FAKULTA
APLIKOVANÝCH VĚD
ZÁPADOČESKÉ
UNIVERZITY
V PLZNI

Semestrální práce z předmětu KIV/PC

# Přebarvování souvislých oblastí ve snímku

Max Nonfried
A19B0601P

13. prosince 2021

# Obsah

# Zadání

Naprogramujte v ANSI C přenositelnou[1] konzolovou aplikaci, která provede v binárním digitálním obrázku (tj. obsahuje jen černé a bílé body) obarvení souvislých oblastí pomocí níže uvedeného algoritmu Connected-Component Labeling z oboru počítačového vidění. Vaším úkolem je tedy implementace tohoto algoritmu a funkcí rozhraní (tj. načítání a ukládání obrázku apod.). Program se bude spouštět příkazem:

`ccl.exe`[2] ⟨`input-file[.pgm]`⟩ ⟨`output-file`⟩

Symbol ⟨input-file⟩ zastupuje jméno vstupního souboru s binárním obrázkem ve formátu Por- table Gray Map, přípona souboru nemusí být uvedena; pokud uvedena není, předpokládejte, že má soubor příponu .pgm. Symbol ⟨output-file⟩ zastupuje jméno výstupního souboru s obarveným[3] obrázkem, který vytvoří vaše aplikace. Program tedy může být během testování spuštěn například takto:

```
...\>ccl.exe e:\images\img-inp-01.pgm e:\images\img-res-01.pgm
```

Úkolem vašeho programu tedy je vytvořit výsledný soubor s obarveným obrázkem v uvedeném umístění a s uvedeným jménem. Vstupní i výstupní obrázek bude uložen v souboru ve formátu PGM, který je popsán níže. Obarvení proveďte podle níže uvedeného algoritmu. Testujte, zda je vstupní obraz skutečně černobílý. Musí obsahovat pouze pixely s hodnotou 0x00 a 0xFF. Pokud tomu tak není, vypište krátké chybové hlášení (anglicky) a oznamte chybu operačnímu prostředí pomocí nenulového návratového kódu[4].

Hotovou práci odevzdejte v jediném archivu typu ZIP prostřednictvím automatického odevzdávacího a validačního systému. Postupujte podle instrukcí uvedených na webu předmětu. Archiv nechť obsahuje všechny zdrojové soubory potřebné k

[1] Je třeba, aby bylo možné Váš program přeložit a spustit na PC s operačním prostředím Win32/64 (tj. operační systémy Microsoft Windows NT/2000/XP/Vista/7/8/10) a s běžnými distribucemi Linuxu (např. Ubuntu, Mint, OpenSUSE, Debian atp.). Server, na který budete Vaši práci odevzdávat a který ji otestuje, má nainstalovaný operační systém Debian GNU/Linux 10 Buster s jádrem verze 4.19.0-11-amd64 a s překladačem gcc 8.3.0.

[2] Přípona .exe je povinná i při sestavení v Linuxu, zejm. při automatické kontrole validačním systémem.

[3] Nejde o obarvení v pravém smyslu slova, protože použitý formát PGM je šedotónový. Obarvením se myslí nahrazení bílé barvy v souvislých oddělených oblastech různými tóny šedé.

[4] Stejně tak čiňte i v případě jiných chyb, např. nesprávném formátu vstupního souboru a podobně.

přeložení programu, makefile pro Windows i Linux (pro překlad v Linuxu připravte soubor pojmenovaný makefile a pro Windows makefile.win) a dokumentaci ve formátu PDF vytvořenou v typografickém systému TEX, resp. LATEX. Bude-li některá z částí chybět, kontrolní skript Vaši práci odmítne.

