HW 10 - Integrace načítání grafu a prioritní fronta v úloze hledání nejkratších cest

Termín odevzdání	10.01.2025 23:59 PST Bonusová úloha - 11.01.2025 23:59 CET
Povinné zadání	3b
Volitelné zadání	3b
Bonusové zadání	2+[1)]
Počet uploadů	10 / (bonusová část bez omezení)

Cílem úlohy je integrovat existující řešení hledání nejkratší cesty (nejlevnější) cesty do všech uzlů orientovaného kladně ohodnoceného grafu a to pro jednoduchost vždy pouze z prvního uzlu grafu (tj. uzel číslo 0). Podrobnosti o úloze hledání cest jsou v přednášce 11 [/wiki/courses/b0b36prp/lectures/start], případně dále v popisu Dijkstrova algoritmu [https://cs.wikipedia.org/wiki/Dijkstr%C5%AFv_algoritmus]. V programu je použita implementace prioritní fronty haldou, která je reprezentována jako binární plný strom uložený v poli. Pro efektivní využití prioritní fronty v hledání nejkratších cest je však vhodné implementovat funkci `update()` .

Při řešení úlohy je možné vyjít z dostupné implementace b0b36prp-lec11-codes.zip [/wiki/_media/courses/b0b36prp/lectures/b0b36prp-lec11-codes.zip] a tím řešit integraci jako doplnění a úpravu existujícího řešení.

Povinná a volitelná část úlohy HW 10 není implementačně náročná. Nejnáročnější (až 90 %) může být nastudování a porozumění struktuře výchozího kódu, vhodné zvolení co je nutné doimplementovat a upravit, testování a výběr souborů pro upload. Nebuďte zaskočeni a věnujte dostatek času seznámení se s kódem.

Povinné zadání

Pro splnění povinného zadání lze vyjít ze zdrojových kódů 11. přednášky [/wiki/_media/courses/b0b36prp/lectures/b0b36prp-lec11-codes.zip]. Následující instrukce předpokládají využití těchto zdrojových kódů. Implementace bonusového zadání či procvičení může být vhodnější vyjít z vlastní implementace.

Pro řešení úlohy hledání nejkratších cest je předepsané rozhraní v hlavičkovém souboru `dijkstra.h` **.**

Dílčí úkoly

- V programu je použita implementace prioritní fronty haldou, která je reprezentována jako binární plný strom uložený v poli.
- Implementace je však naivní, proto implementaci upravete, aby algoritmus fungoval efektivně, tj. opravte/upravte implementaci funkce `pq_update()`, která se nachází v souboru `pq_heap-no_update.c`. Její složitost by v nejméně příznivém případě neměla být horší než O(log n), kde n je počet uzlů prohledávaného grafu.
- Zajistěte implementaci definovaného rozhraní `dijkstra.h` (implementovat vlastní nebo využít implementace z lec11), tj. implementovat všechny předpsané funkce v `dijkstra.h`.
- Kódy z přednášek obsahují soubory `lec11/djikstra.h` a `lec11/dijkstra.c`, ve kterých ale chybí funkce `dijkstra_set_graph()` a `dijkstra_get_solution()`, které se používájí pro testování v rámci sestavené binárky V BRUTE. Proto je nutné je doplnit. *Přiložené programy pro testování tyto funkce nepoužívají, musíte je otestovat sami.*
- Otestovat rychlost (a funkčnost) programu můžete využitím referenčního programu `tdijkstra`.
- Náhrat nezbytné soubory pro kompilaci a sestavení testovacího programu do BRUTE, tj. všechny soubory s implementací, ale bez funkce `main()`

Můžete si povšimnout, že některé proměnné v rámci kódů z přednášek by mohly mít deskriptivnější názvy. Proto, pokud Vám ty aktuální nevyhovují, doporučujeme dané proměnné přejmenovat poté, co identifikujete, co reprezentují, což je záměrně náročnější část úlohy.

