Zadání projektu do předmětu **Algoritmy II**
zimní semestr 2026/2027
prezenční a kombinované studium

Historie modifikací

29. října 2026 První verze

Obsah

Obecné pokyny 2

Odevzdání projektu 2

Kritéria hodnocení projektů 3

Prohlášení o využití generativní AI 5

1 Makroinstrukce pro procesor Kvark 7

2 Síť chodeb pod městem Er 12

3 Minimalizace počtu popelnic 14

4 Výšková mapa 18

5 Sobí bludiště 19

6 Parkety 22

Rozdělení zadání mezi studenty 24

Prezenční forma studia 24

Kombinovaná forma studia 34

Deadline

Vypracované řešení projektu je nutno odevzdat do

- prezenční forma studia – **6. prosince 2026 23:59** a
- kombinovaná forma studia – **13. prosince 2026 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.
- 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.

Odevzdání projektu

Odevzdané řešení projektu musí obsahovat následující části:

- **Zdrojové kódy v C++** Pouze soubory nezbytné pro kompilaci (zdrojové kódy, hlavičky, projektové soubory).
- **Technickou dokumentaci** Vygenerovanou dokumentaci vůbec neodevzdávejte, postačí odevzdání konfiguračního souboru doxyfile, který umožní dokumentaci vygenerovat.
- **Prohlášení o využití AI** Vyplněný formulář, kde student deklaruje rozsah zapojení nástrojů jako ChatGPT nebo Copilot a popíše, jak ověřil správnost jejich výstupů. Postačuje, když odevzdáte scan vytištěného formuláře opatřeného vlastnoručním podpisem.

Co naopak **neodevzdávat**:

- testovací data, která jsou součástí zadání,
- spustitelný soubor s implementovaným projektem,
- ostatní pomocné soubory produkované překladačem.

Kritéria 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. **Kvalita 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.

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

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.

8. **Prohlášení o využití generativní AI** V tomto předmětu je povoleno využívat nástroje generativní AI (ChatGPT, Copilot aj.) pro pomoc s návrhem, implementací, datovými strukturami, laděním. Podmínkou je však *naprostá transparentnost*. Každý student musí k projektu přiložit prohlášení o tom, jak byla AI využita. Celková logika předloženého řešení projektu musí být dílem studenta. Schopnost u obhajoby vysvětlit detailně vlastní kód je klíčovým požadavkem pro úspěšnou obhajobu projektu.

Prohlášení o využití generativní AI

Jméno a příjmení, login: _____

Název projektu: _____

Zvolte jednu z následujících možností:

- □ **Varianta A:** Při vypracování tohoto projektu nebyly použity žádné nástroje generativní AI.
- □ **Varianta B:** Při vypracování tohoto projektu byly využity nástroje generativní AI v rozsahu uvedeném níže.

Podrobnosti o využití AI:

Fáze	Nástroj	Způsob využití
Plánování a návrh		
Generování kódu		
Ladění a optimalizace		
Tvorba dokumentace		

Vyjádření k integritě a ověření výstupů:
Zkopírování zadání do AI a odevzdání surového výstupu je hodnoceno jako plagiátorství. Konkrétně popište:

- **Analýza a úpravy:** Jaké nedostatky, neefektivity či omyly jste ve výstupu AI odhalili a jak jste kód museli upravit?
- **Vlastní vklad:** Jakou část logiky jste navrhli sami a jak jste AI kód integrovali do svého řešení?
- **Ověření:** Jak jste nezávisle potvrdili správnost (např. trasování na vlastních datech, testování okrajových případů)?

Prohlášení studenta/studentky:

Potvrzuji, že hlavní logika a finální implementace tohoto projektu jsou mým vlastním dílem. Beru na vědomí, že nesu plnou odpovědnost za správnost veškerého obsahu, včetně částí vytvořených za pomoci AI, a že musím být schopen/schopna veškerou logiku vysvětlit během ústní obhajoby.

