



Semestrální práce z KIV/PC  
PŘEBARVOVÁNÍ SOUVISLÝCH OBLASTÍ  
VE SNÍMKU

Lukáš Runt (A20B0226P)

*lrunt@students.zcu.cz*

2. ledna 2022

# Obsah

<b>Obsah</b>	<b>1</b>
<b>1 Zadání</b>	<b>2</b>
<b>2 Analýza úlohy</b>	<b>3</b>
2.1 Formát PGM souboru . . . . .	3
2.2 Algoritmus Přebarvování souvislých oblastí . . . . .	3
2.2.1 První průchod . . . . .	3
2.2.2 Druhý průchod . . . . .	4
2.3 Zaznamenání kolizí . . . . .	4
2.3.1 Množina . . . . .	5
2.3.2 Spojový seznam . . . . .	5
2.3.3 Pole . . . . .	5
<b>3 Popis implementace</b>	<b>5</b>
3.1 Kontrola a úprava argumentů . . . . .	5
3.2 Struktura pgm . . . . .	6
3.2.1 Načítání ze souboru . . . . .	6
3.2.2 Algoritmus obarvení dat . . . . .	6
3.2.3 Zaznamenávání kolizí . . . . .	6
3.2.4 Vytváření souboru . . . . .	6
3.3 Struktura matrix . . . . .	7
<b>4 Uživatelská příručka</b>	<b>7</b>
4.1 Sestavení . . . . .	7
4.2 Spuštění . . . . .	7
<b>5 Závěr</b>	<b>8</b>
<b>6 Reference</b>	<b>9</b>