Předepsané názvy metod (dijkstra.h)

```
#ifndef __DIJKSTRA_H__
#define __DIJKSTRA_H__

/*
 * Initialize structure for storing graph, solution, and eventual
 * variables for dijkstra algorithm
 *
 * return: point to the allocated structure; NULL on an error
 */
void* dijkstra_init(void);


/*
 * Load input graph (in text format) from the given file and store it
 * to the dijkstra structure previously created by dijkstra_init()
 *
 * return: true on success; false otherwise
 */
```

```
_Bool dijkstra_load_graph(const char *filename, void *dijkstra);


/*
 * Set the graph to the dijkstra structure instead of direct
 * loading the file as in dijkstra_load_graph() function.
 * The given array edges should not be directly used for the
 * internal representation of the graph in the dijkstra algorithm
 *
 * e - number of edges, i.e., size of the array
 *
 * return: true on success; false on an error, e.g., memory allocation
 */
_Bool dijkstra_set_graph(int e, int edges[][3], void *dijkstra);


/*
 * Solve the dijsktra algorithm on the graph previously
 * loaded by the dijkstra_load_graph() set by dijkstra_set_graph()
 * The solution is stored in some internal structure of the dijkstra
 * type passed to the function
 *
 * label - start node (0 for HW10)
 *
 * return: true on success; false otherwise
 */
_Bool dijkstra_solve(void *dijkstra, int label);


/*
 * Retrived the solution found by the function dijkstra_solve()
 * It is assumed the passed argument solution[][3] is properly allocated,
 * and thus the internal solution of the dijkstra can used to fill the
 * solution[][3].
 *
 * n - number of nodes, i.e., size of the array
 *
 * return: true on success; false otherwise
 */
_Bool dijkstra_get_solution(const void *dijkstra, int n, int solution[][3]);


/*
 * Directly solve the solution found by the dijkstra_solve() in to the
 * file (in the text format) with the given filename.
 *
 * return: true on success; false otherwise
 */
_Bool dijkstra_save_path(const void *dijkstra, const char *filename);
```

```
/*
 * Release on allocated memory of the passed structure for
 * the dijkstra algorithm, e.g., graph as an array of pointers to struct edge
 * and solution as an array of pointers to struct node.
 * All previosly allocated memory related to solution of the shortest
 * paths is freed
 */
void dijkstra_free(void *dijkstra);

#endif
```

V případě, že do nějakého uzlu cesta z uzlu 0 nevede, je hodnota délky nejkratší cesty -1 a podobně jako hodnota určující předcházejí uzel na nejkratší cestě (také -1).

Příklad vstupního a výstupního souboru

Vstup (graf)

Vstupní soubor s grafem obsahuje seznam hran, ve kterém je každá hrana uvedena na samostatném řádku a je specifikována třemi celými čísly v rozsahu 32-bitového typu `int` definující počáteční uzel a koncový uzel hrany spolu s cenou hrany. Vstup tak vypadá například tak jak je uvedeno níže. Z výpisu je patrné, že seznam hran je uspořádán a jsou nejdříve uvedeny hrany začínající v uzlech s postupně narůstajícím označením uzlu (indexem). Tato vlastnost je pro grafy v úkolu HW 10 garantována a lze ji s výhodou využít pro vnitřní reprezentaci uzlů a incidentních hran. Také je toho možné využít při načítání hran a konstrukci datové reprezentace grafu vhodného pro řešení úlohy hledání nejkratších cest.

```
0 1 4
0 3 85
1 0 74
1 2 18
1 4 12
2 1 12
2 5 74
2 9 12
3 4 32
3 6 38
4 3 66
4 5 76
4 7 33
5 9 21
5 8 11
6 7 10
6 3 12
```

```
7 6 2
7 8 72
8 7 18
8 5 31
8 9 78
9 5 8
```

Vstupní graf je definován jako seznam hran ve vstupním souboru, kde každá hrana je jeden řádek, tří `int` hodnot. A každý řádek je zakončen znakem nového řádku.