Podpis: _____ **Datum:** _____

1 Makroinstrukce pro procesor Kvark

Problém

Vývojový tým malého vestavěného procesoru Kvark si při ladění programů všiml zajímavé věci: v přeložených programech se neustále opakují stejné dvojice instrukcí, které jdou bezprostředně po sobě. Například po instrukci `LOAD` velmi často následuje instrukce `ADD`. Každé načtení a dekódování instrukce přitom stojí procesor čas. Tým se proto rozhodl rozšířit instrukční sadu o *makroinstrukce*. Makroinstrukce je nová instrukce, která má vlastní operační kód a provede najednou dvojici instrukcí, kterou nahrazuje. Makroinstrukce může nahrazovat i dvojici, ve které je jiná, dříve vytvořená makroinstrukce.

Program pro procesor Kvark je posloupnost operačních kódů instrukcí, tedy posloupnost kladných celých čísel. Makroinstrukce se vytvářejí tímto postupem:

1. V programu se spočítá četnost všech dvojic instrukcí, které jdou bezprostředně po sobě. Četnost se počítá ze všech sousedních pozic, výskyty se tedy mohou překrývat: v programu `7 7 7 7` má dvojice $(7, 7)$ četnost 3.

2. Vybere se dvojice s nejvyšší četností. Má-li nejvyšší četnost více dvojic, vybere se *lexikograficky nejmenší* z nich, tedy dvojice s menší první instrukcí, a při shodě první instrukce dvojice s menší druhou instrukcí.

3. Pokud je nejvyšší četnost menší než 2, postup končí.

4. Vybrané dvojici se přidělí nová makroinstrukce. Její operační kód je o 1 větší než naposledy přidělený kód. První makroinstrukce dostane kód o 1 větší než největší kód v původním programu.

5. Program se projde zleva doprava a každý výskyt vybrané dvojice se nahradí jedinou novou makroinstrukcí. Nahrazované výskyty se nepřekrývají: v programu `7 7 7 7` se dvojice $(7, 7)$ nahradí dvakrát, v programu `7 7 7` jen jednou, a to na prvních dvou pozicích.

6. Postup se opakuje od bodu 1 nad takto upraveným programem.

Kolega z vývojového týmu napsal nástroj, který tento postup provádí. Nástroj funguje správně, ale na skutečných programech o statisících instrukcí je nepoužitelně pomalý. Jeho zdrojový kód je uveden níže a slouží jako *specifikace chování*, nikoli jako vzor řešení.

