Zadání projektu do předmětu **Algoritmy II**
zimní semestr 2025/2026
prezenční a kombinované studium

Historie modifikací

29. října 2025 První verze

Obsah

Deadline

Vypracované řešení projektu je nutno odevzdat do

- prezenční forma studia – **7. prosince 2025 23:59** a
- kombinovaná forma studia – **14. prosince 2025 23:59**.

Obecné pokyny

- Každý student či studentka má přiděleno jedno zadání. Rozdělení zadání mezi studenty je součástí tohoto dokumentu.
- S případnými dotazy kontaktujte svého cvičícího (studenti prezenční formy studia) či tutora (studenti kombinované formy studia).
- Termíny obhajob budou vypsány v systému Edison.
- Deadline je konečný a nebude dále posunován. Na projekty odevzdané po tomto datu nebude brán zřetel. Projekt lze pochopitelně odevzdat a domluvit si termín obhajoby i dříve.
- Každý cvičící (tutor) Vám sdělí, jakým způsobem budete projekt odevzdávat – pomocí git repozitáře, emailem, uložením na sdílené úložiště, systému Kelvin a tak podobně. Obecně platí, že se odevzdávají soubory se zdrojovým kódem, hlavičkové soubory, soubory s projektem atd., jinak řečeno vše, co je potřebné pro bezproblémovou kompilaci odevzdaného projektu a nic navíc.
- Součástí zdrojových kódů Vašeho programu bude programátorská dokumentace ve formě dokumentačních komentářů, zpracovatelných programem Doxygen, viz `www.doxygen.org`. Vygenerovanou dokumentaci není nutné odevzdávat. Postačuje, pokud Vámi odevzdaný archiv bude obsahovat konfigurační soubor `doxyfile`, případné adresáře pro vygenerovanou dokumentaci, vkládané obrázky atd.
- Ačkoliv se zadání mohou jevit na první pohled složitá, nepropadejte panice. Vyřešení kteréhokoliv zadání by nemělo zkušenému programátorovi zabrat více než „jeden večer“. Studentům prvního a druhého ročníku to zabere asi více času, ale v žádném případě by řešení nemělo trvat „desítky a desítky“ člověkohodin práce. Stejně tak co se týče délky vytvořeného kódu. Pokud jste už napsali „tisíce“ řádků kódu a stále není konec v dohledu, je to špatně. Takový zdrojový kód rovnou zahoďte. Klíčem k řešení je tužka, papír a hlava. Zkuste si řešení nejdříve promyslet, kreslit si u toho různá schémata, náčrtky datových struktur, volání funkcí a tak dále. Projděte si literaturu. A až se dostaví „aha moment“ tak začněte psát kód.

Hodnocení projektů

Kritéria hodnocení řešení projektů jsou tato:

1. **Správnost řešení** Správnost řešení je podmínka nezbytná. Aplikace, která nebude poskytovat správné výsledky, bude hodnocena automaticky 0 body bez ohledu na další kritéria. Ke každému zadání jsou k dispozici testovací data a výsledky, lze si tedy ověřit správnost řešení.
2. **Volba vhodných datových struktur** Toto kritérium hodnotí, v závislosti na konkrétním zadání, volbu vhodné datové struktury a algoritmů pro manipulaci se zvolenou datovou strukturou.
3. **Dekompozice problému na menší celky**
   - V předmětu Algoritmy I je požadována procedurální dekompozice, čili dekompozice řešení do funkcí. Využití objektově orientovaného programování je v tomto předmětu dobrovolné.
   - V předmětu Algoritmy II je požadován objektový návrh řešení a tomu odpovídající implementace.

4. **Způsob implementace** Toto kritérium hodnotí oddělení deklarace a definice funkcí nebo tříd do h a cpp souborů, využívání konstant místo přímo zapsaných hodnot, využívání parametrů funkcí a výsledků funkcí místo využívání side efektu založeném na globálních proměnných. Dále sem patří i úroveň zápisu zdrojového kódu, například odsazování vnořených konstrukcí, vhodné pojmenování proměnných, funkcí, tříd, dodržování zvolené konvence pojmenování[1] proměnných, funkcí a tříd, dodržování zvolené konvence psaní bloků kódu[2] a tak dále.

5. **Efektivita implementovaného algoritmu** Smyslem tohoto kritéria není nutit vás k implementaci nejlepšího známého algoritmu tím nejlepším možným způsobem. Tímto kritériem si vyučující ponechávají prostor pro případné snížení bodového hodnocení za použití algoritmu zcela nevhodného, nesmyslného, zmateného. *Příklad*: Součástí řešení projektu je třídění pole či vektoru s n prvky. Pokud použijete algoritmus se složitostí $O(n \log_2 n)$ je vše v pořádku. Pokud použijete některý z jednodušších algoritmů se složitostí $O(n^2)$, asi vás vyučující při obhajobě upozorní, že to nebyla dobrá volba, ale stále je to v pořádku. Za zcela nesmyslný algoritmus je v tomto případě považován algoritmus se složitostí větší než $O(n^2)$, protože takový algoritmus z podstaty věci dělá zbytečnou práci.