Výstup (nejkratší cesty)

Výstupní souborem je opět textový soubor, který obsahuje informaci o nejkratší cestě z počátačního uzlu (uzel číslo 0) do všech ostatních uzlů. Soubor obsahuje seznam uzlů, kde na každém řádku je trojice udávající číslo uzlu, cenu (déku) nejkratší cesty z uzlu 0 do příslušného uzlu a číslo bezprostředního předchůdce na nejkratší cestě z uzlu 0 do příslušného uzlu. Například z obsahu souboru

```
0 0 -1
1 4 0
2 22 1
3 63 6
4 16 1
5 42 9
6 51 7
7 49 4
8 53 5
9 34 2
```

je nejkratší cesta z uzlu 0 do uzlu 2 přes uzel 1 (řádek 2 22 1) s délkou 22, tj. délka (cena) hrany z uzlu 1 do uzlu 2 je 18. Dále například nejkratší cesta do uzlu 9 je přes uzel 2 a tím pádem také 1, tj. nejkratší cesta z 0 do 9 se skládá z hran propojující uzly 0, 1, 2, 9.

V případě neexistence cesty do daného uzlu jsou hodnoty délky cesty a bezprostředního předchůdce -1, např.

```
0 0 -1
1 4 0
...
14 -1 -1
...
```

Volitelné zadání

Volitelné zadání spočívá v doplnění kódu o načítání/ukládání grafu v předepsaném binárním formátu. Lze tedy vyjít z domácího úkolu HW 09 [/wiki/courses/b0b36prp/hw/hw09], kde se používá identické rozhraní definované v hlavičkovém souboru `graph.h`. V tomto případě však využijeme platformově nezávislou binární podobu souboru a budeme respektovat pořadí bytů Endianity [https://cs.wikipedia.org/wiki/Endianita]. Zajištění kompatibility mezi různými systémy je realizováno binárním zápisem hodnot celých čísel v tzv. Network order [https://en.wikipedia.org/wiki/Endianness], který odpovídá pořadí zápisu nejvíce významného bytu na nejnižší adresu (MSB) - Big-endian [https://en.wikipedia.org/wiki/Endianness].

Praktický přínos binárního formátu není v úloze hledání nejkratších cest příliš významný, proto zadání neklade požadavek na využití binárního formátu v úloze hledání nejkratších cest. Hlavní motivací volitelného zadání je integrace dalších funkcí a případně modulů do zdrojových souborů, které řeší jak povinnou, tak volitelnou část.

Dílčí úkoly

- Implementovat rozhraní definované v `graph.h` (níže), např. s využitím vlastní implementace úlohy HW 09.
- V implementaci zajistit uložení / načtení grafu v definované binární podobě.
- Zvolit nezbytné soubory s implementací načtení/uložení grafu tak, aby bylo možné zdrojové soubory začlenit do implementace a odevzdání povinné části HW 10.
- Otestovat funkčnost implementace před odevzdáním v BRUTE..
- Odevzdat volitelné zadání v BRUTE.

Definice rozhraní a binárního formátu

```
#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. */
void save_bin(const graph_t * const graph, const char *fname);

#endif // __GRAPH_H__
```

V tomto případě však využijeme platformově nezávislou binární podobu souboru a budeme respektovat pořadí bytů (Endianity [https://cs.wikipedia.org/wiki/Endianita]). Pro zajištění kompatibility mezi různými systémy proto bude pro binární zápis hodnoty celých čísel použit tzv. Network order [https://en.wikipedia.org/wiki/Endianness], který odpovídá pořadí zápisu nejvíce významného bytu na nejnižší adresu (MSB) - Big-endian [https://en.wikipedia.org/wiki/Endianness].

Pravděpodobně nejrozšířenější architektura Intel x86/AMD64/x86-64 používá Little-endian nicméně i tak je vhodné detekovat konkrétní architekturu v programu a podle toho zapisovat celá čísla ve správném pořadí bytů. Způsobů jak toho dosáhnout je několik. Jedním z nich je využití funkcí `htonl()` a `ntohl()` z knihovny `<netinet/in.h>` . Další možností je inspirovat se přístupem One-line endianness detection in the C preprocessor [http://esr.ibiblio.org/?p=5095] a detekovat architekturu při překladu a následně využít konverze například prohozením bytů ve 32 bitovém typu `int` makrem.