Vaším úkolem je navrhnout a implementovat výrazně efektivnější verzi tohoto nástroje, která pro libovolný vstup vytvoří *přesně stejný výstup*, tedy stejné makroinstrukce ve stejném pořadí se stejnými operačními kódy a stejný výsledný program. Algoritmus i datové struktury můžete změnit podle svého uvážení. Zachovejte však formát vstupu i výstupu.

```
#include <iostream>
#include <vector>
#include <map>
#include <utility>
```

```cpp
using namespace std;

struct Macro
{
	int opcode;	// operacni kod nove makroinstrukce
	int first;	// prvni nahrazovana instrukce
	int second;	// druha nahrazovana instrukce
};

void process(vector<int>& program, vector<Macro>& macros)
{
	if (program.empty()) return;

	// Najdi nejvyssi pouzity kod instrukce v programu
	int nextOpcode = program[0];
	for (int i = 1; i < (int)program.size(); i++)
	{
		if (program[i] > nextOpcode)
			nextOpcode = program[i];
	}
	// Predpoklad vstupu: je mozne pridelovat dalsi kody bez preteceni int.
	nextOpcode++;

	while (program.size() >= 2)
	{
		// 1. Spocitani cetnosti vsech dvojic po sobe jdoucich instrukci
		vector<vector<int>> counts(nextOpcode + 1, vector<int>(nextOpcode + 1, 0));
		for (int i = 0; i + 1 < (int)program.size(); i++)
		{
			counts[program[i]][program[i + 1]]++;
		}

		// 2. Vyber nejcastejsi dvojice
		int bestCount = 0;
		int bestOpcode = 0;
		int bestOpcode2 = 0;
		for (int i = 0; i <= nextOpcode; i++)
		{
			for (int j = 0; j <= nextOpcode; j++)
			{
				if (counts[i][j] > bestCount)
				{
```

```
                                        bestCount = counts[i][j];
                                        bestOpcode = i;
                                        bestOpcode2 = j;
                                }
                        }
                }
                if (bestCount < 2)
                        break;

                // 3. Ulozeni definice nove makroinstrukce
                Macro macro;
                macro.opcode = nextOpcode;
                macro.first = bestOpcode;
                macro.second = bestOpcode2;
                macros.push_back(macro);

                // 4. Vytvoreni noveho programu
                vector<int> result;
                for (int i = 0; i < (int)program.size();)
                {
                        if (i + 1 < (int)program.size() &&
                                program[i] == bestOpcode &&
                                program[i + 1] == bestOpcode2)
                        {
                                result.push_back(nextOpcode);
                                i += 2;
                        }
                        else
                        {
                                result.push_back(program[i]);
                                i++;
                        }
                }
                program = result;  // puvodni program se zahodi
                nextOpcode++;
        }
}

// Generuje testovaci program s nahodnymi instrukcemi v rozsahu minOpcode a maxOpcode
vector<int> generateTestInstructions(int minOpcode, int maxOpcode, int programSize)
{
        vector<int> program;
```

```
        for (int i = 0; i < programSize; i++)
        {
                program.push_back(rand() % (maxOpcode - minOpcode + 1) + minOpcode);
        }
        return program;
}

int main()
{
        vector<int> program = {1,2,3,1,2,3,1,2,4,4,4,4};
        vector<Macro> macros;
        process(program, macros);

        cout << "Makroinstrukce:\n";
        for (int i = 0; i < (int)macros.size(); i++)
        {
                cout << macros[i].opcode << " -> "
                        << macros[i].first
                        << " " << macros[i].second << std::endl;
        }
        cout << "Program:\n";
        for (int i = 0; i < (int)program.size(); i++)
        {
                cout << program[i] << " ";
        }
        cout << std::endl;

        // Testovaci programy
        // Test, ktery byste meli zvladnout pro splneni projektu za minimalni pocet bodu.
        vector<int> program1 = generateTestInstructions(1, 10, 100000);
        vector<Macro> macros1;
        process(program1, macros1);

        // Testovaci programy
        // Test, ktery byste meli zvladnout pro splneni projektu za plny pocet bodu.
        vector<int> program2 = generateTestInstructions(1, 10, 1000000);
        vector<Macro> macros2;
        process(program2, macros2);
}
```

Ukázkový příklad

Uvažujme instrukce s operačními kódy `1` = `LOAD`, `2` = `ADD`, `3` = `STORE` a `4` = `NOP`. Vstupem je program

```
1 2 3 1 2 3 1 2 4 4 4 4
```

tedy `LOAD ADD STORE LOAD ADD STORE LOAD ADD NOP NOP NOP NOP`. Největší kód v programu je 4, první makroinstrukce tedy dostane kód 5. Průběh výpočtu ukazuje tabulka 1.

Krok	Program před krokem	Četnosti dvojic	Nová makroinstrukce
1	`1 2 3 1 2 3 1 2 4 4 4 4`	$(1,2):3$, $(4,4):3$, $(2,3):2$, $(3,1):2$, $(2,4):1$	$5 \to 1\ 2$
2	`5 3 5 3 5 4 4 4 4`	$(4,4):3$, $(3,5):2$, $(5,3):2$, $(5,4):1$	$6 \to 4\ 4$
3	`5 3 5 3 5 6 6`	$(3,5):2$, $(5,3):2$, $(5,6):1$, $(6,6):1$	$7 \to 3\ 5$
4	`5 7 7 6 6`	všechny dvojice mají četnost 1	konec

Tabulka 1: Průběh vytváření makroinstrukcí pro ukázkový program

V kroku 1 mají nejvyšší četnost 3 dvě dvojice, $(1,2)$ a $(4,4)$. Vybere se lexikograficky menší dvojice $(1,2)$. V kroku 2 má dvojice $(4,4)$ četnost 3, protože se její výskyty v `4 4 4 4` překrývají, nahradí se však jen dvakrát. V kroku 3 se ze stejně četných dvojic $(3,5)$ a $(5,3)$ vybere $(3,5)$. Výstup programu je tento:

```
Makroinstrukce:
5 -> 1 2
6 -> 4 4
7 -> 3 5
Program:
5 7 7 6 6
```

Pro vstup `1 2 1 2 3 1 2 1 2 3` vypadá výstup takto:

```
Makroinstrukce:
4 -> 1 2
5 -> 4 3
6 -> 4 5
Program:
6 6
```

Poznámky

- *Shodný výstup.* Pro každý vstup musí vaše řešení vypsat přesně stejné makroinstrukce a stejný výsledný program jako výchozí implementace. Výchozí implementaci použijte k ověření správnosti na vlastních vstupech.

