto prvok existuje. Ak neexistuje, funkcia vráti -1. Zložitosť by mala byť O(log N), kde N je počet prvkov v halde.

Odpoveď: Nech je halda reprezentovaná prvkami heap[1], … heap[n]. Výsledok (druhý najväčší prvok) je väčší z prvkov heap[2] a heap[3] (ak existujú). Odstránene tohto prvku prebieha rovnako ako odstránenie najväčšieho prvku (heap[1]): presunúť posledný prvok heap[n] do heap[2] a (ak je to potrebné) posunúť ho smerom dole pre obnovenie haldovej vlastnosti.

D (max. 2b): Daná je neusporiadaná postupnosť N celých čísel, pričom každý prvok je najviac K pozícií vzdialený od cieľovej pozície v usporiadanom poradí. Navrhnite algoritmus, ktorý usporiada vstupnú postupnosť v čase O(N log K). Stačí slovný opis hlavnej myšlienky alebo pseudokód.

Odpoveď: Je možné riešiť viacerými spôsobmi. Napr. využitím haldy: vložiť prvých K prvkov do haldy, a potom opakovane (N-K krát) vložiť ďalší prvok z poľa do haldy a vybrať minimum z haldy do poľa. Nakoniec vybrať zostávajúcich K prvkov z haldy od najmenšieho a uložiť ich do poľa.

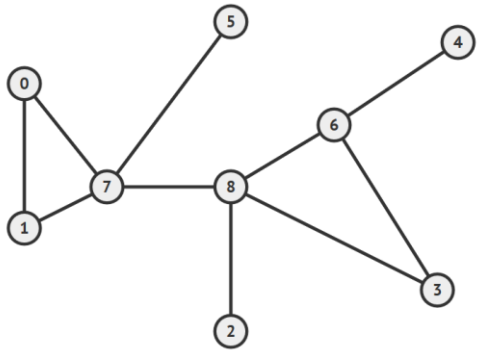
Iné riešenie: Rozdeľ pole na súvislé bloky veľkosti K čísel (blokov bude N/K), každý usporiadaj v čase O(K log K) a nakoniec využitím haldy spoj týchto K blokov v čase O(N log K) – z každého bloku bude v halde vždy najviac jeden prvok; po odstránení z haldy tam opäť pridáme prvok (ak existuje) z rovnakého bloku.

E (max. 2b): Stručne porovnaj AVL strom so Splay stromom. Podrobne vysvetli scenár vyhľadávania, v ktorom je výhodné použiť Splay strom namiesto AVL stromu.

Odpoveď: Štruktúra AVL stromu sa upravuje len pri vkladaní a vyberaní. Splay strom sa upravuje aj pri vyhľadaní a preto sa dokáže prispôsobiť (zrýchliť) aj pri vykonávaní vyhľadaní.

Teda napriek tomu, že AVL strom je vyvážený a najhoršia zložitosť vykonania operácií nad ním je O(log N), tak v scenároch, kedy opakovane vyhľadávame malú množinu prvkov, ktoré sú akurát v najväčšej hĺbke v AVL strome je použitie Splay stromu výhodnejšie. Pri vyhľadaní v AVL strome musíme vždy prejsť celú hĺbku stromu. Naopak v prípade Splay stromu sa pri prvom vyhľadaní prvok dostane do koreňa, a ďalšie vyhľadania už prebehnú v konštantnom čase O(1).

F (max. 2b): Daný je graf na obrázku. Určte nasledovné vlastnosti tohto grafu (pri prehľadávaní uvažujte, že susedné vrcholy sa navštevujú postupne najskôr od menších čísel).