Příklad

Celé číslo o hodnotě 20001 odpovídá binární reprezentaci $100111000100001_{\text{bin}}$ a v šestnáctkovém zápise $4e21_{\text{hex}}$. Pokud rozdělíme takové číslo na jednotlivé byty, jsou tyto byty v paměti uloženy buď od nejvýznamnějšího k nejméně významnému bytu, nebo naopak. Řazení bytů Network order [https://en.wikipedia.org/wiki/Endianness] je vlastně Big-endian [https://en.wikipedia.org/wiki/Endianness], které odpovídá řazení nejvýznamnějšího bytu na nejnižší adresu. Pro vstupní graf v textovém formátu

```
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
```

odpovídá požadovaný binární formát (získaný například z výpisu `hexdump -C`):

```
0000000: 00 00 00 00 00 00 00 01  ........
0000008: 00 00 00 07 00 00 00 00  ........
0000010: 00 00 00 02 00 00 00 09  ........
0000018: 00 00 00 00 00 00 00 05  ........
0000020: 00 00 00 0e 00 00 00 01  ........
0000028: 00 00 00 02 00 00 00 0a  ........
0000030: 00 00 00 01 00 00 00 03  ........
0000038: 00 00 00 0f 00 00 00 02  ........
0000040: 00 00 00 03 00 00 00 0b  ........
0000048: 00 00 00 02 00 00 00 05  ........
0000050: 00 00 00 ff 00 00 00 03  ........
0000058: 00 00 00 04 00 00 4e 21  ......N!
0000060: 00 00 00 04 00 00 00 05  ........
0000068: 00 00 00 09              ....
```

Pro připomenutí jak zapsat celé číslo binárně, tak stačí přetypovat příslušnou proměnnou ukazatele s adresou hodnoty celého čísla typu `int`, například jako ukazatel na hodnoty typu char (byte):

```
int a = 20001, *p1 = &a;
char *p2 = (char *)p1;
char byte1 = *(p2);
char byte2 = *(p2+1);
...
```

Získání jednotlivých bytů lze také realizovat bitovými operacemi, např.

```
int a = 20001;
char byte1 = (char)(p1 & 0xff);
char byte2 = (char)((p1>>8) & 0xff);
...
```

Uvedené možnosti reprezentují dva z mnoha způsobů jak úkol řešit a můžete také navrhnout řešení jiné, vlastní.

Bonusové zadání - soutěž

Statistiky výpočetních časů studentských a referenčních programů HW10(b)
Průběžné a následně finální (po uzavření bonusového zadání) výsledky studentských implementací a referenčních programu řešení hledání nejkratší cesty jsou k nahlédnutí po přihlášení ve výpočetní náročnost odevzdaných a referenčních programů [https://cw.felk.cvut.cz/brute/data/ae/release/2024z_b0b36prp/b0b36prp-ae/web/hw10impls.html].
Odevzdáním implementovaného hledání nejkratší cesty v rámci bonusového zadání přihlašujete

svůj program do soutěže o nejrychlejší implementaci. Průběžné a zejména výsledné hodnocení proběhne v chráněném výpočetním prostředí, tak aby všechny programy měly k dispozici identické zdroje bez vnějších vlivů. Závěrečné hodnocení bonusových úloh a určení pořadí nejrychlejších implementací proběhne v posledním týdnu semestru. Termín odevzdání bonusové úlohy je proto fixní a po termínu již nebude možné program do soutěže přihlásit a tedy ani odevzdat řešení bonusové části úkolu.

