Cvičení 2

Příklad 1: Použijte master theorem pro odvození časové složitosti rekurzivních algoritmů, u nichž je tato složitost dána následujícími rekurentními vztahy:

a) $T(n) = 3T(n/2) + n$

b) $T(n) = 4T(n/2) + n^2$

c) $T(n) = 5T(n/2) + n^3$

Příklad 2: Popište nějaký polynomiální algoritmus řešící následující problém:

VSTUP: Orientovaný graf $G = (V, E)$ a dvojice vrcholů $s, t \in V$.

VÝSTUP: Nejkratší cesta z vrcholu s do vrcholu t nebo informace, že žádná taková cesta neexistuje.

Poznámka: V grafu může existovat více než jedna nejkratší cesta. Výstupem může být libovolná z těchto nejkratších cest.

Zapište algoritmus řešící tento problém ve formě pseudokódu a analyzujte co nejpřesněji jeho časovou a prostorovou složitost.

Příklad 3: Připomeňme, že ***sled*** v neorientovaném grafu $G = (V, E)$ je posloupnost vrcholů a hran

$$v_0, e_1, v_1, e_2, v_2, \ldots, v_{r-1}, e_r, v_r,$$

kde $v_0, \ldots, v_r \in V$ a $e_1, \ldots, e_r \in E$, přičemž pro každé e_i (kde $1 \leq i \leq r$) platí, že e_i je hrana z vrcholu v_{i-1} do vrcholu v_i. (Jak vrcholy, tak hrany se mohou v této posloupnosti opakovat.)

Tah je speciálním případem sledu, kde se sice mohou opakovat vrcholy, ale nesmí se opakovat hrany. (Tj. hrany $e_1, \ldots, e_r$ v této posloupnosti musí být všechny navzájem různé.)

Tah se nazývá ***Eulerovský*** (resp. ***Eulerův***), jestliže obsahuje všechny hrany grafu G. Eulerovský tah tedy popisuje, jak projít celý graf G jedním tahem tak, abychom každou hranou prošli právě jednou.

Uvažujme následující problém:

NÁZEV: EULER-PATH

VSTUP: Neorientovaný graf $G = (V, E)$.

VÝSTUP: Eulerovský tah procházející všemi hranami grafu G nebo informace, že žádný takový tah neexistuje.

(*Poznámka:* Eulerovských tahů může pro daný graf G existovat více. Výstupem může být libovolný z nich.)

a) Ukažte, že problém EULER-PATH je algoritmicky řešitelný. Jaká je výpočetní složitost vámi navrženého algorimu?

b) Ukažte, že existuje polynomiální algoritmus řešící problém Euler-Path.

Příklad 4: Uvažujme problém ***hledání maximálního toku v síti*** (Max-Flow).
Pojmem ***síť*** se zde rozumí orientovaný graf $G = (V, E)$, kde je každé hraně $(u, v) \in E$ přiřazena její ***kapacita*** $c(u, v)$, což je nezáporné číslo. Pro jednoduchost to můžeme brát tak, že pokud mezi danou dvojicí vrcholů nevede hrana, je to v principu to samé, jako by mezi danými vrcholy vedla hrana s kapacitou 0, tj. pokud $(u, v) \notin E$, tak $c(u, v) = 0$. Formálně tedy můžeme kapacity definovat pomocí funkce $c : V \times V \to \mathbb{R}_{\geq 0}$. Dále jsou v síti vyčleněny dva speciální vrcholy s (***zdroj***) a t (***stok***).
Tok v síti G s kapacitami hran c se zdrojem s a stokem t je funkce $f : V \times V \to \mathbb{R}$ splňující následující dvě podmínky:

- Tok teče jen hranami daného grafu a není překročena kapacita žádné hrany, tj. pro každé $u, v \in V$ platí
$$0 \leq f(u, v) \leq c(u, v)$$
(Pokud $(u, v) \notin E$, tak $c(u, v) = 0$, a nutně tedy platí $f(u, v) = 0$.)
- Pro každý vrchol s výjimkou zdroje a stoku platí, že co do něj vtéká, to z něj vytéká, tj. pro každé $u \in (V - \{s, t\})$ platí
$$\sum_{v \in V} f(v, u) = \sum_{v \in V} f(u, v)$$

Hodnota toku f, označovaná zápisem $|f|$, je dána jako součet toho, co ze zdroje s vytéká, minus to, co do něj vtéká, tj.
$$|f| = \sum_{v \in V} f(s, v) - \sum_{v \in V} f(v, s)$$
Tok f je ***maximální***, jestliže pro jakýkoli jiný tok f' platí $|f| \geq |f'|$.
Problém Max-Flow je definován následovně:

Název: Max-Flow
Vstup: Síť $G = (V, E)$ s kapacitami hran $c : V \times V \to \mathbb{R}_{\geq 0}$, zdrojem s a stokem t (kde $s, t \in V$).
Výstup: Maximální tok f v dané síti.

(*Poznámka:* Maximálních toků může v dané síti existovat více. Výstupem může být libovolný z nich.)
Pro jednoduchost předpokládejte, že v následujících zadáních jsou kapacity hran dány jako přirozená čísla.

a) Ukažte, že problém Max-Flow je algoritmicky řešitelný, tj. popište nějaký algoritmus řešící tento problém.

