CVIČENÍ 7

Téma: Vlastnosti stromů, algoritmy na stromech.
Cíle: Zvládnout vzájemnou provázanost tří hlavních vlastností stromů (graf, který je: souvislý, bez kružnic a má o jednu hranu méně než uzlů), dokázat vhodně upravit základní grafové algoritmy, pokud se mají týkat stromů.

Stromy	
1.	Kolik může mít maximálně, resp. minimálně listů neorientovaný strom s celkem n (≥2) uzly?
2.	Nechť **T** je neorientovaný strom, v němž je maximální stupeň uzlu roven δ_{max}(≥2). Ukažte, že strom **T** má nejméně δ_{max} listů (tj. uzlů stupně 1).
3.	Nechť **T** je neorientovaný strom mající přesně **k** listů, jehož všechny vnitřní uzly mají stejný stupeň **d**(≥3). Určete počet vnitřních uzlů stromu **T** v závislosti na **k** a **d**.
4.	Navrhněte algoritmus, který pro zadaný obyčejný neorientovaný graf G = ⟨H,U⟩ zjistí v čase O(|U|), zda graf G obsahuje nějakou kružnici.
5.	Neorientovaný strom **S** je zadaný maticí sousednosti **V**. Sestavte algoritmus výpočtu jeho poloměru, průměru a středů z této matice. Určete složitost tohoto algoritmu. Změňte reprezentaci stromu na pole ukazatelů na seznamy sousedů a navrhněte skutečně efektivní algoritmus (podobný jako při testování acykličnosti, ale pozor na to, že listy se vždy musí odebrat "naráz").
6.	Nechť $\mathbf{T_1}$ a $\mathbf{T_2}$ jsou dvě hranově disjunktní kostry (souvislého) neorientovaného grafu **G** s **n** uzly. Je možné určit minimální počet kružnic, které vzniknou sjednocením $\mathbf{T_1} \cup \mathbf{T_2}$?
7.	Nechť $\mathbf{T_1}$, resp. $\mathbf{T_2}$ je strom prohledání souvislého neorientovaného grafu **G** do šířky, resp. do hloubky. Je možné z nějakých vlastností stromů $\mathbf{T_1}$ a $\mathbf{T_2}$ usuzovat něco o poloměru **r(G)** a průměru **T(G)** grafu **G**?
8.	V neorientovaném stromu o n uzlech provedeme náhodně orientaci hran. Jaká je pravděpodobnost, že vznikne kořenový strom?
9.	V úplném neorientovaném grafu provedeme náhodně orientaci hran. • Dokažte, že vzniklý graf má kořenovou kostru. • Pokud výchozí graf není úplný, lze naopak nalézt taková orientace, že výsledný orientovaný graf nemá kořenovou kostru (určete jak).
10.	Potvrďte nebo vyvraťte následující tvrzení: Při procházení binárního stromu libovolným ze způsobů preorder, inorder, postorder se listy stromu projdou vždy ve stejném pořadí.
11.	Nalezněte obecný tvar binárních stromů, jejichž uzly se projdou ve stejném pořadí při průchodech • preorder a inorder • preorder a postorder • inorder a postorder
12.	Nechť **T** je kořenový strom s uzly očíslovanými **1, 2, ..., n**, který je reprezentován pouze pomocí odkazů **pred[u]** každého uzlu na svého předchůdce ve stromu, pro kořen **r** je **pred[r]=0**. Navrhněte algoritmus systematického průchodu stromem **T** s časovou složitostí **O(|U|)**, jehož cílem je

	• určit hloubku (tj. vzdálenost od kořene) všech uzlů • určit celkovou hloubku stromu **T** • určit maximální šířku stromu **T**

CVIČENÍ 7 – možná řešení

Téma: Vlastnosti stromů, algoritmy na stromech.
Cíle: Zvládnout vzájemnou provázanost tří hlavních vlastností stromů (graf, který je: souvislý, bez kružnic a má o jednu hranu méně než uzlů), dokázat vhodně upravit základní grafové algoritmy, pokud se mají týkat stromů.