- *Výkon.* Pro splnění zadání na minimální počet bodů by Váš program měl doběhnout na 100 tisicích instrukcích. Zpracování velkého testovacího programu s 1 milionem instrukci pak znamená splnění zadání rojektu s aspirací na plný počet bodů.
- *Měření.* Změřte dobu běhu svého řešení pro vstupy o velikostech 100 tisíc a 1 milion instrukcí vytvořené stejným způsobem pomocí Vaší implementace a uveďte naměřené časy.
- *Analýza složitosti.* Popište, kde jsou kritická místa výchozí implementace a proč. Popište a vysvětlete změny, které vedly ke zrychlení. Uveďte naměřené časy.
- Lze předpokládat, že při přidělování nových operačních kódů nedojde k přetečení typu `int`.
- Zdrojový kód ze zadání najdete v archivu s testovacími daty.

2 Síť chodeb pod městem Er

Problém

Měšťané starobylého města Er se rozhodli pro případ nebezpečí propojit chodbami dosud navzájem oddělené sklepy svých domů. O propojení všech sklepů dohromady se snaží z jednoho prostého důvodu: aby se obyvatelé při dobytí ulic mohli pod zemí přesunout ke dvěma tajným únikovým štolám, které vedou za hradby města. Tyto únikové štoly nazveme Severní štola a Jižní štola. Tyto štoly se nachází na protilehlých stranách města. Výstavby chodeb se účastní všichni měšťané, to znamená, že na síť chodeb jsou napojeny všechny sklepy. Z každého sklepa (z každého domu) se musí dát dostat k oběma únikovým štolám. Pro výstavbu chodeb platí striktní pravidlo: v síti chodeb nesmí být možné vpadnout prchajícím obyvatelům do zad a obklíčit je, což znamená, že z libovolného sklepa se k oběma únikovým štolám nelze dostat dvěma cestami.

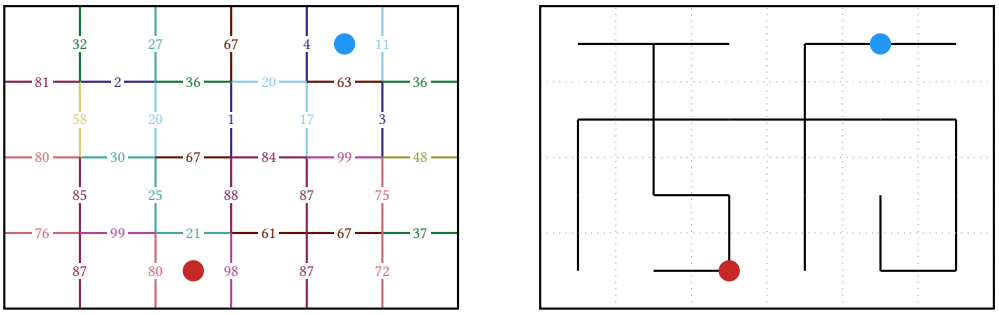
Podloží pod městem Er však není homogenní. Stěny oddělující jednotlivé sklepy jsou tvořeny různými materiály – od měkké hlíny přes pískovec až po tvrdou žulu. Proražení každé stěny tak stojí různé množství úsilí. Měšťané chtějí ušetřit síly i nástroje. Cílem je vybudovat síť chodeb tak, aby celková náročnost projektu (součet úsilí na proražení všech vybraných stěn) byla nejmenší možná.

Vaším úkolem je navrhnout a implementovat algoritmus, který pro město Er vygeneruje stavební plán sítě chodeb mezi sklepy tak, aby:

1. v síti chodeb nebylo možné prchající obyvatele obklíčit,
2. z každého sklepa bylo možné se dostat k oběma únikovým štolám a
3. celková náročnost stavby byla nejmenší možná.