a) Určite poradie navštevovania vrcholov pri prehľadávaní do hĺbky z vrcholu 8:
8, 2, 3, 6, 4, 7, 0, 1, 5

b) Určite poradie navštevovania vrcholov pri prehľadávaní do šírky z vrcholu 8:
8, 2, 3, 6, 7, 4, 0, 1, 5

c) Určite artikulácie:
6, 7, 8

d) Určite mosty:
2-8, 4-6, 5-7, 7-8

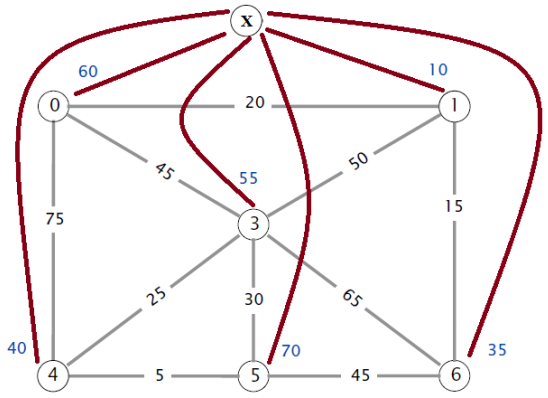
G (max 3b): Bežný Dijkstrov algoritmus nájde najkratšiu u-v cestu v orientovanom ohodnotenom grafe. Uvažujte teraz postupnosť všetkých u-v ciest usporiadaných podľa dĺžky od najkratších. Navrhnite modifikáciu Dijkstrovho algoritmu pre nájdenie druhej najkratšej u-v cesty. (Predpokladajte, že dĺžka druhej najkratšej u-v cesty je väčšia ako dĺžka najkratšej u-v cesty.) Stručne opíšte myšlienku algoritmu a zdôvodnite jeho správnosť. Hlavnú slučku algoritmu implementujte v pseudokóde.

Odpoveď: Pre každý vrchol v si okrem dĺžky najkratšej cesty d[v] z počiatočného vrcholu do v budeme pamätať aj dĺžku druhej najkratšej cesty d2[v]. Zmena oproti štandardnému Dijkstrovmu algoritmu je pri relaxácii hrán: pred prepísaním d[v] na menšiu hodnotu si predchádzajúcu hodnotu d[v] odložíme do d2[v]. Pre každý vrchol teraz ale musíme (definitívne) určiť dve hodnoty: d[v] a d2[v], a preto každý vrchol v môže byť až v troch možných stavoch: 0) ani jedna z hodnôt d[v] a d2[v] ešte nie je definitívna, 1) hodnota d[v] je definitívna ale hodnota d2[v] ešte nie je, 2) obe hodnoty d[v] a d2[v] sú definitívne.

Pseudokód hlavnej slučky: **while (stav[ciel] < 2) {**
min = vrchol pre ktorý stav[min]=0 a d[min] = min{d[v] : pre v, ktorého stav[v] == 0}
min2 = vrchol pre ktorý stav[min2]=1 a d2[min2] = min{d2[v] : pre v, ktorého stav[v] == 1}
ak d[min] <= d2[min2] stav[min]=1 a z vrchola min relaxujeme dĺžky najkratších ciest (d) do susedov (so zapamätaním starej d[v] do d2[v])
ak d[min] > d2[min2] stav[min2] = 2 a z vrchola min2 relaxujeme len dĺžky druhých najkratších ciest d2
} return d2[ciel];

H (max. 3b): Do zaostalej oblasti sa rozhodli zaviesť plyn. V oblasti je N dedín a pre každú vieme cenu pripojenia na centrálny rozvod $x_i > 0$. Poznáme tiež (graf) ceny $w_{ij} > 0$ prepojenia i-tej dediny s j-tou dedinou. Plyn je do nejakej dediny zavedený vtedy, ak je alebo pripojená na centrálny rozvod alebo prepojená s inou dedinou do ktorej je zavedený plyn. Plyn chceme zaviesť do každej dediny tak, aby celková cena bola čo najnižšia. Navrhnite graf, ktorý reprezentuje tento problém ako úlohu o najlacnejšej kostre. Opíšte hlavnú myšlienku a vyznačte riešenie pre ukážkový graf prepojení na obrázku nižšie.