Stromy	
1.	Kolik může mít maximálně, resp. minimálně listů neorientovaný strom s celkem n (≥2) uzly? n-1 (hvězdice), resp. 2 (cesta)
2.	Nechť **T** je neorientovaný strom, v němž je maximální stupeň uzlu roven δ_{max}(≥2). Ukažte, že strom **T** má nejméně δ_{max} listů (tj. uzlů stupně 1). **Intuitivně**: s uzlem u_{max} o maximálním stupni inciduje celkem δ_{max} hran, každou takovou hranou začíná cesta končící v nějakém listu, tyto cesty nemají (kromě výchozího uzlu nic společného).
3.	Nechť **T** je neorientovaný strom mající přesně **k** listů, jehož všechny vnitřní uzly mají stejný stupeň **d**(≥3). Určete počet vnitřních uzlů stromu **T** v závislosti na **k** a **d**. Platí $\sum\delta(u) = 2.\|H\|$, pro strom tedy $\sum\delta(u) = 2.(\|U\|-1)$. Označme počet vnitřních uzlů jako n, pak bude $\sum\delta(u) = k + n.d = 2.(\|U\|-1) = 2.((k+n) - 1)$, tedy $k + n.d = 2k + 2n - 2$, čili $n.(d-2) = k - 2$, neboli $n = (k-2) / (d-2)$.
4.	Navrhněte algoritmus, který pro zadaný obyčejný neorientovaný graf $G = \langle H,U \rangle$ zjistí v čase $O(\|U\|)$, zda graf G obsahuje nějakou kružnici. Necháme běžet prohledání do hloubky a zastavíme se při nalezení první zpětné hrany – ta znamená existenci kružnice a muselo to nastat nejpozději při procházení $\|U\|$-té hrany. Pokud se žádná zpětná hrana nenašla, pak má graf nejvýše $\|U\|-1$ hran a složitost DFS je $O(\|H\|+\|U\|) = O(\|U\|)$.
5.	Neorientovaný strom **S** je zadaný maticí sousednosti **V**. Sestavte algoritmus výpočtu jeho poloměru, průměru a středů z této matice. Určete složitost tohoto algoritmu. Změňte reprezentaci stromu na pole ukazatelů na seznamy sousedů a navrhněte skutečně efektivní algoritmus (podobný jako při testování acykličnosti, ale pozor na to, že listy se vždy musí odebrat "naráz"). Projdeme matici **V** a zjišťujeme, které řádky obsahují pouze jednu jedničku (listy), poté dotyčné řádky a sloupce z matice vypustíme. Tento krok opakujeme tak dlouho, až nám zbude pouze jeden nebo dva uzly, počítáme celkový počet kroků k. Pokud zbude jeden uzel, je jediným středem a platí **r(S) = k, T(S) = 2.r(S) = 2.k**. Pokud zbudou dva uzly, jsou oba středy a platí **r(S) = k+1, T(S) = 2.r(S) - 1 = 2.k+1** Kvůli efektivnosti stačí procházet jen polovinu matice V nad diagonálou a místo vypouštění řádků a sloupců přistupovat do matice přes permutační vektor indexů, takže pak stačí redukovat řád matice. Nicméně i tak má algoritmus složitost $O(\|U\|^3)$.
6.	Nechť $\mathbf{T_1}$ a $\mathbf{T_2}$ jsou dvě hranově disjunktní kostry (souvislého) neorientovaného grafu **G** s **n** uzly. Je možné určit minimální počet kružnic, které vzniknou sjednocením $\mathbf{T_1} \cup \mathbf{T_2}$? Všechny hrany kostry $\mathbf{T_2}$ představují vůči kostře $\mathbf{T_1}$ tětivy, takže jejich přidáním