6. **Dokumentace k projektu** Ke každé funkci, třídě, atributu třídy musí existovat alespoň krátký dokumentační komentář ve formátu zpracovatelném programem Doxygen. Pro zápis dokumentačních komentářů lze využít libovolný z formátů, které Doxygen podporuje. Vygenerovanou dokumentaci není nutné odevzdávat, ale je nutné odevzdat konfigurační soubor `doxyfile`, aby bylo možné dokumentaci bez problémů vygenerovat.

   K programu Doxygen existuje rozsáhlá dokumentace volně dostupná na URL `https://www.doxygen.nl/manual/index.html`. Pro dokumentaci řešení semestrálního projektu je vhodné si prostudovat následující části dokumentace:

   - „Getting started“, `https://www.doxygen.nl/manual/starting.html`,
   - „Documenting the code“, `https://www.doxygen.nl/manual/docblocks.html`
   - „Doxygen usage“, `https://www.doxygen.nl/manual/doxygen_usage.html`
   - „Doxywizard usage“, `https://www.doxygen.nl/manual/doxywizard_usage.html`
     Program Doxywizard slouží k pohodlnému vygenerování konfiguračního souboru `doxyfile`, který tak není nutné vytvářet ručně.

7. **Citace zdrojů** Žádný program nevzniká úplně z ničeho, ani řešení semestrálních projektů. K řešení projektů můžete používat odbornou literaturu, učebnice, příklady zdrojových kódů z ostatních předmětů, internetové zdroje. V tom případě je nutné uvést, že „Tento algoritmus jsem převzal z…“, „Tuto část kódu jsem převzal z…“. Tyto případné citace umístěte do dokumentačních komentářů. Asi není nutné citovat Levitinovu knihu, že jsem si tam „přečetl něco o průchodu grafem do šířky“ nebo, že „toto jsme řešili na cvičení“. Ostatní zdroje by se však citovat měly.

[1] Typicky camelCase, PascalCase, méně vhodná je už například maďarská notace.
[2] Typicky – složená levá závorka za příkazem nebo na novém řádku.

4 Závislosti zdrojových kódů

Problém

V tomto zadání máte za úkol řešit kompilaci zdrojových kódů rozsáhlého projektu. Takový projekt se obvykle skládá z mnoha a mnoha souborů se zdrojovým kódem. U projektů implementovaných v jazyce C++ jde typicky o `cpp` soubory a do nich vkládané hlavičkové soubory s příponou `h`. Představme si nyní, že jsme v roli vývojáře pracujícího na projektu, provedli změnu v jednom ze souborů projektu. Pokud chceme projekt po této změně zkompilovat, abychom získali aktuální verzi, máme dvě možnosti:

1. zkompilovat úplně všechny zdrojové kódy, což ale může být docela zdlouhavý proces, nebo
2. zkompilovat jen změněné části projektu.

a.cpp

```
#include "a.h"
```

b.cpp

```
#include "b.h"
```

c.cpp

```
#include "c.h"
```

a.h

```
#include <iostream>
#include <vector>
#include "b.h"
```

b.h

```
#include <fstream>
#include <vector>
```

c.h

```
#include <cstdlib>
#include <iostream>
#include "a.h"
```

iostream

```
#include <ios>
```

fstream

```
#include <ios>
```

Obrázek 9: Zdrojové a hlavičkové soubory, jednoduchý příklad. Hlavičkový soubor `ios`, zde není uvedený, protože se už neodkazuje na další soubory.

Pokud chceme kompilovat jen nezbytně nutné části projektu, čili jen některé zdrojové kódy, musíme znát závislosti mezi jednotlivými soubory se zdrojovým kódem. Závislost vznikne například pomocí direktivy `#include`. Tyto závislosti můžeme reprezentovat orientovaným grafem G, kde vrcholy grafu reprezentují jednotlivé soubory a hrany reprezentují závislosti souborů. Vede-li například orientovaná hrana z vrcholu u do vrcholu v, znamená to, že soubor u závisí na souboru v. Z tohoto grafu můžeme ale také určit, které soubory musíme znovu zkompilovat, když provedeme změnu v souboru reprezentovaném nějakým vrcholem w.

Vaším úkolem je implementovat algoritmus, který pro zadaný graf závislostí mezi soubory G a daný soubor (vrchol grafu) v určí, které soubory jsou změnou souboru v ovlivněny a musí se znovu zkompilovat. Následně tento algoritmus spusťte pro všechny vrcholy v grafu G.

Ukázkový příklad

Pro ukázku předpokládejme, že máme tři soubory se zdrojovým kódem – `a.cpp`, `b.cpp` a `a.cpp`. Každý z těchto zdrojových kódů má svůj vlastní hlavičkový soubor – `a.h`, `b.h` a `a.h`. Hlavičkové soubory jsou navzájem dále provázány pomocí direktivy `#include` a dále se tyto hlavičkové soubory odkazují na hlavičkové soubory ze standardní knihovny, například `fstream`.