## 1.1 Algoritmus

Přebarvování souvislých oblastí (Connected-Component Labeling, CCL) je dvouprůchodový algoritmus z oblasti počítačového vidění. Jeho vstupem je binární obrázek, tj. takový, který obsahuje jen bílé a černé pixely. Bílé pixely představují jednotlivé objekty, které se tento algoritmus pokouší izolovat a označit různými barvami, resp. hodnotami intenzity šedé barvy (tedy tzv. labely, česky štítky, značkami). Černé pixely pak představují pozadí.

Algoritmus CCL pracuje tak, jak je uvedeno v následujícím popisu:

**1. průchod:** Procházejte obrázek po řádcích. Každému nenulovému pixelu (tj. pixelu představujícímu objekt) na souřadnicích [i, j] přiřaďte hodnotu podle hodnoty jeho sousedních pixelů, pokud nenulové sousední pixely existují (poloha sousedů je dána maskou na obrázku č. 1). Všechny sousední pixely dané maskou již byly obarveny v předchozích krocích (to je zajištěno tvarem masky).
- Jsou-li všechny sousední pixely součástí pozadí (mají hodnotu 0x00), přiřaďte pixelu [i, j] dosud nepřidělenou hodnotu - novou barvu.
- Má-li právě jeden ze sousedních pixelů nenulovou hodnotu, přiřaďte obarvovanému pixelu hodnotu nenulového sousedního pixelu.
- Je-li více sousedních pixelů nenulových, přiřaďte obarvovanému pixelu hodnotu kteréhokoli nenulového pixelu ze zkoumaného okolí. Pokud byly hodnoty pixelů v masce různé (došlo k tzv. kolizi barev), zaznamenejte ekvivalenci dvojic hodnot do zvláštní datové struktury tabulky ekvivalence barev.

**2. průchod:** Po průchodu celého obrázku jsou všechny pixely náležející oblastem (objektům) obarveny, některé oblasti jsou však obarveny více barvami (díky kolizím). Proto projděte znovu celý obraz po řádcích a podle informací o ekvivalenci barev (z tabulky ekvivalence barev získané v průběhu 1. průchodu) přebarvěte pixely těchto oblastí. Z množiny kolizních barev je možné nějak vybrat jednu (první, poslední, náhodně), nebo opět postupovat při přiřazování barev „od začátku“ a jako novou barvu použít index množiny (nesmí být samozřejmě nulový, protože barva 0x00 představuje pozadí). Po tomto kroku odpovídá každé oblasti označení (obarvení) jedinou, v jiné oblasti se nevyskytující hodnotou (barvou).

Obrázek č. 2 představuje binární (černobílý) obrázek na vstupu algoritmu. Černé oblasti představují pozadí, bílé oblasti (získané tzv. prahováním) představují objekty. Na obrázku č. 3 vidíte výsledek činnosti algoritmu - každý objekt je obarven unikátní barvou. Lze tedy např. zjistit počet objektů ve scéně, zkoumat jejich tvar apod.

Obarvením se zde - v případě šedotónových obrázků - pochopitelně nemyslí změna barvy pixelů v reprezentaci např. RGB (jak již bylo naznačeno), ale označení každé

souvislé oblasti jinou úrovní šedi, takže je pak každý „obarvený“ objekt v obrázku snadno identifikovatelný touto svojí „barvou“.

Celé znění zadání semestrální práce je dostupné na
`https://www.kiv.zcu.cz/studies/predmety/pc/#work`

# Analýza úlohy

## 2.1 Connected-component labeling

*Connected-component labeling (CCL)* je algoritmus z teorie grafů pro detekci spojitých částí v digitálním snímku v binárním formátu (tj. snímek obsahuje pouze dvě barvy, typicky bílou a černou), nicméně lze ho použít i pro vícebarevné snímky. Základní princip algoritmu je takový, že pixely ve spojité části mají stejný popisek (a následně mohou být obarveny stejným odstínem).

Z hlediska teorie grafů se ze vstupních dat vytvoří graf. Pixely představují vrcholy, sousedé hrany. Vrcholy jsou algoritmem označovány na základě konektivity sousedů (obrázek 2.1 a 2.2), hodnot jejich sousedů a hodnot popisků sousedů.