	vznikne určitě **n-1** nezávislých kružnic, jejich kombinací pomocí operace symetrické diference lze případně nalézt kružnice další.
7.	Nechť $\mathbf{T_1}$, resp. $\mathbf{T_2}$ je strom prohledání souvislého neorientovaného grafu **G** do šířky, resp. do hloubky. Je možné z nějakých vlastností stromů $\mathbf{T_1}$ a $\mathbf{T_2}$ usuzovat něco o poloměru **r(G)** a průměru **T(G)** grafu **G**? Z prohledání do šířky – poloměr grafu bude menší nebo roven hloubce $\mathbf{T_1}$, průměr bude větší nebo roven hloubce $\mathbf{T_1}$.
8.	V neorientovaném stromu o *n* uzlech provedeme náhodně orientaci hran. Jaká je pravděpodobnost, že vznikne kořenový strom? Máme $\mathbf{2^{n-1}}$ různých orientací, přitom jen **n** různých kořenových stromů, takže $\mathbf{n / 2^{n-1}}$.
9.	V úplném neorientovaném grafu $\mathbf{K_n}$ provedeme náhodně orientaci hran, výsledek označme jako **K**. • Dokažte, že vzniklý graf **K** má kořenovou kostru. Sporem: Nechť T je maximální kořenový strom (s kořenem **r**) obsažený v **K** a nechť T nezahrnuje všechny uzly původního úplného grafu, takže např. $u \notin T$. Koře r je spojen se všemi ostatními uzly grafu, tedy i s uzlem u. Je-li hrana [r,u] orientována ven z kořene, pak by musel být uzel u součástí stromu T – spor! Je-li orientována opačně, pak zvolíme za nový kořen uzel u a z něj vychází určitě větší kořenový strom než T – spor! • Pokud výchozí graf není úplný, lze naopak nalézt taková orientace, že výsledný orientovaný graf nemá kořenovou kostru (určete jak). Stačí vzít dvojici nesousedních uzlů a orientovat jejich hrany vždy směrem ven – tím vyloučíme existenci kořenové kostry.
10.	Potvrďte nebo vyvraťte následující tvrzení: Při procházení binárního stromu libovolným ze způsobů preorder, inorder, postorder se listy stromu projdou vždy ve stejném pořadí. Silnou indukcí podle výšky stromu ...
11.	Nalezněte obecný tvar binárních stromů, jejichž uzly se projdou ve stejném pořadí při průchodech • preorder a inorder Každý vnitřní uzel má jen pravého potomka. • preorder a postorder Jediný uzel • inorder a postorder Každý vnitřní uzel má jen levého potomka.
12.	Nechť **T** je kořenový strom s uzly očíslovanými **1, 2, ..., n**, který je reprezentován pouze pomocí odkazů **pred[u]** každého uzlu na svého předchůdce ve stromu, pro kořen **r** je **pred[r]=0**. Navrhněte algoritmus systematického průchodu stromem **T** s časovou složitostí **O(|U|)**, jehož cílem je • určit hloubku (tj. vzdálenost od kořene) všech uzlů • určit celkovou hloubku stromu **T** • určit maximální šířku stromu **T** Podle potřeby doplňte využívané datové struktury. Doplníme stav[u] – FRESH | CLOSED, h[u] – hloubka uzlu u, succ[u] – následník, ze kterého jsme přišli do u, s[k] – šířka stromu T ve hloubce k. Základní myšlenka: postupuje se vzhůru stromem po odkazech pred, až kam to jde (kořen nebo už označený uzel), současně se zaznamenává cesta dolů, podle níž se ve druhé fázi doplňují hodnoty hloubky uzlů. Pak už se jen využije

hloubek k získání požadovaných hodnot.

```
01 Foreach (u in U) {stav[u]=FRESH; h[u]=0;}
02 maxS = 0; maxH = 0;
03 Foreach (u in U: stav[u]==FRESH) Označ(u);  // projde uzly vzhůru a určí h[u]
04 Foreach (u in U)  If (maxH < h[u]) {maxH = h[u];}
05 For (k = 1 to maxH) s[k]=0;
06 Foreach (u in U) s[h[u]]++;
07 For (k = 1 to maxH) If (maxS < s[u]) {maxS = s[u];}

08 void Označ (Node u) {
09   stav[u]=CLOSED; succ[u]=0; v=pred[u];
10   while ((v != 0) & (stav[u]==FRESH)) {  // jde po predchudcich vzhuru
11      succ[v]=u; stav[u]=CLOSED; u=v; v=pred[v];
12   }
13   stav[u]=CLOSED;       // uplatni se jen v korenu
14   v = succ[u];
15   while (v != 0) {      // jde po succ smerem dolu a pocita hloubku
16      h[v]=h[u]+1; u=v; v=succ[v];
17   }
18 }
```

Lineární složitost plyne z toho, že cykly na řádcích 1, 4, 5, 6, 7 proběhnou nejvýše |U|-krát, v rámci cyklu while na řádcích 10 - 12 se každá hrana stromu projde jednou proti směru, ve cyklu na řádcích 15 – 17 pak jednou po směru, hran je ale |U|-1.

Elegantnější řešení lze získat pomocí rekurze podobně jako u funkce FIND v dynamických rozkladech. Procedura Označ(u) bude vracet hloubku předchůdce uzlu u a pole succ není zapotřebí.

```
01 Foreach (u in U) {stav[u]=FRESH; h[u]=0;}
02 maxS = 0; maxH = 0;
03 Foreach (u in U: stav[u]==FRESH) h[u]=Označ(u);  // projde uzly a určí h[u]
04 Foreach (u in U)  If (maxH < h[u]) {maxH = h[u];}
05 For (k = 1 to maxH) s[k]=0;
06 Foreach (u in U) s[h[u]]++;
07 For (k = 1 to maxH) If (maxS < s[u]) {maxS = s[u];}

08 void Označ (Node u) {
09   stav[u]=CLOSED; v=pred[u];
10   if (v == 0) return -1;
11   else if (stav[u]==CLOSED) return h[u];
12   else h[v]=Označ(v)+1;
13   return h[v] ;
14 }
```