Warning

This page is located in archive. Go to the latest version of this course pages. Go the latest version of this page.

HW 09 - Načítání a ukládání grafu

Termín odevzdání	18.12.2021 23:59 PST
Povinné zadání	3+1b
Volitelné zadání	2b
Bonusové zadání	Není
Počet uploadů	10
Podpůrné soubory	b0b36prp-hw09.zip [/b211/_media/courses/b0b36prp/hw/b0b36prp-hw09.zip]

Povinné zadání

V této úloze je úkolem implementovat knihovnu pro načítání a ukládání datové struktury (grafu) z/do souboru a to jak v textové reprezentaci, tak v binární podobě. Cílem domácího úkolu je tak především procvičit si práci se soubory. Konkrétní datová struktura pro uložení grafu v programu není předepsána, ale lze využít struktury z 9. přednášky. Vstupní zadání grafu předepsané je a graf je zadán jako seznam hran, kde každá hrana je definována číslem počátečního vrcholu, číslem koncového vrcholu a dále pak celým číslem udávajícím cenu hrany. Všechny číselné hodnoty odpovídají rozsahu 32-bitového celého čísla, typ `int` .

Vlastní strukturu grafu (graph_t) lze implementovat libovolně, ale je potřeba dodržet předepsané funkce pro práci s grafem ze souboru `graph.h` b0b36prp-hw09.zip [/b211/_media/courses/b0b36prp/hw/b0b36prp-hw09.zip]. Implementaci požadovaných funkcí uložte do souboru `graph.c` . Pro domácí testování vašich implementací je nezbytné si vytvořit vlastní funkci `main()` v samostatném souboru (např. `main.c`), ale **odevzdávají se pouze soubory - `graph.h` a `graph.c` .** Načítání a ukládání grafu si můžete dále otestovat podobně jako se testuje program v Upload systému. Graf můžete vygenerovat například programem `graph_creator.c` , který je součástí demonstračních programů [/b211/_media/courses/b0b36prp/lectures/b0b36prp-lec11-codes.zip] z přednášky 9. V podpůrných souborech b0b36prp-hw09.zip [/b211/_media/courses/b0b36prp/hw/b0b36prp-hw09.zip] dále naleznete příklad testovacího skriptu spolu s programy využívajícími požadovanou implementaci funkcí předepsaných v `graph.h` pro

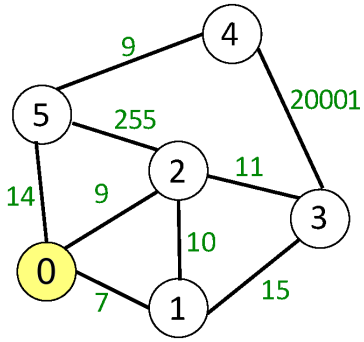
konverzi textového formátu do binárního a zpět. Podobný způsob je využit i pro testování vaší implementace v Upload systému.

Formát grafu

Graf je reprezentován jako seznam všech hran v grafu. Hrany grafu jsou definovány pomocí trojice celých čísel (počáteční vrchol, koncový vrchol, cena hrany).

Textový formát grafu

Každý řádek obsahuje tři celá čísla (jednu hranu) z rozsahu 32-bitového znaménkového integeru. Čísla jsou oddělená vždy jednou mezerou a za každou trojicí čísel je jeden znak nového řádku. Příklad textového zápisu grafu na obrázku



[/b211/_detail/courses/b0b36prp/hw/undirectedgraph1.png?id=courses%3Ab0b36prp%3Ahw%3Ahw09]

je

```
0 1 7
0 2 9
0 5 14
1 2 10
1 3 15
2 3 11
2 5 255
3 4 20001
4 5 9
```

Binární formát grafu

V této úloze můžete použít libovolný binární způsob uložení grafu. Jedinou podmínkou, vycházející z velikosti reprezentace celočíselného typu `int`, je, aby každé jednotlivé číslo v textovém formátu bylo reprezentováno nejvýše 4 byty v binárním formátu.