## 2.2 Algoritmy *CCL*

Existuje vícero algoritmů, kterými lze dosáhnout obarvení obrázku. Popsané jsou níže dva. Předpokládá se, že vstupní soubor je binární obrázek s pixely, které tvoří buď pozadí nebo popředí, přičemž spojité části se hledají v popředí.

### 2.2.1 Obarvení jedné spojité části po druhé

Kroky algoritmu podle Wikipedie.[1]

Vytvořte zásobník implementovaný spojovým seznamem, který bude uchovávat indexy pixelů, které tvoří spojitou část.

1. Začněte od prvního pixelu v obrázku. Aktuální popisek nastavte na 1. Pokračujte bodem 3.

2. Jestliže je tento pixel pixel popředí a ještě nemá popisek, přiřaďte mu aktuální popisek a přidejte ho do zásobníku. Pokračujte bodem 3. Jestliže je daný pixel pozadí nebo již má popisek, pak pokračujte bodem 2 s následujícím pixelem.

3. Vraťte a vymažte prvek ze zásobníku, zjistěte, kdo jsou jeho sousedi (na základě typu konektivity). Jestliže je soused pixel v popředí a ještě nemá

[1] `https://en.wikipedia.org/wiki/Connected-component_labeling`

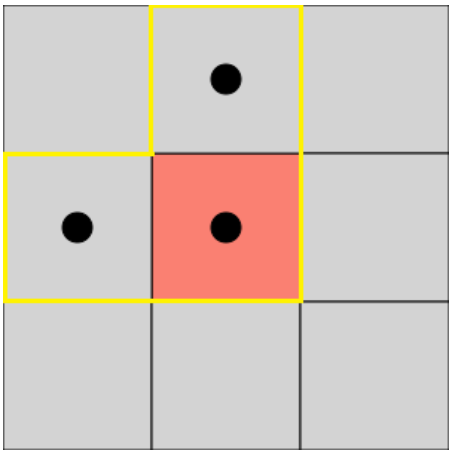
popisek, přiřaďte mu aktuální popisek a vložte ho do zásobníku. Opakujte bod 3, dokud není zásobník prázdný.

4. Pokračujte bodem 2 s dalším pixelem v obrázku a zvyšte popisek o 1.

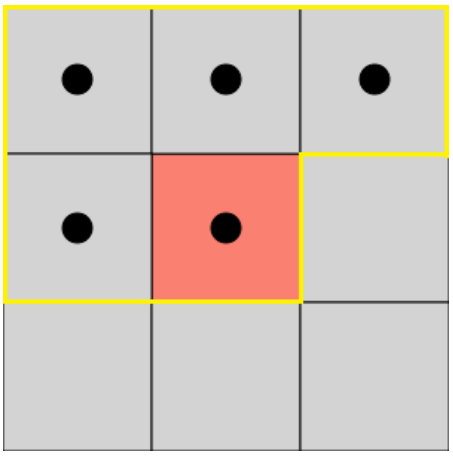
### 2.2.2 Dvouprůchodový algoritmus

Algoritmus má dva průchody souborem. V prvním se obarvují pixely a přiřazují se jim pravidla pro přepis. V druhém kole se podle vzniklých pravidel přebarví dané pixely. Tento algoritmus byl lehce upravený použit pro implementaci.

Popis algoritmu viz kapitola 1.1.



Obrázek 2.1: Konektivita sousedů typu *4 sousedé*



Obrázek 2.2: Konektivita sousedů typu *8 sousedů*

# Implementace

Pro obarvení obrázku byl použit algoritmus *Connected-component labeling (CCL)*.

## 3.1 Popis souborů

Stručný popis hlavních souborů, ze kterých se program skládá.

`ccl.c` Hlavní soubor programu. Obsahuje funkci `main()`, dále funkce pro řízení kontroly souboru, pro načítání a výpis souboru.

