Extra úlohy

Jsou zadány čtyři extra úlohy. Za každou lze získat 1 bod a případný malý bonus pokud bude řešení hezky zpracované. S vyjímkou první úlohy je podrobně rozepsáno hodnocení dle jednotlivých kroků řešení, pro vaše pohodlí.

Můžete si libovolně vybrat úlohu, kterou budete řešit. Od každého studenta přijmu řešení pouze jedné úlohy s vyjímkou studentů kterým chybí více bodů a osobně se dohodnou. Řešení je možné iterovat, ale neočekávejte dodatečné sugestivní hinty jako u domácích úkolů.

Extra1 - Nemocnice

Uvažme následující model nemocnice: Máme dánu množinu nemocí, diagnostických přístrojů a tabulku diagnózy. Diagnostický přístroj d je dán cenou použití $c(d)$, kterou musí pacient zaplatit, pokud podstupuje diagnózu. Tabulka diagnózy pro každou kombinaci nemoci pacienta a diagnostického přístroje obsahuje jednu ze tří hodnot D, P, N. Pokud je v tabulce D, znamená to, že přístroj automaticky detekuje konkrétní nemoc pacienta a diagnóza je hotova. Pokud je v tabulce P nebo N, znamená to, že pro příslušnou nemoc vyjde výsledek pozitivní nebo negativní, konkrétní nemoc ale neznáme.

Pokud existuje pouze jedna nemoc odpovídající výsledkům všech použitých přístrojů, diagnóza je také ukončena - tzn. neexistují nemoci o kterých nevíme. Předpokládáme, že každé dvě nemoci je možné nějakým přístrojem rozlišit.

Každý pacient nejprve navštíví doktora, který určí jeho distribuci nemocí. Tedy pro každou nemoc určí pravděpodobnost, že ji pacient má. Zároveň zjistí trpělivost pacienta - kolik maximálně diagnóz podstoupí než prostě odejde. Doktor vybaví pacienta zakořeněným binárním stromem podle kterého se bude pacient orientovat v nemocnini. Pacient začíná v kořeni. Některé uzly obsahují identifikátor diagnostického přístroje, který má navštívit. takový uzel má dva syny označené P a N popisující jak má pacient pokračovat podle dílčích výsledků. Ostatní uzly obsahují nemoc, která byla průchodem stromu určena, tyto uzly nemají žádné syny. Syny představující výsledek D nereprezentujeme, je to zbytečné.

Navrhněte algortimus, jak může doktor sestavit strom diagnózy pacienta tak, aby v žádné eventualitě nepřekročil jeho trpělivost (předpokládejme, že je to vždy možné) a přitom maximalizoval očekávaný zisk nemocnice. Tedy vezmeme všecny nemoci, které může pacient mít, podíváme se kudy se bude pacient pohybovat ve stromě pokud tuto nemoc má a sečteme cenu absolvovaných diagnóz. Takto získané hodnoty pro všechny nemoci váženě sečteme, kde váha je pravděpodobnost, že pacient nemoc skutečně má.

Hint: Úlohu není radno řešit přímočaře - určit co je nejlepší další přístroj nelze přímo. Zamyslete se nad tím, jak úlohu reprezentovat a jaké jsou její vlastnosti. Tzn. reprezentujte úlohu nějakám grafem (co by měly v rozumné reprezentaci představovat vrcholy?) a určete jeho vlastnosti, které můžeme využít k řešení.

Hint: Nesnažte se optimalizovat velikost grafu, nemá to smysl, naopak si můžeme počet vrcholů nafouknout a zvlášť řešit výběr přístroje a jeho výsledek. Algoritmus se pak jednodušeji popíše.

Hint: Existuje vcelku jednoduché řešení na pár řádek, avšak nečekejte nízkou asymptotickou složitost (to nás ale trápit nebude).

Hint: Hlavním cílem úlohy je najít elegantní řešení, které lze snadno popsat, je prokazatelně korektní (optimální) a snadno by se skutečně naprogramovalo. Hodnocení bude záviset na eleganci a průzračnosti řešení. Existuje řešení, které se dá popsat i s kompletním pseudokódem na půl stránky.

