Zadání maturitní zkoušky z IVT

Z bitmapových grafických editorů určitě znáte nástroj plechovka. Vaším úkolem bude naprogramovat chytřejší verzi plechovky, která bude ignorovat drobné změny v barvě.

Popis plechovky

Plechovka je nástroj, který umožňuje vybarvit plochu stejnou barvou. Uživatel klikne kamkoliv do této plochy a celá plocha se nastaví na zadanou barvu. Každé použití plechovky má tři parametry – souřadnice pixelu, kam uživatel kliknul, barvu, na kterou chce uživatel danou plochu nastavit a citlivost, která udává, jak moc je plechovka „tolerantní“ ke změnám v barvě.

Pojďme tuto intuitivní představu rozvést přesněji. Plechovka dostane obrázek o rozměrech `width` × `height`. Levý horní pixel má souřadnice $(0, 0)$. Pravý dolní pixel má souřadnice ($\texttt{height} - 1$, $\texttt{width} - 1$). Pixel, na který uživatel kliknul, má souřadnice (`start_x`, `start_y`) a budeme mu říkat startovní. Startovní pixel generuje plochu – množinu pixelů, které jsou dosažitelné ze startovního pixelu. Plechovka nastaví všechny pixely z plochy generované startovním pixelem na zadanou barvu (`target_r`, `target_g`, `target_b`). Pixel p je dosažitelný, když existuje cesta ze startovního pixelu do p, kde pro každé dva po sobě jdoucí pixelu je splněný predikát `transition`.[1]

Predikát (funkce) `transition` dostane dva (v praxi sousední) pixely a vrátí `true`, pokud mají „dostatečně podobnou barvu, aby se skrze ně rozlila barva z plechovky dál“. Nalezení vhodného predikátu je klíčové pro správnou detekci drobných změn v barvě. Pro naše účely použijeme následující predikát, který dostane dva pixely – dvě trojice (r, g, b)

$$\texttt{transition}((r_1, g_1, b_1), (r_2, g_2, b_2)) := |r_1 - r_2| + |g_1 - g_2| + |b_1 - b_2| \leq \texttt{sensitivity}$$

Pojďme si ujasnit, jaké má tato definice důsledky. Pokud bychom místo našeho predikátu, dali jako predikát `transition` rovnost (tedy funkci, která vrátí `true` právě tehdy, když se odpovídající barevné složky rovnají), dostaneme hloupou plechovku, která se „zastaví“ na jakkoliv malé změně barvy. Stejného efektu docílíme, když nastavíme parametr `sensitivity` na 0. Naopak když nastavíme `sensitivity` na $3 \cdot 256 = 768$ nebo větší, získáme místo plechovky nástroj, který vybarví celou plochu, protože je má citlivost tak nízkou, že se na žádné změně barvy „nezastaví“. Také si uvědomme, že při rozumném nastavení `sensitivity` může plocha stále obsahovat dva velmi rozdílné pixely – například pokud bude celý obrázek gradient (plynulý přechod) od bílé po černou.

Uživatelské rozhraní

Většinou bývá plechovka součástí nějakého většího grafického editoru. Pro naše účely ale budeme implementovat jednoduché rozhraní pro používání plechovky pomocí *receptů*. Váš program dostane na standartním vstupu jednoduchý recept, podle kterého bude vytvářet výstup. Recept bude jednoduchý textový soubor, kde na každém řádku bude jeden parametr. Jednotlivé řádky obsahují následující údaje (na řádku oddělené mezerou).

1. Cesta ke vstupnímu souboru
2. Cesta k výstupnímu souboru
3. `start_x start_y`
4. `target_r target_g target_b`
5. `sensitivity`

Můžete předpokládát, že uživatel vytvoří cestu pro výstupní soubor před spuštěním programu. Pro příklad viz testy.

Formát obrázku

Všechny obrázky (vstupní i výstupní) budou ve formátu BMP. Obrázek bude 24 bit per pixel s barvami R8G8B8. Výstup musí být čitelný pomocí běžné knihovny na práci s obrázky. Porovnáván s originálem bude v rozměrech a v barvě všech pixelů.

Hint: Pokud chcete používat nějakou knihovnu, doporučuji nevyhledávat BMP knihovnu, ale obecně knihovnu pro práci s obrázky. Skoro určitě bude mít podporu pro BMP a pravděpodobně lepší rozhraní a dokumentaci než specifická BMP knihovna.