`definitions.h` Obsahuje definice konstant a struktury, které se používají v celém programu.

`checking_tools.c` Obsahuje nástroje pro kontrolu souboru.

`shading.c` Obsahuje veškeré funkce pro obarvení souboru.

## 3.2 Popis použitých proměnných

Data vstupního souboru a téměř veškeré proměnné potřebné pro práci s ním jsou zabaleny ve struktuře `myfile` (viz kód 3.1). S tou se pracuje v celém programu za pomoci ukazatelů.

```
1         typedef unsigned char uchar;
2         typedef struct themyfile {
3            FILE *fp;
4            uchar *data, equivalency[NUMBER_OF_SHADES],
   neighbours_values[NUMBER_OF_NEIGHBOURS];
5            int columns, rows, data_length, shades_labels[
   NUMBER_OF_SHADES];
6         } myfile;
7
```

Zdrojový kód 3.1: Struktura `myfile`

Pro pochopení implementace je nezbytný její důkladný popis.

`fp` Je ukazatel na soubor.

`data` Je ukazatel na pole, ve kterém jsou uložena pouze data souboru (nikoliv hlavička).

`equivalency` Je polem implementovaná tabulka ekvivalence. Slouží k zaznamenání pravidel pro přepis odstínů, zároveň slouží jako indikátor, zda je daný odstín použitý či ne. Na začátku je inicializována na samé 0. Jednotlivé odstíny odpovídají indexování pole. Výčet možných typových situací s vysvětlením:

- `equivalency[5] = 0` odstín 5 nebyl dosud použit (je volný),
- `equivalency[5] = 5` odstín 5 se přepisuje sám na sebe,
- `equivalency[5] = 25` odstín 5 se přepisuje na odstín 25.

`neighbours_values` Je pole, které slouží pro uložení hodnot aktuálních sousedů, tj. jsou v něm uloženy hodnoty sousedních pixelů.

`columns` Je počet sloupců.

`rows` Je počet řádků.

`data_lenght` Je délka dat.

`shades_labels` Je pole, do kterého jsou zaznamenávána pořadová čísla („popisky“) jednotlivých odstínů. Příklad:

- `shades_labels[5] = 37` odstín 5 je 37. použitý odstín.

## 3.3 Popis funkcí

Popis složitějších funkcí, které jsou součástí algoritmu. Jsou seřazeny chronologicky.

1. Funkce `choose_neigbours_values()` vybere sousedy k aktuálnímu pixelu na základě tvaru masky. Ta se upravuje podle pozice pixelu, tj. maska je upravená pro první řádek a pro první a poslední sloupec. Hodnoty sousedů se ukládají do pole `neighbours_values`.

2. Funkce `shade_data_round1()` obarví aktuální pixel.

   - **4 sousedi s hodnotou 0** Vybere se nový odstín funkcí `choose_shade()` (např. 5). V tabulce ekvivalence vznikne záznam `equivalency[5] = 5` (přepisuje se sám na sebe). Pořadí odstínu se zaznamená jako `shades_labels[5] = <pořadí odstínu>`.
   - **1 nenulový soused** Pixelu se přiřadí hodnota, na kterou se přepisuje jeho soused. Tedy pokud se soused přepisuje např. na odstín 25, tak „náš“ pixel se obarví na odstín 25.
   - **2, 3, 4 nenuloví sousedi** Sousedi se přeskupí funkcí `sort_neighbours_by_labels()`. Pixel se obarví na odstín, na který se přepisuje soused vybraný zmíněnou funkcí. Ostatním sousedům se do tabulky ekvivalence uloží pravidlo, že se budou přepisovat též na odstín, na který se přepisuje vybraný soused.

2.1 Funkce `sort_neighbours_by_labels()` vybere souseda, jehož odstín, na který se přepisuje, má nejmenší pořadové číslo. Vybraného souseda uloží na třetí pozici v `neighbours_values()`.