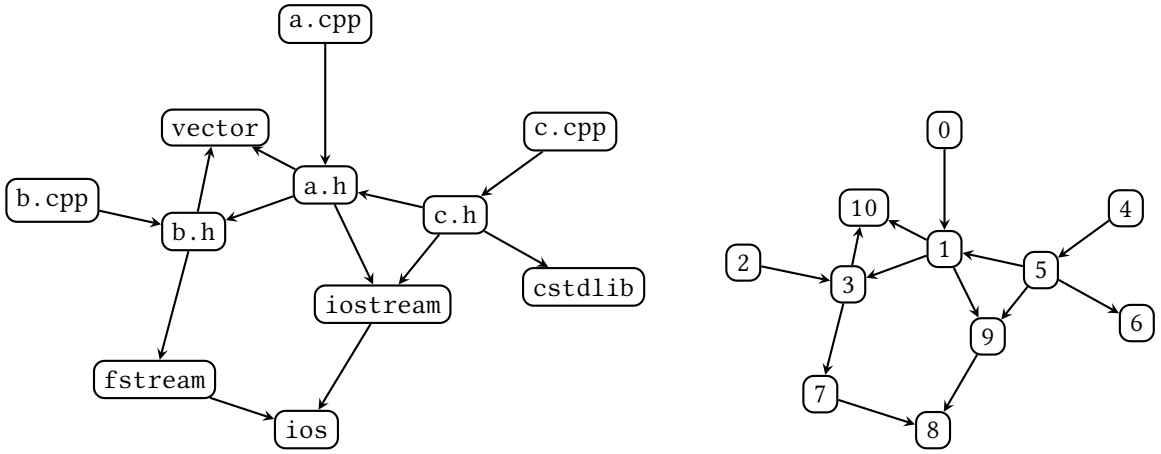
Obsah ukázkových zdrojových a hlavičkových souborů můžeme vidět na obrázku 9. Veškerý obsah jak zdrojových, tak hlavičkových souborů je nezajímavý a je pro přehlednost zcela vynechán. Co je ale na ukázkových souborech důležité je vzájemná provázanost mezi soubory daná direktivami `#include`. Z obrázku tak plyne, že soubor `a.cpp` se odkazuje na soubor `a.h` a ten se odkazuje na soubory `iostream`, `vector` a `b.h`. A tyto soubory se odkazují na další soubory a další soubory. Výsledkem je tak graf závislostí mezi soubory G zobrazený na obrázku 10a. Pro zjednodušení zpracování v počítači můžeme jména souborů nahradit celočíselnými identifikátory pomocí tabulky 7. Použijeme-li tyto identifikátory místo jmen souborů dostáváme graf na obrázku 10b. Výsledky, tj. které soubory musíme znovu zkompilovat při změně daného souboru, jsou uvedeny v tabulce 8.

Graf závislostí pro větší skupinu souborů je uveden na obrázku 11.

Poznámky

- Vstupem do vašeho programu bude textový soubor obsahující specifikaci grafu závislostí souborů v pomyslném projektu. Vrcholy grafu jsou označeny nezápornými celými čísly. Graf je uložen ve formě seznamu hran. Na každém řádku vstupního textového souboru je uložena jedna hrana grafu ve tvaru `u -> v`, kde `u`, `v` jsou vrcholy grafu a orientovaná hrana vede z vrcholu `u` do vrcholu `v`. Graf z obrázku 10b tak bude uložen ve tvaru:

  ```
  0 -> 1
  ```



(a) jména souborů (b) číselné identifikátory

Obrázek 10: Jednoduchá ukázka závislosti zdrojových kódů

Soubor	Id
a.cpp	0
a.h	1
b.cpp	2
b.h	3
c.cpp	4
c.h	5

Soubor	Id
cstdlib	6
fstream	7
ios	8
iostream	9
vector	10

Tabulka 7: Soubory a jejich číselné identifikátory

Modifikovaný soubor	Nutná kompilace souborů
0	0
1	0, 1, 4, 5
2	2
3	0, 1, 2, 3, 4, 5
4	4
5	4, 5
6	4, 5, 6
7	0, 1, 2, 3, 4, 5, 7
8	0, 1, 2, 3, 4, 5, 7, 8, 9
9	0, 1, 4, 5, 9
10	0, 1, 2, 3, 4, 5, 10

Tabulka 8: Výsledky pro soubory a jejich závislosti zobrazené na obrázku 10

```
2 -> 3
4 -> 5
1 -> 3
1 -> 10
1 -> 9
3 -> 10
3 -> 7
5 -> 9
5 -> 6
5 -> 1
9 -> 8
7 -> 8
```

- Výsledek vypište ve vhodné formě na standardní výstup.