Nápověda: Pro danou síť G s kapacitami hran c a daný tok f je možné sestrojit ***reziduální síť*** G' s kapacitami hran c' vyjadřujícími, o kolik je možné tok jednotlivými hranami původní sítě v jednom či druhém směru měnit (tj. zvýšit či snížit) bez toho, aby byly překročeny kapacity hran. Dá se ukázat, že daný tok f je maximální právě tehdy, jestliže v reziduální síti odpovídající tomuto toku neexistuje cesta z s do t.

b) Ukažte, že pokud bychom vzali jednoduchý algoritmus založený na výše uvedené myšlence, kde bychom vždy jen zjistili, jestli v dané reziduální síti existuje zlepšující cesta z s do t, a pokud ano, tak bychom zvýšili tok podél dané cesty, přičemž bychom tento krok bychom opakovali tak dlouho, dokud by existovala nějaká zlepšující cesta, tak při nevhodném volení těchto zlepšujících cest může nastat situace, kdy počet těchto iterací může odpovídat ***hodnotě*** maximálního toku, což je hodnota, která může být mnohem vyšší než počet vrcholů a hran dané sítě.

c) Navrhněte nějakou modifikaci výše uvedeného přístupu, která zaručí, že počet iterací bude polynomiální vzhledem k počtu vrcholů a hran dané sítě a bude tedy shora omezen nějakou hodnotou, která nebude záviset na konkrétních hodnotách kapacit.

   Jaká je výpočetní složitost vámi navrženého algoritmu?

Příklad 5: Uvažujme problém nalezení ***párování s maximálním počtem hran v bipartitním grafu***.

Bipartitní graf $G = (U, V, E)$ je tvořen dvěma navzájem disjunktními množinami vrcholů $U = \{u_1, \ldots, u_n\}$ a $V = \{v_1, \ldots, v_m\}$ (kde $U \cap V = \emptyset$) a množinou hran $E \subseteq U \times V$. ***Párování*** M v grafu G je libovolná podmnožina hran $M \subseteq E$, kde pro každé dvě různé hrany $(u, v), (u', v') \in M$ platí, $u \neq u'$ a $v \neq v'$. Párování M s maximálním počtem hran je takové, kde pro každé další párování M' v grafu G platí $|M'| \leq |M|$.

NÁZEV: Nalezení párování s maximálním počtem hran v bipartitním grafu

VSTUP: Bipartitní graf $G = (U, V, E)$.

VÝSTUP: Párování s maximálním počtem hran v grafu G.

(*Poznámka:* Párování s maximálním počtem hran může v grafu G existovat více. Výstupem může být libovolné z nich.)

Navrhněte algoritmus pro tento problém s polynomiální časovou složitostí.

Nápověda: Tento problém se je možné převést na problém hledání maximálního toku v síti.

Příklad 6: Připomeňme ***problém obchodního cestujícího*** (TSP— ***travelling salesmen problem***).

Instancí tohoto problému je množina n měst $V = \{1, \ldots, n\}$ a vzdálenosti pro každou dvojici těchto měst. Zápisem $d(i, j)$ označme vzdálenost z města i do města j. Můžeme předpokládat, že pro každé $i \in V$ platí $d(i, i) = 0$ a pro každou dvojici $i, j \in V$ platí $d(i, j) = d(j, i)$.

Úkolem je najít okružní cestu, která projde každým městem právě jednou a skončí ve stejném městě, kde začala, přičemž celková délka této cesty bude nejmenší možná mezi všemi takovýmito okružními cestami.

Je zřejmé, že cesta může vždy začít a skončit ve městě 1. Můžeme se tedy omezit na tento případ. Ve výše uvedené variantě také není povoleno navštěvovat města opakovaně (s výjimkou města 1, které je navštíveno dvakrát — na začátku a na konci). Úkolem je tedy najít permutaci $(i_1, \ldots, i_n)$ měst $\{1, \ldots, n\}$, kde $i_1 = 1$ a kde součet

$$d(i_1, i_2) + d(i_2, i_3) + \cdots + d(i_{n-1}, i_n) + d(i_n, i_1)$$

je nejmenší možný.

a) Navrhněte nějaký algoritmus řešící tento problém. Jaká je časová a prostorová složitost vašeho algoritmu?

b) Navrhněte algoritmus založený na následující myšlence:

   Pro každou podmnožinu $S \subseteq (V - \{1\})$ a každé $j \in S$ definujme $c[S, j]$ jako délku nejkratší cesty, která začne ve městě 1, navštíví každé město z S právě jednou (a nenavštíví žádná další města) a skončí ve městě j.

   Všechny hodnoty $c[S, j]$ je možné spočítat použitím ***dynamického programování***, kdy tyto hodnoty počítáme v pořadí podle velikosti množiny S (od nejmenších po největší), přičemž při počítání hodnot pro větší množiny můžeme využívat dříve spočítané hodnoty pro menší množiny.

   Ukažte, že tento algoritmus je možné implementovat tak, aby jeho časová složitost byla $\mathcal{O}(n^2 2^n)$. Jaká je prostorová složitost tohoto algoritmu?

   Jak vychází časová a prostorová složitost tohoto algoritmu v porovnání s vámi navrženým algoritmem v předchozím bodě. Je lepší, horší nebo stejná?