2.2 Funkce `choose_shade()` náhodně vybere nový nepoužitý odstín pro pixel. Zda je odstín použitý či ne, je uloženo v tabulce ekvivalence. Pokud je odstín 250. v pořadí, tak funkce skončí s návratovou hodnotou `-1`.

3. Funkce `shade_data_round2()` na základě tabulky ekvivalence provede přebarvení dat (druhé kolo algoritmu). Předchází jí funkce `simplify_equivalency()`, která zajistí, že se odstíny přepisují přímo na konečný odstín.

Pokud je nový odstín v pořadí 250., tak se předčasně provede druhé kolo algoritmu. Následně se z tabulky ekvivalence odstraní záznamy, kde se odstíny nepřepisují samy na sebe (přiřadí se jim 0). A zkontroluje se, zda počet zbylých odstínů v tabulce ekvivalence (reálný počet použitých odstínů) nepřekračuje hranici 250. Pokud ano, program uloží výsledný obrázek v aktuálním, rozpracovaném stavu a ukončí činnost.

# Uživatelská příručka

## 4.1 Spuštění

Program se spouští se dvěma argumenty. Pokud jich bude jiný počet, program skončí a vypíše chybovou hlášku. První argument je cesta ke vstupnímu souboru, druhý je cesta, kam se má uložit výsledný soubor.

### 4.1.1 Formát vstupního a výstupního souboru

Vstupní soubor musí být ve formátu *Portable Gray Map (PGM)*. Přípona souboru není povinná, avšak bude-li jiná než `.pgm`, tak program soubor odmítne.

Vstupní soubor musí splňovat tyto parametry:

- musí být ve formátu *Portable Gray Map (PGM)*;
- přípona souboru není povinná, avšak bude-li jiná než `.pgm`, tak program soubor odmítne;
- hlavička souboru musí být ve formátu:

  ```
  P5
  <počet řádek> <počet sloupců>
  255
  ```

- data souboru musí nabývat pouze hodnot 0 a 1.

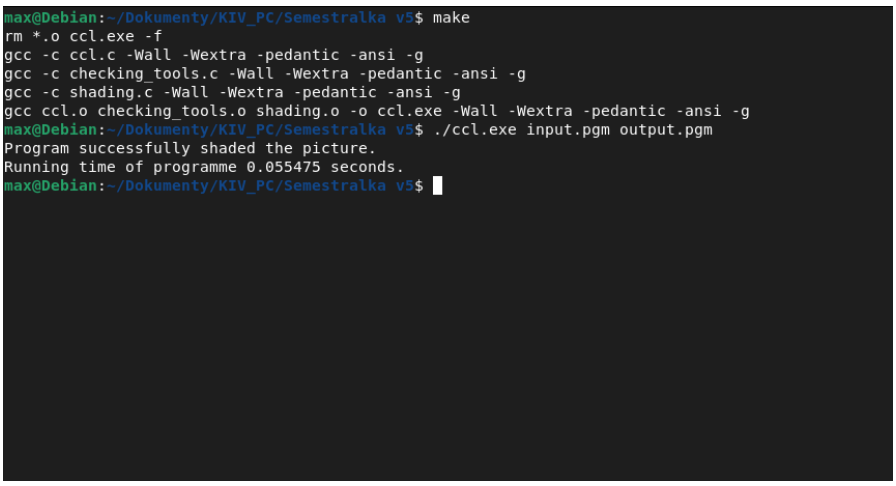
Výstupní soubor je vždy formátu *PGM*.