V implementaci není doporučeno používat specifických nekompatibilních funkcí (např. z glibc). Použijete funkce knihovny C99 nebo POSIX C [https://en.wikipedia.org/wiki/POSIX#POSIX.1-2008_(with_two_TCs)] (POSIX.1-2008/IEEE Std 1003.1-2008/). Programy se v BRUTE kompilují s `-std=c99 -D_POSIX_C_SOURCE=200809L` .
Součástí vyhodnocení implementace je textové načítání, samotné řešení hledání nejkratší cesty, a ukládání řešení v textovém formátu. Binární načítání a ukládání není v HW10B využíváno.
Testovací grafy jsou v řádu milionů uzlů a desítek milionů hran.
Tipy:

- Nahrazení funkcí `scanf()` a `print()` nějakou vlastní implementací je pro snížení výpočetních nároku nezbytné, tj. řešení volitelné části HW09.
- Při načítání lze chytře předalokovávat paměť na základě znalosti organizace vstupního souboru a *předpokladu, že uzly v grafu mají v průměru nějaký počet hran*.
- Správné použití prioritní fronty s využitím haldy je také nezbytným předpokladem pro efektivní řešení.
- Dalšího zrychlení lze dosáhnout laděním volání funkcí a vhodným paměťovým modelem.

Hodnocení

Každý program přihlášený do soutěže, který bude srovnatelný (rychlejší než zvolený násobek času referenčního programu) s referenční implementací získá **4 body**.

Návrhy možného bodového hodnocení za bonusovou část
A. Dále si pak prvních 10 nejrychlejších programů rozdělí 20 dalších bodů. Pořadí však nebude stanovené absolutně podle časů, ale podle nejlepšího času s nějakým definovaným okolím (např. +/- 100 ms), proto může být nejrychlejších programů více. Např. pokud bude nejrychlejší program řešit danou úlohu v čase 500 ms a jiný program v čase 505 ms, budou oba programy ohodnoceny jako nejrychlejší programy a získají 6 bodů za rychlost a v případě, že budou rychlejší než referenční program, tak každý získá další 4 body, tj. celkem 10 bodů.

B. Prvních 100 nejrychlejších programů si rozdělí 100 bodů.

C. Bodové hodnocení na základě detekovaných milníků dosažených časů studentských řešení.

Rozhodnutí o způsobu ohodnocení provedeme na základě studentských implementací, ve většině případů to je varianta C.

Odevzdávané soubory

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

Povinné soubory:

- `dijkstra.h` - s povinnými rozhraním definovaným výše
- `graph.h` - (**Pouze pro volitelné zadání!**) s povinným rozhraním pro načtení/uložení grafu do/z textového/binárního formátu viz hw09 [/wiki/courses/b0b36prp/hw/hw09]
- Další soubory s implementací dle vlastního návrhu nebo viz doporučené soubory níže.

Doporučené soubory s implementací:

Například založené na souborech z přednášky 11 [/wiki/_media/courses/b0b36prp/lectures/b0b36prp-lec11-codes.zip]

- `dijsktra.c` - implementace hledání nejkratší cesty dle výchozího `dijkstra.h.`.
- `graph.h` - definice grafu.
- `graph_utils.h` - definice rozhraní pro práci s grafem (včetně načítání a ukládání).
- `graph_utils.c` - implementace rozhraní (funkcí) pro práci s grafem.
- `pq_heap.h` - definice rozhraní prioritní fronty realizované haldou.
- `pq_heap.c` - implementance prioritní fronty haldou (binárním plným stromem) reprezentované v poli.
- `my_malloc.h` - deklarace rozhraní pro vlastní alokaci paměti.
- `my_malloc.c` - implementace rozhraní pro vlastní alokaci paměti.
- `load_simple.h` - deklarace jednoduché funkce pro načítání grafu.
- `load_simple.c` - implementace jednoduché funkce pro načítání grafu.
- `modules.mk` - soubor specifikující pořadí jednotlivých modulů pro sestavení (linkování) výsledného binárního programu. Pro uvedené doporučené moduly (.c soubory) může mít tvar

```
SRC=dijkstra.c graph_utils.c load_simple.c pq_heap.c my_malloc.c
```

Testování řešení

Správnost řešení nejkratších cest lze ověřit na nějakém vstupním grafu a porovnáním se správným řešením. Protože záludné chyby se mohou projevit až pro velké grafy, je součástí balíku b0b36prp-lec11-codes.zip [/wiki/_media/courses/b0b36prp/lectures/b0b36prp-lec11-codes.zip] program `tdijkstra`, který umí vygenerovat jak náhodný graf o zadaném počtu hran, tak nalézt