# 1 Zadání

Naprogramujte v ANSI C přenositelnou **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 <input-file[.pgm]> <output-file>
```

Symbol `<input-file>` zastupuje jméno vstupního souboru s binárním obrázkem ve formátu *Portable 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 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 programu je tedy vytvořit výsledný souboru s obarveným obrázkem uvedeným umístění a s uvedeným jménem. Vstupní i výstupní obrázek bude uložen ve formátu PGM.

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.

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 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  $\text{\TeX}$ , resp.  $\text{\LaTeX}$ . Bude-li některá z částí chybět, kontrolní skript Vaši práci odmítne.

Celé zadání je dostupné na adrese: <http://www.kiv.zcu.cz/studies/predmety/pc/data/works/sw2021-02.pdf>

## 2 Analýza úlohy

Zadaný problém lze rozdělit na několik dílčích podproblémů, a to na kontrolu vstupních argumentů, načítání a vytváření pgm souboru a následně algoritmus, který přebarví souvislé oblasti.

### 2.1 Formát PGM souboru

Na vstupu a výstupu programu je soubor ve formátu PGM. Formát tohoto souboru je následující:

<b>P5</b>	Každý PGM soubor obsahuje z "Magické
<b>1024 768</b>	číslo" pro identifikaci typu souboru (v našem
<b>255</b>	případě 'P5', tedy binární šedotónová data),
.....	bílý znak (může být mezera, TAB, CR, LF
.....	...), šířku (počet sloupců, řetězcem znaků),
	bílý znak, výšku (počet řádků, řetězcem
	znaků), bílý znak, index nejvyšší hodnoty
	šedé (řetězcem znaků), bílý znak, byty (v
	jazyce C datový typ <code>unsigned char</code> ) před-

Obrázek 1: PGM soubor

stavující hodnoty jednotlivých pixelů.

### 2.2 Algoritmus Přebarvování souvislých oblastí

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

#### 2.2.1 První průchod

Procházíme obrázek po řádcích. Každému nenulovému pixelu ne souřadnicích  $[i, j]$  přiřadíme hodnotu podle hodnoty jeho sousedních pixelů, pokud nenulové sousední pixely existují (poloha sousedů je daná 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), přiřadíme pixelu  $[i, j]$  dosud nepřidělenou hodnotu, nebo-li novou barvu.

- Má-li právě jeden ze sousedních pixelů nenulovou hodnotu, přiřadíme obarvovanému pixelu hodnotu nenulového sousedního pixelu.
- Je-li více sousedních pixelů nenulových, přiřadíme obarvovanému pixelu hodnotu kteréhokoli nenulového pixelu ze zkoumaného okolí. Pokud byly hodnoty pixelů v masce různé (došlo k tzv. *kolizi barev*), zaznamenáme ekvivalenci dvojic do zvláštní datové struktury - tabulky ekvivalence barev.



Obrázek 2: Maska pro přebarvování [1]

### 2.2.2 Druhý průchod

Po prvním průchodu celého obrázku jsou všechny pixely náležející oblastem (objektům) obarveny, některé oblasti jsou však obarveny více barvami (díky kolizím). Proto musíme znova projít celý obrázek a podle informací o ekvivalenci barev (z tabulky ekvivalence barev získané v průběhu 1. průchodu) přebarvíme pixely těchto oblastí. Z množiny kolizních barev je možné nějak vybrat jednu (první, poslední, náhodnou) 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).

## 2.3 Zaznamenání kolizí

Při prvním průchodu je potřeba si zaznamenávat kolize, pokud je přítomno v masce více rozdílných barev. Tyto kolize lze zaznamenávat pomocí několika níže uvedených metod.

### 2.3.1 Množina

Množina je abstraktní datový typ, který je schopen uložit určité hodnoty bez jakéhokoliv pořadí a bez opakujících se hodnot. Narozdíl od pole jsou množiny neuspořádané a neindexované. V informatice je teorie množin užitečná pokud potřebujeme shromáždit data a nezáleží nám na jejich násobnosti nebo pořadí, což je právě náš případ. V případě kolize by se prvky přidávali do množin a při druhém průchodu by se pak bral nejmenší prvek.

### 2.3.2 Spojový seznam

Spojový seznam je dynamická datová struktura, určená k ukládání dat předem neznámé délky. Základní stavební jednotkou spojového seznamu je uzel, který vždy obsahuje ukládanou hodnotu a ukazatel na následující prvek. V našem případě by toto šlo implementovat tak, že by jeden ukazatel měl více pointerů a nebo by vždy při kolizi vložil prvky do svého seznamu pokud by ještě prvky v seznamu nebyly.

### 2.3.3 Pole

Tento způsob je velice podobný způsobu se spojovými seznamy. Jedná se o pole, kde každý index, kromě indexu 0, znamená číslo barvy. Obsah indexu pak znamená v případě, že není zrovna roven -1, ukazatel na další barvu (index), se kterou je v kolizi. Pokud je hodnota na indexu rovna hodnotě -1, pak neexistuje žádná další kolize s "menší" barvou.

## 3 Popis implementace

Program obsahuje zdrojové soubory `main.c`, který zajišťuje hlavní běh programu, `pgm.c` zajišťuje práci s `pgm` souborem a `matrix.c` zajišťuje uložení dat `pgm` souboru.

### 3.1 Kontrola a úprava argumentů

Při spuštění programu se musejí zadat dva argumenty, program tedy kontroluje zda byl zadán správný počet argumentů. Jelikož uživatel nemusí zadávat typ vstupního souboru (přípona `.pgm`) musí být zajištěno, aby se soubor při čtení našel. Program tedy k argumentu tuto příponu přidá pokud chybí pomocí funkce `strncat`.

## 3.2 Struktura pgm

Struktura pgm reprezentuje soubor pgm. Tato struktura uchovává informace o typu souboru (náš program přijíma pouze P5), šířku, výšku a kontrast obrázku a nakonec data, která jsou uložena ve struktuře matice 3.3.

### 3.2.1 Načítání ze souboru

Pro načítání dat ze souboru slouží funkce `read_file`, která kontroluje, zda přišel soubor ve správném formátu, dále ukládá informace o souboru (šířka, výška, kontrast) a poté se vytvoří matice a naplní se hodnotami. Hodnoty se do matice plní znak po znaku, zároveň se kontroluje, zda jsou data jen černobílá (0x00 nebo 0xFF). Pokud data nejsou černobílá, program skončí vyjímku.

### 3.2.2 Algoritmus obarvení dat

Algoritmus má za úkol funkce `data_coloring`. Jedná se o dvojprůchodový algoritmus 2.2. Pro rychlejší běh programu jsou pro první průchod vytvořeny tři cykly. První pro levý horní roh (tj. kontroluje se jen hodnota rohu), druhý pro první řádek kromě levého rohu (tj. kontroluje se jen prvek "na levo") a nakonec pro zbylá data, kde se musí kontrolovat všechna data podle masky. U posledního cyklu se musí také kontrolovat kolize, které se zaznamenávají do pole 2.3.3, neboť je to asi paměťově nevýhodnější. Pole má na začátku danou počáteční velikost a postupně se zvětšuje podle potřeby. Po dokončení prvního průchodu se provede průchod druhý, tedy přebarvení podle zaznamenaných kolizí.

### 3.2.3 Zaznamenávání kolizí

O zaznamenávání kolizí se stará funkce `detect_collision`, která prochází data matice podle masky obr.2. Do pole na zaznamenání kolizí se pak vždy ukládá nejmenší hodnota. Musí se však také kontrolovat, zda už v poli není uložena menší hodnota. Ukládání o kolizích se pak provádí tak, že se prochází celé pole o kolizích (`color_equivalence`) a hledají se hodnoty u kterých byla zjištěna kolize, jejich hodnota se pak mění na nejmenší zjištěnou hodnotu.

### 3.2.4 Vytváření souboru

Pro vytváření souboru slouží funkce `make_file`. Tato funkce vytváří pgm soubor dle stanoveného formátu 2.1. Data se násobí hodnotou 0x30 a ná-

sledně se do souboru uloží zbytek z dělení hodnotou `0xFF`, aby byly oblasti lépe rozeznatelné (pomocí realokování paměti).

### 3.3 Struktura matrix

Struktura matrix představuje matici, která je implementována jednorozměrným polem. Struktura dále uchovává údaj o počtu sloupců (šířce) a řádků (výšce). Dále jsou implementovány tři funkce, a to: `create_matrix` vytvářející matici, `print_matrix` vypisující matici a `free_matrix` uvolňující paměť.

## 4 Uživatelská příručka

### 4.1 Sestavení

Pro snadné sestavení je připraven `makefile`, který funguje na operačním systému Linux a v případě správné konfigurace i na Windows. Sestavení se provede po zavolání příkazu `make` v kořenovém adresáři. Předpokladem je nainstalovaný *GCC* a nástroj *make*.

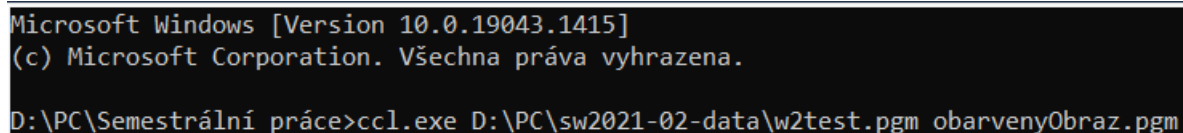
Pro operační systém Windows je připraven ještě poněkud typičtější `makefile.win`. Překlad lze provést příkazem `make -f makefile.win`. Předpokladem je nainstalovaný překladač *Microsoft C/C++* a nástroj *make*. Příkazy pro přeložení se mohou lišit podle použitého překladače a nastavení.

### 4.2 Spuštění

Program lze spustit pomocí příkazu:

```
ccl.exe <input-file[.pgm]> <output-file>
```

Kde `input-file` je cesta k pgm souboru, který obsahuje jen hodnoty `0x00` a `0xFF`. Při spuštění se může soubor zadat s příponou `.pgm` i bez. `output-file` pak obsahuje cestu a jméno výstupního souboru, který se může zadat s příponou `.pgm` i bez.



```
Microsoft Windows [Version 10.0.19043.1415]  
(c) Microsoft Corporation. Všechna práva vyhrazena.  
D:\PC\Semestrální práce>ccl.exe D:\PC\sw2021-02-data\w2test.pgm obarvenyObraz.pgm
```

Obrázek 3: Příklad spuštění



Formát vstupního i výstupního souboru je typu pgm (formát je detailně popsán viz.2.1). Přebarvení obrázku by mělo vypadat následovně:



Obrázek 4: Příklad vstupu



Obrázek 5: Příklad výstupu

## 5 Závěr

Byl vytvořen program na přebarvování souvyslých oblastí. Myslím, že z hlediska paměťové náročnosti se jedná o celkem slušné řešení, které se moc vylepšit nedá, avšak to samé nelze říci o rychlosti. Myslím, že je v mém programu přehnaný počet podmínek a cyklů, bez kterých by však program nefungoval správně. K zlepšení rychlosti by bylo potřeba se na problém podívat z jiného úhlu, a to hlavně na implementování jiného způsobu ukládání kolizí, při této implementaci by nejspíše utrpěla paměťová složitost. Další škraloup na mé práci je horší modifikovatelnost. Metody by možná mohli být kratší a obecnější. Například algoritmus by se dal rozdělit na první a druhý průchod, nebo vylepšit načítání souboru, aby šel lehce implementovat kód pro jiný typ pgm souboru (např. P2).

Celkovou práci hodnotím pozitivně, neboť jsem si vyzkoušel práci s jazykem C. Byl to pro mne nepopsatelný zážitek, který mě studijně obohatil a posunul o krok blíže k praktickým aplikacím teoreticky získaných vědomostí. Získal jsem prstoklad na klávesnici, jak kdybych hrál na klavír, každou noc jsem programoval, byl jsem jako netopýr. Myslím, že "Céčko" mi změnilo pohled na svět a můj život už nebude takový jako předtím. přinejmenším po každé když půjdu s holkou na rande se dívat na noční oblohu, uvidím místo hvězd miliony pointerů. Od té doby co umím aslespoň trochu programovat v tomto skvělém jazyce, si připadám, jako kdybych měl oproti normálním programátorům tajnou superschopnost. Takže, když se mne někdo ode dneška zeptá na znalost jazyků, řeknu, že umím v Cčku, javě, sql a basicu.

## 6 Reference

- [1] Podrobné zadání úlohy č.2, <http://www.kiv.zcu.cz/studies/predmety/pc/data/works/sw2021-02.pdf>, Kamil Ekštein, 2021
- [2] Connected-component labeling, [https://en.wikipedia.org/wiki/Connected-component\\_labeling](https://en.wikipedia.org/wiki/Connected-component_labeling), Wikipedia, Wikimedia Foundation, 2021