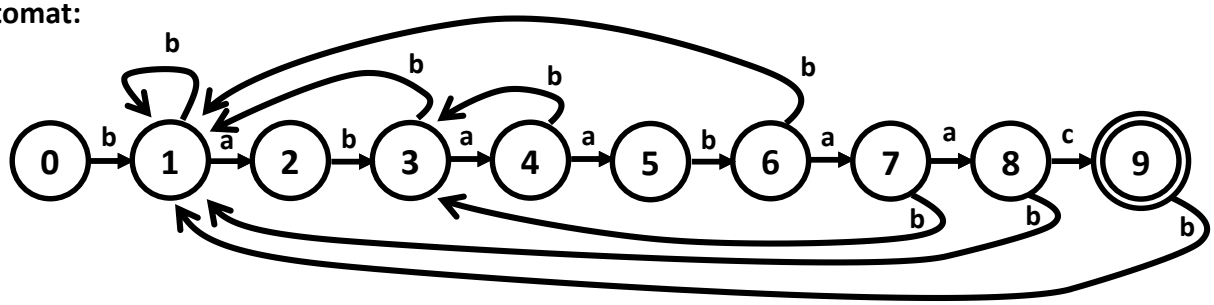
Odpoveď: Oproti štandardnej úlohe o najlacnejšej kostre máme v tomto prípade aj vrcholové ohodnotenia. Graf prepojení rozšírime o jeden vrchol X, ktorý bude predstavovať centrálny rozvod. Pridáme hrany z každej dediny do X, cena bude zodpovedať pripojeniu príslušnej dediny na centrálny rozvod. Vo výslednom grafe nájdeme najlacnejšiu kostru. Nájdená kostra je strom a preto každá dedina bude pripojená na plyn alebo priamo na vrchol X alebo na inú dedinu.



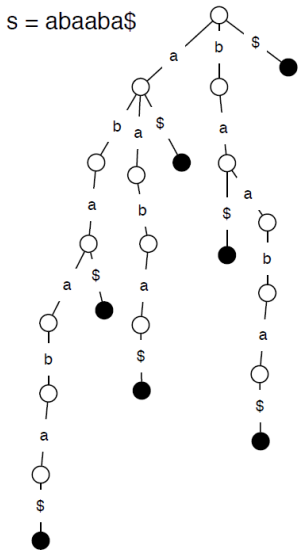
I (max. 3b): Doplňte hodnoty funkcie π v algoritme KMP pre vzorku P="babaabaac" do tabuľky nižšie a zostrojte príslušný deterministický konečný automat, ktorý akceptuje P.

i	1	2	3	4	5	6	7	8	9
P[i]	b	a	b	a	a	b	a	a	c
π[i]	**0**	**0**	**1**	**2**	**0**	**1**	**2**	**0**	**0**

Automat:



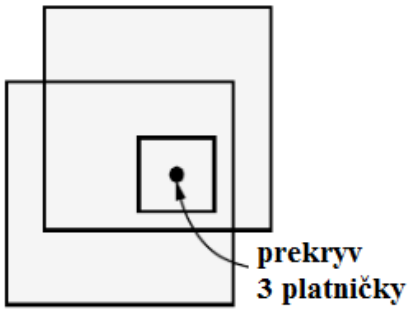
J (max. 2b): Daný je sufixový strom pre text s. Ukážka je na obrázku vpravo. Navrhnite algoritmus, ktorý (v ľubovoľnom sufixovom strome) určí počet všetkých výskytov reťazca t v reťazci s. Stačí slovný opis hlavnej myšlienky.



Odpoveď: Prehľadávame sufixový strom od koreňa, postupujúc po písmenkách reťazca t, až prídeme do vrcholu x. Počet listov podstromu s koreňom x zodpovedá počtu výskytov t v s.

Ak si v každom vrchole sufixového stromu pamätáme počet listov podstromu, tak výpočtová zložitosť navrhnutého algoritmu je lineárna vzhľadom na dĺžku t: O(|t|)

K (max. 4b): Stroj pokladá obdĺžnikové platničky v 2D rovine. Každá platnička je položená rovnobežne s osami roviny. Niektoré platničky sa prekrývajú, hrúbku platničky zanedbajte, dôležitý je počet platničiek, ktoré sa v nejakom bode môžu prekrývať. Dané sú súradnice a rozmery N platničiek, navrhnite algoritmus v čase $O(N^2)$, ktorý zistí najväčší počet platničiek, ktoré sa v nejakom bode prekrývajú. Opíšte myšlienku algoritmu a zdôvodnite jeho správnosť. Hlavnú slučku algoritmu implementujte v pseudokóde. V ukážke na obrázku je najväčší počet prekrývajúcich sa platničiek (3) vo vyznačenom bode.