### 4.1.2 Příkazy pro spuštění

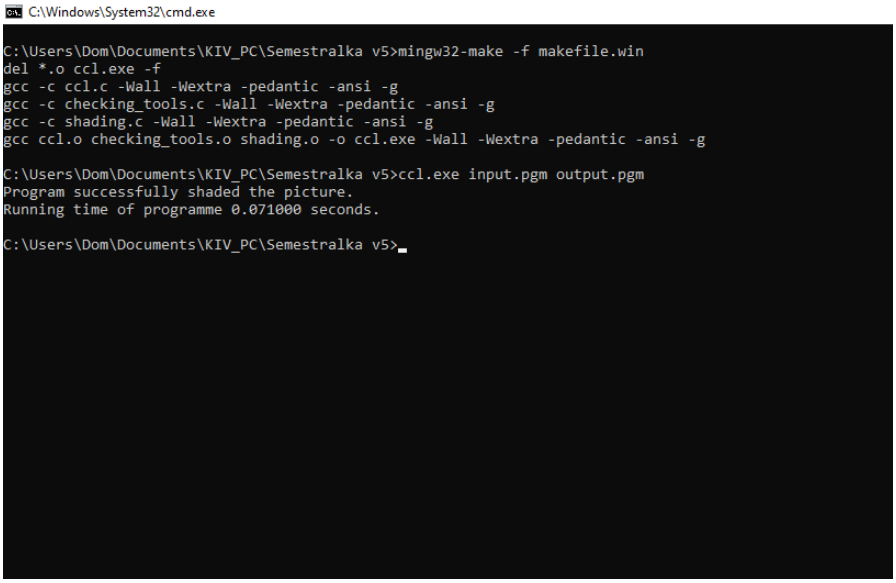
#### Linux

Příkazy pro spuštění v interpretovi příkazů *Bash* (viz obrázek 4.1):

1. `make` - příkaz pro překlad a zkompilování;
2. `./ccl.exe <cesta ke vstupnímu souboru> <cesta k výstupnímu souboru>` - příkaz pro spuštění samotného programu.




Obrázek 4.1: Spuštění programu v interpretovi příkazů *Bash* na Linuxu




Obrázek 4.2: Spuštění programu v interpretovi příkazů *cmd* na Windows

### Windows

Příkazy pro spuštění v interpretovi příkazů *Command Prompt* (viz obrázek 4.2):

1. `mingw32-make -f makefile.win` - příkaz pro překlad a zkompilování;
2. `ccl.exe <cesta ke vstupnímu souboru> <cesta k výstupnímu souboru>` - příkaz pro spuštění samotného programu.

Spuštění v interpretovi příkazů *Powershell*:

1. `mingw32-make -f makefile.win` - příkaz pro překlad a zkompilování;
2. `./ccl.exe <cesta ke vstupnímu souboru> <cesta k výstupnímu souboru>` - příkaz pro spuštění samotného programu.

## 4.2 Ukázka obrázků

Ukázka obrázků před obarvením a po obarvení.

Obrázek 4.3: Vstupní černobílý obrázek

Obrázek 4.4: Výstupní obarvený obrázek

# Závěr

Program úspěšně obarví obrázek na platformách Linux i Windows. Doba běhu programu se vstupním obrázkem 4.3 se na operačních systémech Linux a Windows pohybuje od 0,05 do 0,08 sekundy.

Možné vylepšení vidím ve výběru nového odstínu, tedy ve funkci `choose_shade()`. Ta nyní vybírá nové odstíny pseudonáhodně funkcí `rand()`, dokud počet aktuálně použitých odstínů není 250. Čím více se počet odstínů zvyšuje, tím nalezení nového odstínu trvá déle. Řešením by bylo vytvoření „deterministějšího“ algoritmu, který by tvořil posloupnost celých čísel například po deseti. Samozřejmě by se muselo ošetřit, aby odstíny nabývaly hodnot v intervalu (0, 256).

Příklad takové posloupnosti:
10, 20, …, 240, 250,
1, 11, 21, …, 241, 251,
2, 12, 22, …, 242, 252,
…

# Seznam obrázků

# Seznam zdrojových kódů

# Literatura

1] https://en.wikipedia.org/wiki/Connected-component_labeling