Pro jednoduchost má město Er tvar obdélníku o rozměrech $n \times m$ čtvercových buněk. Každý čtverec, buňka čtvercové sítě, tvoří jeden dům, pod každým domem je jeden sklep[3]. Na začátku je každý sklep, buňka čtvercové sítě, ohraničen čtyřmi stěnami. Při budování sítě chodeb můžeme z libovolného sklepa prokopat chodbu do čtyř sousedních sklepů, do sklepa vlevo, vpravo, pod a nad v dané čtvercové síti. Chodbu lze prokopat jen pokud sousední sklep existuje. Náročnost proražení stěny je dána přirozeným číslem v intervalu $\langle 1; 100 \rangle$. Platí, že čím vyšší číslo, tím vyšší náročnost.

[3] Pro názornost si představme čtverečkovaný papír. Jednotlivé čtverečky představují sklepy pod domy města.



(a) plánek sklepů (b) plán výstavby

Obrázek 1: Síť chodeb pod městem Er

Ukázkový příklad

Plánek ukázkového města o rozměrech 4×6 čtvercových buněk můžeme vidět na obrázku 1a. Náročnost proražení stěn je uvedena přímo na stěnách a je, pro lepší přehlednost, vyznačena i barvou. Severní štola je označena modrým puntíkem, Jižní štola je označena červeným puntíkem. Výsledný plán chodem je vidět na obrázku 1b.

Poznámky

- Plán města Er je uložen v textovém souboru. Na prvním řádku je uloženo číslo n, na dalším řádku číslo m. Dále následuje sloupec Severní štoly a na dalším řádku je to sloupec Jižní štoly. Severní štola se nachází vždy na řádku 0, Jižní štola se vždy nachází na řádku $n - 1$. Na dalších řádcích jsou uložena data o stěnách mezi jednotlivými sklepy ve formátu (a, b) (c, d) e, kde (a, b) je řádek a sloupec jednoho sklepa, (c, d) je řádek a sloupec druhého sklepa na mapě města. Číslo e pak udává náročnost proražení stěny mezi těmito dvěma sklepy. Plán města z ukázkového příkladu, viz obrázek 1a je uložen v textovém souboru `SampleCityPlan.txt` takto:

```
4
6
4
2
(0,2)       (0,3)       67
(1,4)       (1,5)       3
(3,2)       (3,3)       98
(1,3)       (2,3)       84
(2,0)       (2,1)       85
(0,4)       (1,4)       63
(3,4)       (3,5)       72
(3,1)       (3,2)       80
(1,2)       (1,3)       1
(2,3)       (2,4)       87
(2,5)       (3,5)       37
```

```
(0,0)       (0,1)       32
(1,1)       (1,2)       20
(1,5)       (2,5)       48
(3,3)       (3,4)       87
(0,0)       (1,0)       81
(0,1)       (1,1)       2
(1,3)       (1,4)       17
(0,4)       (0,5)       11
(1,0)       (1,1)       58
(2,2)       (2,3)       88
(2,0)       (3,0)       76
(3,0)       (3,1)       87
(2,3)       (3,3)       61
(1,1)       (2,1)       30
(0,3)       (0,4)       4
(2,1)       (2,2)       25
(2,4)       (2,5)       75
(0,1)       (0,2)       27
(2,2)       (3,2)       21
(1,2)       (2,2)       67
(1,0)       (2,0)       80
(0,2)       (1,2)       36
(2,1)       (3,1)       99
(0,5)       (1,5)       36
(2,4)       (3,4)       67
(1,4)       (2,4)       99
(0,3)       (1,3)       20
```

- Vygenerovaný plán sítě chodeb:

  1. uložte do textového souboru ve stejném tvaru jako má vstup, s tím rozdílem, že na konec vypíšete počet proražených stěn a celkovou náročnost stavby, viz soubor `SampleCitySolution.txt` v testovacích datech.
  2. Plán uložte i do souboru ve formátu SVG. Krátký kurz o formátu SVG naleznete na URL https://www.w3schools.com/graphics/svg_intro.asp

3 Minimalizace počtu popelnic

Problém

Technické služby města Frikulínova mají za úkol rozmístit po městě popelnice pro svoz odpadu. Technické služby na mapě města vytipovaly možná stanoviště, kde by mohly být popelnice umístěny. Nyní je potřeba vyřešit problém, na která stanoviště popelnice skutečně umístit. Výběr se řídí dvěmi podmínkami:

1. po městě má být rozmístěn co nejmenší možný počet popelnic a současně
2. docházková vzdálenost od každého vchodu do domu k nejbližší popelnici musí být menší nebo rovna D, kde D je daná konstanta.