Vzhledem k zadání domácího úkolu HW 10 [/b211/courses/b0b36prp/hw/hw10], kde je formát uložení binárního souboru předepsán, doporučujeme při řešení HW09 inspirovat se následujícím způsobem binární reprezentace. Celočíselné hodnoty definující každou jednotlivou hranu grafu uložte ve stejném pořadí jako v textovém formátu, pouze zapište hodnoty typu `int` binárně. Datový typ `int` je v uvažovaných případech reprezentovaný 32-bitovou hodnotou, a proto je každé číslo

uloženo jako 4 byty. Jednoduše tak zapište přímo paměťový obraz celého čísla do souboru, např. funkcí `fwrite()` . Taková implementace sice bude závislá na konkrétní architektuře použitého procesoru počítače, ale pro potřeby řešení HW09 je plně postačující. Řešení nezávislé na pořadí bytů více-bytové reprezentace celých čísel, tzv. Endianita [https://cs.wikipedia.org/wiki/Endianita], je věnováno volitelné zadání úkolu HW 10 [/b211/courses/b0b36prp/hw/hw10].

Předepsané názvy funkcí "graph.h"

```
#ifndef __GRAPH_H__
#define __GRAPH_H__

typedef struct {
   // TODO - implement your own struct for graph
} graph_t;

/* Allocate a new graph and return a reference to it. */
graph_t* allocate_graph();
/* Free all allocated memory and set reference to the graph to NULL. */
void free_graph(graph_t **graph);

/* Load a graph from the text file. */
void load_txt(const char *fname, graph_t *graph);
/* Load a graph from the binary file. */
void load_bin(const char *fname, graph_t *graph);

/* Save the graph to the text file. */
void save_txt(const graph_t * const graph, const char *fname);
/* Save the graph to the binary file. */
===== Povinné zadání =====
void save_bin(const graph_t * const graph, const char *fname);

#endif // __GRAPH_H__
```

Volitelné zadání

Rychlost vašich řešení bude otestována manuálně mimo BRUTE až po termínu odevzdání. Načítání celých čísel funkcí `fscanf()` a zápis funkcí `fprintf()` sice představuje robustní a spolehlivý způsob, který řeší obecně formátované načítání a zápis, pro svou komplexnost však nepatří mezi nejrychlejší možné způsoby. V případě znalosti vstupu, tj. v našem případě pouze celá čísla v rozsahu 32-bitové typu `int` , můžeme načítání i zápis výrazně urychlit vlastní implementací, která nepředpokládá zpracování speciálních formátovacích požadavků. Proto se ve volitelné části úkolu HW09 pokuste implementovat vlastní načítání a ukládání grafu z/do textového souboru, například s využitím funkce "fgetc()" [https://www.tutorialspoint.com/c_standard_library/c_function_fgetc.htm]). Váš program bude testován na

rychlost načítání a ukládání a v případě, že nebude výrazně pomalejší (či dokonce bude rychlejší) než referenční program, získáte po jednom bodu navíc za rychlé načítání a rychlé ukládání. Navíc se vám zkušenost s rychlejším načítáním a ukládáním může hodit pro bonusové zadání domácího úkolu HW 10.

V případě, že při použití `getc()` nebo `fgetc()` je kód pomalejší než použitím `fscanf()` zkuste použít variantu bez zamykání přístupu do stream např. `getc_unlocked()` .

Odevzdávané soubory

Odevzdávejte pouze soubory

- `graph.h` - s povinnými rozhraním pro načítání a ukládání grafu dle výše uvedené definice
- `graph.c` - implementace rozhraní a funkcí pro načítání a ukládání grafu

Žádný soubor nesmí obsahovat `main()` . Všechny soubory uložte přímo do zip archivu a **nevytvářejte žádné složky**.