Extra - Škytavé kostry

Mějme proklatě rychlou haldu S se škytavkou řízenou parametrem $s \in (0, 1]$, která si pamatuje dvojice klíč-hodnota. S umí Insert(x) v čase $\mathcal{O}(1 + log(1/s))$, který vloží dvojici (x, x) (tedy klíč = hodnota) a ExtraktMin v čase $\mathcal{O}(1)$, který vydá dvojici s nejnižší hodnotou klíče. Navíc umí Reset, který vše uvnitř smaže a nastaví novou hodnotu s. Škytavka se projevuje tak že některé dvojice poníčí. Pokud za celou dobu své existence (od posledního resetu) provedla q Insertů, pak nejvýše $s * q$ uložených dvojic má nějak náhodně zvýšenou hodnotu klíče prvků. Ačkoliv po ExtractMin vidíme původní vloženou hodnotu, nedostáváme prvky ve správném pořadí. I tak lze navrhnout velmi rychlý algortitmus na hledání minimální kostry s využitím této struktury.

1/0.1b Chceme použít Jarníka se škytavou haldou. Problém je, že nemáme Decrease. To je třeba obejít a určit časovou složitost upraveného Jarníka.

2/0.2b Uvažme, že spustíme Jarníka se škytavou haldou, dostaneme kostru T a z haldy vytáhneme všechny poničené hrany P. Ukažte, že pokud nyní najdeme minimální kostru na hranách $T \cup P$, bude minimální. Tedy ukažte, že neponičené hrany mimo T nejsou v minimální kostře.

3 Budeme iterovat postup z (1) následovně. V i-té fázi pustíme k_i kontrahujících Borůvkových kroků, nastavíme nový parametr s_i, spustíme Jarníka, získáme $T_i \cup P_i$ a iterujeme na tomto podrafu znovu.

4/0.3b V každé fázi i chceme najít nějakou vhodnou hodnotu c_i a vhodně navolit k_i a s_i t.ž. složitost fáze bude $\mathcal{O}((n_i + m_i) * log(c_i))$ a na konci zbyde nejvýše n_i/c_i vrcholů a m_i/c_i hran. Jak je zvolíme? Aby to vyšlo přesně, bude potřeba ještě vložit nějaký trik, ale budou nám stači i přibližně tak dobré vztahy.

5/0.1b Jak v lineárním čase vyřešíme rekurzi jakmile $m_i = \mathcal{O}(n_i)$?

6/0.3b Kolik celkem fází proběhne pokud vždy volíme c_i tak, aby složitost každé fáze byla zhruba stejná jako složitost první fáze? Jakou dostaneme výslednou časovou složitost?

Hint: chceme ukázat, že dosáhneme času $\mathcal{O}((n + m) * log^*(n))$. Kde $log^*(n)$ je inverz k věžové funkci. Tzn. $log^*(2^k) = 1 + log^*(k)$.

Pozn: Škytavá halda skutečně existuje a lze s ní vybudovat takto rychlý algoritmus na MST. Pro účely domácího úkolu má ale naše halda mírně hezčí (a už nereálně dobré) vlastnosti. Rozdíl je ale pouze velmi malá technikálie, kteou je třeba velmi technicky obejít.

Extra - Kreslení vnějškově rovinných grafů

Vnějškově rovinný graf je takový, který se dá nakreslit do roviny (bez křížení) tak, že všechny jeho vrcholy jsou na vnější stěně. Příkladem takových grafů jsou třeba všechny stromy, několik kružnic slepených přes jeden vrchol, žebřík, šesticyklus s vepsaným trojúhelníkem, a spoustu dalších grafů a jejich slepeniny.

Mějme na vstupu graf, který chceme takto nakreslit do roviny, nebo rozhodnout, že není vnějškově rovinný. Pro kreslení si dovolme přirozené operace, jako kreslení čar, vkládání už nakreslených částí do oblastí roviny, nakreslení cesty spojující dva vrcholy apod., rozhodně neřešme jak to technicky provést, nebo nějaké souřadnice. Není problém libovolně mnoho věcí vmáčknout do libovolně malé plochy nebo plochu deformovat.

1/0.1b Jak nakreslit do plochy libovolný strom s předkreslenýcm kořenem? Co když máme předurčené cyklické pořadí hran kolem každého vrcholu?

2/0.1b Chceme graf rozložit na bloky oddělené artikulacemi a ty kreslit zvlášť. Jakým postupem graf rozkrájíme a jak budou bloky vypadat? Pokud se podíváme na graf incidence bloků, jakou bude mít strukturu?