Technické služby města Frikulínov se rozhodly rozmístění popelnic řešit pomocí algoritmu. Vaším úkolem je tedy navrhnout a implementovat algoritmus, který tento problém vyřeší.

Vstupem do požadovaného algoritmu je mapa města. Pro jednoduchost předpokládejme, že město je tvořeno obecně obdélníkovou mřížkou znaků. V mřížce jsou použity tyto znaky:

`#`	neprůchozí pole (budova, zeď nebo jiná překážka)
`.`	průchozí pole (volný terén, ulice)
`E`	vchod do domu
`P`	možné stanoviště popelnice

Vchod `E` je zapuštěn přímo ve zdi domu, tedy nahrazuje jedno pole `#`. Z hlediska pohybu je ale `E` průchozí. Stanoviště `P` je rovněž průchozí.

Pohyb po mapě je povolen pouze ve čtyřech směrech: nahoru, dolů, vlevo a vpravo. Diagonální pohyb není povolen. Přesun z jednoho políčka mapy na sousední představuje jeden krok. Docházkovou vzdálenost od vchodu `E` ke stanovišti popelnice `P` bereme jako nejmenší možný počet kroků mezi oběma místy.

Obyvatelé jsou srozuměni s tím, že k popelnici musí dojít pěšky. Zda to bude 5 nebo 15 kroků, nehraje roli, dokud je dodržen maximální povolený limit. Technické služby chtějí nakoupit co nejméně popelnic, nikoliv občanům zkracovat ranní procházku. Součet vzdáleností k popelnicím není nutné nijak optimalizovat.

Ukázkový příklad

Na obrázku 2 můžeme vidět jednoduchou mapu města. Pro lepší vizualizaci jsou znaky znázorněny pomocí barev: `#` černě, `.` zeleně, `E` červeně a konečně `P` modře. Ukázkové město obsahuje domy s celkem pěti vchody, na obrázku jsou vyznačeny červeně. Technické služby dále vytipovaly devět možných stanovišť popelnic, které jsou na obrázku vyznačené modře. Polohy vchodů a stanovišť popelnic jsou uvedené v tabulkách 2a a 2b. Souřadnice $(0, 0)$ označuje na obrázku 2 levý horní roh. Docházkové vzdálenosti od každého vchodu ke všem stanovištím popelnic jsou uvedeny v tabulce 3. Pokud z tabulky vyřadíme vzdálenosti větší než $D = 20$ dostáváme tabulku 4.

Vchody $(12, 22)$, $(17, 10)$ a $(24, 25)$ můžeme pokrýt popelnicí na stanovišti $(13, 19)$, pro vchod $(25, 13)$ je jediné přípustné stanoviště $(21, 13)$. a zbylý vchod $(4, 8)$ můžeme pokrýt některým ze stanovišť $(1, 7)$, $(3, 3)$, $(5, 17)$, $(13, 10)$ a $(16, 4)$. Celkově tedy bude nutné použít tři popelnice.

Stejně tak bychom mohli pokrýt tři vchody ze stanoviště $(5, 17)$ nebo $(13, 10)$ a dostaneme další možná řešení. A opět budeme potřebovat tři popelnice.

Poznámky

- Vstupní mapa města je uložena v textovém souboru. Mapa města z ukázkového příkladu, viz obrázek 2 je uložena v textovém souboru `SampleCityMap.txt` způsobem, který je patrný z obrázku 3.