[1] Pro úplnost pojďme dosažitelnost definovat formálněji. Pixel p je dosažitelný z pixelu `start`, když existuje posloupnost (cesta) pixelů ($\texttt{start} = p_0, p_1, \ldots, p_n = p$) taková, že pro všechna i z množiny $\{0, 1, \ldots, n-1\}$ platí, že p_i a p_{i+1} jsou sousední a p_i a p_{i+1} splňují predikát `transition`. Pixely se souřadnicemi (x_1, y_1) a (x_2, y_2) jsou sousední právě tehdy když $|x_1 - x_2| + |y_1 - y_2| = 1$.

Pokyny

Ve vhodném programovacím jazyce naprogramujte zadaný úkol. Zakázáno máte akorát komunikovat. Můžete používat internet, literaturu, knihovny, tutoriály, dokumentace, svoje vlastní poznámky, svůj vlastní kód apod. Pište přehledný, komentovaný, čitelný kód. Nebojte se mě ptát, v nejhorším vám neodpovím, případně vám nabídnu nějakou Devil's deal – popostrčím vás kupředu za cenu bodové ztráty (abyste se nezasekli na „blbosti“).

Začněte jednoduchými částmi. Část testovacích vstupů nevyžaduje plné splnění úkolu, ale stačí implementace jednoduššího algoritmu. Pokud nestíháte implementovat složitější algoritmus, nebojte se commitnout jenom popis algoritmu a náznak implementace.

Proveďte analýzu časové složitosti – skutečné. Tedy ne jakou byste chtěli, aby váš program měl, ale jakou váš program skutečně má. Lepší neoptimální řešení, než nefunkční řešení. Pokud nebudete mít optimální řešení, pravděpodobně si o vylepšení popovídáme.

Váš program bude vyhodnocován na serveru Hiccup. Pokud není použití vašeho programu zcela jasné (např. `python program.py` nebo `cargo build`), přiložte `README.md`. Silně doporučuji – **průběžně zkoušejte, jestli jde program na serveru Hiccup spustit**.

Odevzdávání

Založte si repozitář v domovské složce na serveru Hiccup s názvem `maturita.git` (název dodržte přesně). Do tohoto repozitáře musím být schopný pushovat. Budu do repozitáře pushovat jako `root`, je tedy potřeba ošetřit, aby nedocházelo k problémům s právy. Pushnutí do repozitáře totiž vytvoří určité interní gitové soubory (ve složce `.git`). Tyto soubory defaultně vlastní a má právo editovat pouze ten, kdo je vytvořil. Když je tedy vytvořím já, nebudete je moct editovat a tedy pracovat s gitem. Je potřeba nastavit (ideálně při vytváření repozitáře), aby nově vytvořené soubory měly jiná práva (hint: `--shared`). Na požádání vám do repozitáře pokusím pushnout, abyste věděli, jestli vše funguje.

Commitujte a pushujte průběžně. Budu vám dávat průběžný feedback ke všemu, co pushnete. Je to ve vašem zájmu.

Analýzu časové složitosti uveďte do souboru `casova_slozitost.md`.

Počítá se to, co je pushnuté.

Testování

K dispozici máte sadu testů – receptů a příslušných obrázků společně se správnými výstupy. Testů není málo. Abych vám proces testování usnadnil, napsal jsem v Bashi framework pro testování. Chybí mu však samotný script na porovnání obrázků. Obrázky se stejným obsahem totiž mohou mít např. jinak velkou hlavičku (viz soubory `header.bmp` a `header2.bmp`). Napsání tohoto scriptu je vhodný způsob, jak se seznámit s knihovnou nebo kódem, který budete používat na zpracovávání obrázků.

Hodnocení

Funkční

Procenta jsou pouze orientační.

- forma – 10 % Formou se rozumí odevzdávání prostřednictvím Gitu, schopnost získat zadání z Linuxového serveru a další technické dovednosti potřebné ke splnění úkolu.
- parsování receptu – 5 %
- načtení obrázku a netriviální manipulace (např. porovnání dvou obrázků) – 15 %
- pouze jedna plocha[2] – 30 %
- více ploch – 30 %
- asymptotická složitost – 10 % V případě triviálního algoritmu nemusí být v této kategorii uděleny body, i když je analýza asymptotické složitosti „korektní“.

[2]Pro splnění této části můžete předpokládat, že všechny pixely podobné barvy (tedy takové, že platí `transition` se startovním pixelem) tvoří jednu plochu, která obsahuje startovní pixel. Neexistují tedy pixely, které by měly podobnou barvu a které by nebyly dosažitelné ze startovního pixelu. Není tedy vůbec potřeba ověřovat „souvislost plochy“.

Nefunkční

V odevzdaném řešení bude posuzována koreknost, postup, efektivita a celková kvalita kódu.