3/0.1b Mějme daný blok B a kružnici s artukulacemi B předkreslenými na hranici. Triviální bloky (obsahujcí pouze hranu) umíme nakreslit snadno. Musíme navrhnout jak kreslit netriviální bloky, využijeme ušaté lemma níže.

4/0.1b Jako první krok bude se bude hodit blok rozložit podle ušatého lemma. Jak to uděláme?

5/0.2b Budeme kreslit blok dle postupu z ušatého lemmatu. Není zaručeno, že ucho lze rovinně nakreslit! Není zaručeno, že pokud máme na výběr kam ucho vložit, tak libovolná možnost bude fungovat! Navrhněte pravidla jak umisťovat uši (do které stěny).

6/0.2b Ukažte, že pokud pravidla z (6) selžou, potom blok vůbec nebyl vnějškově rovinný. To lze udělat jako analogii Kuratowského věty. Naším cílem je ukázat, že buď umíme blok nakreslit a nebo obsahuje jednu ze dvou zakázaných podstruktur, které samy o sobě nejsou vnějškově rovinné. Hint: zakázané struktury budou nějaké podstruktury těch s Kuratowského věy.

7/0.2b Popiště algoritmus, který pro každý graf buď vytvoří nakreslení nebo ukáže, že v grafu existuje jedna ze dvou zakázaných podstruktur, které nejsou vnějškově rovinné. Formálně dokažte pomocí algoritmu, že každý graf je vnějškové rovinný právě tehdy pokud neobsahuje tyto zakázané struktury.

Ušaté lemma: každý 2-souvislý graf se dá postupně vybudovat z kružnice lepením uší - což jsou cesty (nebo hrany) spojující nějaké dva vrcholy. Uši se nelepí pouze na původní kružnici.

Kuratowského věta: graf je rovinný právě když neobsahuje dělení K_5 (úplný graf o 5 vrcholech) nebo $K_{3,3}$ (úplný bipartitní graf s partitami velikosti 3). Tedy v rovinném grafu není podgraf, který by šel získat z K_5 nebo $K_{3,3}$ nahrazením hran za cesty.

Extra - Nejbližší body

Mějme na vstupu seznam (ne nutně celočíselných) 2D-souřadnic. Chceme efektivně najít (nějakou) dvojici bodů s nejmenší vzdáleností. To jde snadno kvadraticky, ale my to chceme rychleji. Prvotní idea je jednoduchá, plochu rozdělíme pomocí svislé přímky tak, aby se body rozdělily víceméně rovnoměrně, vyřešíme v každé části zvlášť a výsledek zkombinujeme. To by mohlo složitostí odpovídat quick-sortu, ale k tomu musíme rozdělení i kombinování výsledků zvládnout lineárně k čemuž chybí spoustu detailů.

1/0.1b Technická otázka: počítání odmocnin je pomalé. Jak se obejít bez odmocnin?

2/0.1b Jak rozdělit body do skupin lineárně? Bude se nám hodit vstup předzpracovat. Body na předělu rozhoďme libovolně, bude se to tak hodit.

3/0.1b Pokud zpracujeme obě skupiny zvlášť, musíme uvážit ještě vzdálenosti napříč. Jakou časovou složitost bude takový post-processing obecně mít?

4/0.1b Krok stranou: kolik bodů vzájemné vzdálenosti alespoň d může ležet uvnitř čtverce $d \times d$?

5/0.1b Využijte (4) a ukažte, že v řešení (3) pro každý bod blízko dělící přímky stačí uvážit konstantně mnoho kandidátů pro vzdálenost menší než d pokud d zvolíme vhodně. Spočítejte kolik jich je přesně.

6/0.1b Navrhněte, jak pomocí (5) vyřešit (3) lineárně pokud máme body seřazené podle y-ové souřednice.

7/0.2b Navrhněte, jak zajistit předpoklad setřízení v (6) se zachováním lineární složitosti uzlů. Třídit lineárně sice neumíme ale stačí uvážit vhodný algoritmus pasující na naši situaci.

8/0.2b Zapište pseudokód algoritmu společně s důkazem korektnosti a časové i prostorové složitosti. Odkazujte se na dříve ukázaná fakta.