Odpoveď: Úlohu budeme riešiť zametaním po x-ovej osi zľava doprava. Budeme si priebežne udržiavať „aktívnu množinu" platničiek, ktoré pretína zametacia čiara. Pre každú platničku uvažujeme (dve) hraničné x-ové súradnice, ktoré predstavujú udalosti na spracovanie. Ľavý okraj znamená pridanie platničky do aktívnej množiny, pravý okraj znamená odstránenie.

Aktívna množina predstavuje zvislý (vertikálny) pás v rámci ktorého už nie sú iné udalosti. Platničky sa v tomto páse redukujú na jednorozmerné (1D) intervaly (na y-ovej osi) a zaujíma nás koľko najviac 1D intervalov sa prekrýva. Intervaly niekde začínajú a niekde končia. Medzi týmito hranicami sa počet pokrývajúcich intervalov nemení, tieto úseky nazveme bunky. Počas celého spracovania budeme mať najviac 2N takýchto buniek (lebo je 2N hraničných y-ových súradníc platničiek): pre každú bunku budeme mať počítadlo koľko intervalov z aktívnej množiny ho pokrýva. Každý interval (platnička) pokrýva najviac všetky tieto bunky, takže pridanie/odobratie jednej platničky zrealizujeme ako pripočítanie/odpočítanie 1 pre najviac 2N buniek: preto spracovanie jednej platničky trvá čas O(N). Priebežne ako budeme posúvať zametaciu čiaru, tak musíme pridať a odobrať N platničiek, celkový čas je teda $O(N^2)$.

Pseudokód hlavnej slučky:
```
int c[2N]={0}, result=0; // počítadlo prekryvov buniek, celkové maximum
Event e[2N] sú usporiadané udalosti (x-ové súradnice okrajov platničiek)
for (i = 0; i < 2*n; i++) {
  if (e[i] is lavyOkraj) { for (i = e[i].platna.y1; i < e[i].platna.y2; i++) c[i]++; /* pridanie y1-y2 */ }
  if (e[i] is pravyOkraj) { for (i = e[i].platna.y1; i < e[i].platna.y2; i++) c[i]--; /* odobratie y1-y2*/ }
  if (result < max{c}) result = max{c} // najväčší prekryv v aktuálnom páse je väčší ako doterajší
} return result;
```

L (max. 2b): Uvažujme hru dvoch hráčov s figúrkami v rade. Hráči ťahajú striedavo. Hráč môže odobrať z radu vždy len párny počet figúrok. Hráč, ktorý nemôže ťahať prehráva.

Určite Grundyho čísla pre túto hru do tabuľky nižšie (N je počet figúrok v rade):

N	0	1	2	3	4	5	6	7	8	9	10	11
g(N)	**0**	**0**	**1**	**1**	**2**	**2**	**3**	**3**	**4**	**4**	**5**	**5**

Uvažujte túto hru hranú s tromi radmi, pričom v jednom ťahu môže hráč odobrať figúrky práve z jedného radu. Ak nemá ťah v žiadnom rade, tak prehráva. Dané sú počty figúrok v týchto troch radoch: A_1, A_2, A_3. Navrhnite algoritmus, ktorý v čase O(1) určí či hráč na ťahu môže vyhrať, ak bude ťahať optimálne.

Odpoveď (stačí veľmi stručný opis hlavnej myšlienky resp. pseudokód):

Z tabuľky vyššie je zrejmý všeobecný výpočet Grundyho čísel pre ľubovoľný počet figúrok: $g(x) = \left\lfloor \frac{x}{2} \right\rfloor$

Preto Grundy číslo hry pre tri rady bude $g(A_1,A_2,A_3) = g(A_1) \oplus g(A_2) \oplus g(A_3) = \left\lfloor \frac{A_1}{2} \right\rfloor \oplus \left\lfloor \frac{A_2}{2} \right\rfloor \oplus \left\lfloor \frac{A_3}{2} \right\rfloor$.