Testování programu

Povinné zadání

Rámcovou správnost implementace si můžete otestovat například tak, že načtete textový soubor s grafem, který uložíte do binární podoby. Do binárního souboru můžete nahlédnout například `hexdump -C` . Alternativně můžete zpětně načíst binární soubor a uložit jej v textové podobě. Takto vytvořený soubor by měl být identický s původním souborem, což můžete otestovat rozdílem souboru `diff` nebo otiskem `md5sum` . Testování přiloženým skriptem `test.sh` může například vypadat:

```
./test.sh
gmake: Nothing to be done for 'all'.
Create graph with 1000000 nodes
Load text graph g and save it to g.bin
Load txt file 'g'
Save bin file 'g.bin'
        1.03 real         0.94 user         0.08 sys
Load g.bin and save it to g.txt
Load bin file 'g.bin'
Save txt file 'g.txt'
        1.15 real         1.09 user         0.05 sys
Compare original txt file and the txt->bin->txt file
f09f7ed228c0c0f930ba609a94bed1ba  g
f09f7ed228c0c0f930ba609a94bed1ba  g.txt
```

V případě odevzdávání volitelného zadání na rychlost načítání a ukládání může výstup podobného (jen mírně komplexnějšího testu) vypadat například:

```
INFO: Create graph with 1000000 nodes
INFO: Student program passed txt->bin bin->txt test
INFO: Graph with 1000000 nodes load time ref: 215.50 ms (*2) student: 745.28 ms -
INFO: Graph with 1000000 nodes save time ref: 433.66 ms (*1.5) student: 756.69 ms
INFO: Create graph with 2000000 nodes
INFO: Student program passed txt->bin bin->txt test
INFO: Graph with 2000000 nodes load time ref: 460.05 ms (*2) student: 1517.08 ms -
INFO: Graph with 2000000 nodes save time ref: 908.87 ms (*1.5) student: 1542.61 ms
INFO: Create graph with 3000000 nodes
INFO: Student program passed txt->bin bin->txt test
INFO: Graph with 3000000 nodes load time ref: 704.30 ms (*2) student: 2261.59 ms -
INFO: Graph with 3000000 nodes save time ref: 1376.67 ms (*1.5) student: 2372.77 m
INFO: Student's program seems to be ok add 4 points
Total student score 4
```

Nebo v případě úspěšnější implementace:

```
INFO: Create graph with 1000000 nodes
INFO: Student program passed txt->bin bin->txt test
INFO: Graph with 1000000 nodes load time ref: 212.31 ms (*2) student: 213.44 ms -
INFO: Graph with 1000000 nodes save time ref: 445.03 ms (*1.5) student: 444.16 ms
INFO: Create graph with 2000000 nodes
INFO: Student program passed txt->bin bin->txt test
INFO: Graph with 2000000 nodes load time ref: 470.25 ms (*2) student: 508.12 ms -
INFO: Graph with 2000000 nodes save time ref: 977.12 ms (*1.5) student: 901.23 ms
INFO: Create graph with 3000000 nodes
INFO: Student program passed txt->bin bin->txt test
INFO: Graph with 3000000 nodes load time ref: 730.60 ms (*2) student: 712.80 ms -
INFO: Graph with 3000000 nodes save time ref: 1623.16 ms (*1.5) student: 1503.50 m
INFO: Student's program seems to be ok add 4 points
INFO: Student's program seems to be fast in loading add +1 for opt
INFO: Student's program seems to be fast in saving add +1 for opt
Total student score 6
```

Omezení použité paměti

Testovací instance jsou vygenerovány pomocí poskytnutého generátoru a alokovaná paměť je omezena následovně:

Instance	Velikost grafu	Maximum alokované paměti [b]
man01	1000	1000000
man02	10000	5000000

Varianty

	Povinné zadání	Volitelné zadání
Název v BRUTE	HW09	
Odevzdávané soubory	graph.h, graph.c	
Kompilace pomocí	clang -pedantic -Wall -Werror -std=c99 -O3	
Očekávaná časová složitost[1)]	$\mathcal{O}(E)$	
Procvičované oblasti	práce s textovými a binárními soubory	

1)

Kde E je počet hran.

courses/b0b36prp/hw/hw09.txt · Last modified: 2021/08/23 09:37 by faiglj