řešení nebo zkontrolovat zdali je řešení uložené v nějakém souboru řešením správným. Program používá vlastní implementací generátoru náhodných čísel, proto při zadání specifického inicializačního semínka bude generovat vždy stejný náhodný graf a to i na různých platformách. Program je k dispozici v binární podobně v pěti variantách:

- `tdijkstra-lnx`
- `tdijkstra-fbsd`
- `tdijkstra.exe`
- `tdijkstra-osx-intel`
- `tdijkstra-osx-m1`

Dále je součástí balíku program `timeexec.exe`, který pracuje podobně jako příkaz `time` a spustí zadaný program s parametry jako argumentem a vrátí čas běhu programu.

Použití programu `tdijkstra` je následující:

```
# Vytvoření náhodného grafu o 1000 vrcholech a uložení do souboru ''g''
./tdijkstra -c 1000 g

# Vytvoření náhodného grafu o 2000000 vrcholech s inicializaci generátoru náhodr
./tdijkstra -c 2000000 -s 123 g

# Řešení úlohy hledání nejkratší cesty z prvního uzlu (uzel číslo 0) do všech os
./tdijkstra g s

# Řešení úlohy hledání spolu s výpisem času potřebného pro řešení dílčích částí
./tdijkstra -v g s
Dijkstra version 2.3.4
Load time ....349ms
        Init time ....10ms
        Find time ....0ms
Solve time ...10ms
Save time ....184ms
Total time ...545ms

# Kontrola správnosti řešení uloženého v souboru ''s'' pro graf ''g'' (program r
./tdijkstra -t g s
echo $?
0

# Kontrola správnosti řešení s informativním (verbose) výstupem
./tdijkstra -v -t g s
Dijkstra version 2.3.4
Load graph time ......355ms
        Init time ....9ms
        Find time ....1ms
```

```
Solve time ...........10ms
Load solution time ...328ms
Compare time..........5ms
Total time ...700ms
Solution is OK
```

Stejným programem je testován program v Upload systému.

Testování vlastní implementace

Testování vlastní implementace je možné voláním implementovaných funkcí dle hlavičkového souboru `dijkstra.h`. S využitím zdrojových souborů b0b36prp-lec11-codes.zip [/wiki/_media/courses/b0b36prp/lectures/b0b36prp-lec11-codes.zip] je možné pro vytvoření řešení s vlastní implementací hledání nejkratší cesty využít program `tgraph_search`, který se nachází v adresáři `b0b36prp-le11-codes/graph_search`. Z tohoto pohledu je vhodné implementovat vlastní řešení právě na základě těchto zdrojových souborů případně využít volání v `tgraph_search.cc`

```
#include "dijkstra.h"

#include <stdio.h>

int main(int argc, char *argv[])
{
   int ret = 0;

   if (argc < 3) {
      fprintf(stderr, "Call as\n %s graph_file solution_file\n", argv[0]);
   } else {
      fprintf(stderr, "Load graph from %s\n", argv[1]);
      void *dij = dijkstra_init();
      dijkstra_load_graph(argv[1], dij);
      fprintf(stderr, "Find all shortest paths from the node 0\n");
      dijkstra_solve(dij, 0);
      fprintf(stderr, "Save solution to %s\n", argv[2]);
      dijkstra_save_path(dij, argv[2]);
      fprintf(stderr, "Free allocated memory\n");
      dijkstra_free(dij);
      ret = 0;
   }
   return ret;
}

/* end of tgraph_search.c */
```