Hráč na ťahu môže vyhrať vtedy, ak hodnota $g(A_1,A_2,A_3) > 0$, čo znamená, že existuje ťah, ktorý dostane protihráča do pozície s Grundyho číslom 0 (prehrávajúca). Ak $g(A_1,A_2,A_3) = 0$, hráč na ťahu prehrá (ak protihráč ťahá optimálne).

M (max. 4b): Pod stromčekom ste si našli N darčekov. Každý darček je škatuľa s podstavou a_i x b_i a výškou c_i. Rozmýšlate ako z nich postaviť čo najvyššiu vežu. Darčeky musia stáť vždy na svojej podstave, a darček X môžete položiť na darček Y vtedy, ak podstava X je v oboch rozmeroch striktne menšia ako podstava darčeku Y. Navrhnite algoritmus, ktorý v čase $O(N^2)$ určí výšku najvyššej veže, ktorú je možné z darčekov postaviť. Hmotnosť a nosnosť darčekov zanedbajte. Úlohu riešte dynamickým programovaním podľa zaužívanej schémy nižšie.

Opis hlavnej myšlienky:
Zostrojíme graf, v ktorom vrcholy budú darčeky, a hrana x→y bude v grafe práve vtedy keď darček x môžeme postaviť na darček y. Graf bude acyklický a umožňuje nám nad ním vykonať dynamické programovanie zaužívaným spôsobom. Podstavu každého darčeka „otočíme“ tak, aby $a_i <= b_i$.

Definujte podproblémy:
Označme H_i výšku najvyššej veže, v ktorej darček i je na vrchu.

Určte rekurentný vzťah medzi podproblémami:
$H_i = \max\{H_j + c_i\}$ pre také j, že existuje hrana (i→j), teda platí $a_j < a_i$ a $b_j<b_i$ resp. inak povedané darček i možeme postaviť na darček j.

Základné prípady:
Ak z vrchola (darčeku) i neexistujú výstupné hrany, tak $H_i = c_i$

N (max. 3b): Daný je graf s N vrcholmi. Úloha nájsť v grafe G úplný podgraf (angl. clique) s aspoň K vrcholmi je NP-úplná. Uvažujme teraz jednoduchšiu úlohu: chceme „len“ zistiť, či sa v grafe G nachádza alebo nie takýto úplný podgraf s aspoň K vrcholmi. Nezaujíma nás teda nájsť konkrétny úplný podgraf, len by sme chceli navrhnúť rýchly algoritmus, ktorý rozhodne (áno/nie) či sa taký úplný podgraf s aspoň K vrcholmi v grafe G nachádza alebo nie. Zistite a stručne zdôvodnite, či je táto „jednoduchšia“ úloha v P alebo v NP.

Odpoveď: Vyriešime redukciou. Predpokladajme, že by sme pre K a graf G vedeli rozhodnúť (áno/nie), či graf G obsahuje úplný podgraf s aspoň K vrcholmi v polynomiálnom čase. Ukážeme, že by sme potom vedeli v grafe G nájsť úplný podgraf s aspoň K vrcholmi v polynomiálnom čase, čo ale nie je (so súčasnými vedeckými poznatkami) možné, a teda nedokážeme vyriešiť túto „jednoduchšiu“ úlohu v polynomiálnom čase, a preto je v NP.

Ako nájdeme konkrétny úplný podgraf? Budeme postupne skúšať každý vrchol x: ak graf G bez vrcholu x (G-{x}) stále obsahuje úplný podgraf s aspoň K vrcholmi, tak x môžeme z G odstrániť a pokračujeme s menším grafom. Tento jeden krok vieme spraviť v polynomiálnom čase (podľa predpokladu). Takto prejdeme všetky vrcholy (N), niektoré odstránime (lebo neovplyvnia existenciu úplného podgrafu) a niektoré ponecháme. Nakoniec zostane z grafu len hľadaný úplný podgraf G s aspoň K vrcholmi.