V kombinaci s programem `tdijkstra` může být použití například následující.

```
#Vytvoření testovacího grafu o 1000000 uzlech do soubor g (relativní volání z b(
% ../bin/tdijkstra -c 1000000 g

#kompilace vlastního řešení a linkování s tgraph_search
make

#spuštění program bez argumentu(ů) vyvolá nápovědu
% ./tgraph_search
Call as
 ./tgraph_search graph_file solution_file

#spuštění programu s vstupním grafem v souboru g a uložení řešení do souboru s
% ./tgraph_search g s
Load graph from g
Find all shortest paths from the node 0
Save solution to s
Free allocated memory

# Otestování správnosti řešení programem tdijkstra (přepínač -t) s výpisem časů
% ../bin/tdijkstra -t -v g s
Load graph time ......187ms
        Init time ....5ms
        Find time ....438ms
Solve time ...........443ms
Load solution time ...256ms
Compare time..........3ms
Total time ...890ms
Solution is OK

#vypsání návratové hodnoty posledního spuštěného programu, kde program indikuje
% echo $?
0

#
```

Kompilace a sestavení programu v US

Zadání úkolu pouze specifikuje rozhraní (funkce) pro načtení grafu, řešení úlohy hledání nejkratších cest, uložení nalezených cest a uvolnění paměti. Rozhraní je definováno v souboru `dijkstra.h` a kromě těchto funkcí lze definovat libovolné další funkce potřebné pro řešení úlohy. Podobně rozdělení na moduly může být řešeno různě. Jedinou podmínkou odevzdávaných souborů je, že žádný odevzdávaný zdrojový soubor neimplementuje hlavní funkci programu `main`. Ovšem pro účely testování takový soubor potřebujete, jen jej nebudete odevzdávat. Z tohoto důvodu jsou odevzdané soubory překládany a linkovány předpisem `Makefile` pro GNU

Make uvedeným níže. Skript je napsán tak, aby využíval automatické detekce zdrojových souborů .c, ale zároveň umožňoval specifikovat konkrétní pořadí linkovaných objektových souborů (.o). V případě, kdy je nutné explicitně uvést pořadí tak, aby při sestavení byly postupně zjišťovány jednotlivé funkce definované v ostatních modulech, je možné využít předpisu pořadí zdrojových souborů v souboru `modules.mk`, který je načítán před automatickou detekcí zdrojových souborů.

```
uniq = $(if $1,$(firstword $1) $(call uniq,$(filter-out $(firstword $1),$1)))

-include modules.mk

TARGET=tgraph_search

CFLAGS+=-std=c99 -O3 -march=native -pedantic -Wall -Werror -D_POSIX_C_SOURCE=20(
LDFLAGS+=-lm

SRC:=$(TARGET).c $(SRC)
SRC+=$(wildcard *.c)

OBJS:=$(patsubst %.c,%.o,$(call uniq,$(SRC)))

bin: $(TARGET)

$(OBJS): %.o: %.c
	$(CC) -c $< $(CFLAGS) $(CPPFLAGS) -o $@

$(TARGET): $(OBJS)
	$(CC) $(OBJS) $(LDFLAGS) -o $@

clean:
	$(RM) $(OBJS) $(TARGET)
```

Před odevzdáním si prosím zkontrolujte, že archiv neobsahuje **Makefile** a žádný soubor v archivu neobsahuje funkci `main()`.

	Povinné zadání	**Volitelné zadání**	**Bonusové zadání (soutěž)**
Název v BRUTE	HW10		HW10B
Odevzdávané soubory	*.h, *.c (žádný soubor neobsahuje implementaci hlavní funkce `main`), implementace rozhraní dle `dijkstra.h`, volitelně modules.mk s definicí pořadí linkování		
Kompilace pomocí	Makefile + clang s `-std=c99 -D_POSIX_C_SOURCE=200809L`		

	Povinné zadání	Volitelné zadání	Bonusové zadání (soutěž)
Očekávaná časová složitost[2]	$\mathcal{O}(E + Vlog(V))$		
Procvičované oblasti	binární halda	integrace zdrojových kodů	rychlost
Výstup programu na `stdout`	Pokud možno žádný		Žádný

1)
Soutěž o body - podmínky budou nastaveny na základě průběhu semestru před zadáním domácí úlohy.
2)
Kde V je počet vrcholů a E je počet hran.

courses/b0b36prp/hw/hw10.txt · Last modified: 2025/01/09 14:34 by faiglj

Copyright © 2025 CTU in Prague | Operated by IT Center of Faculty of Electrical Engineering | Bug reports and suggestions Helpdesk CTU