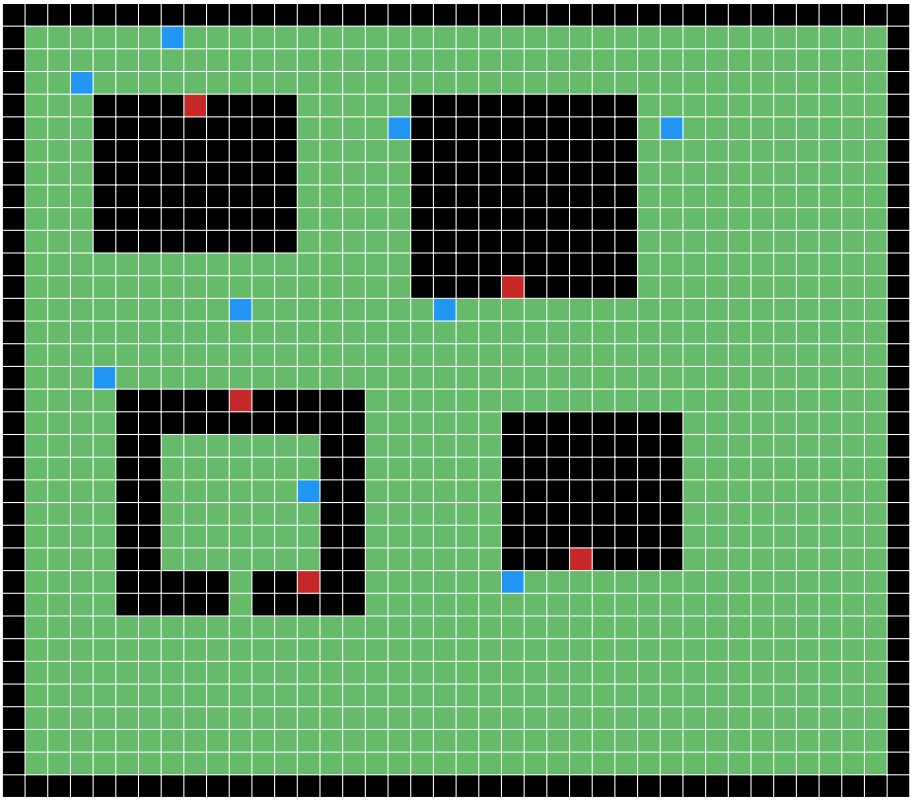
(4, 8)
(12, 22)
(17, 10)
(24, 25)
(25, 13)

(a) Vchody do domů

(1, 7)
(3, 3)
(5, 17)
(5, 29)
(13, 10)
(13, 19)
(16, 4)
(21, 13)
(25, 22)

(b) Stanoviště popelnic

Tabulka 2: Souřadnice vchodů do domů a stanovišť popelnic



Obrázek 2: Ukázková mapa města

	Popelnice								
Vchod	(1, 7)	(3, 3)	(5, 17)	(5, 29)	(13, 10)	(13, 19)	(16, 4)	(21, 13)	(25, 22)
(4, 8)	4	6	12	24	19	22	20	46	37
(12, 22)	28	30	14	16	13	4	22	36	15
(17, 10)	25	21	19	31	4	13	7	33	22
(24, 25)	43	45	29	27	28	19	31	27	4
(25, 13)	44	38	36	48	33	30	24	4	21

Tabulka 3: Tabulka vzdáleností mezi vchody a popelnicemi

	Popelnice								
Vchod	(1, 7)	(3, 3)	(5, 17)	(5, 29)	(13, 10)	(13, 19)	(16, 4)	(21, 13)	(25, 22)
(4, 8)	4	6	12	∞	19	∞	20	∞	∞
(12, 22)	∞	∞	14	16	13	4	∞	∞	15
(17, 10)	∞	∞	19	∞	4	13	7	∞	∞
(24, 25)	∞	∞	∞	∞	∞	19	∞	∞	4
(25, 13)	∞	∞	∞	∞	∞	∞	∞	4	∞

Tabulka 4: Tabulka vzdáleností mezi vchody a popelnicemi. Hodnoty větší než D jsou považovány za nekonečné.

```
########################################
#......P.............................#
#....................................#
#..P.................................#
#...####E####.....##########.........#
#...#########....P##########.P......#
#...#########.....##########........#
#...#########.....##########........#
#...#########.....##########........#
#...#########.....##########........#
#...#########.....##########........#
#................##########........#
#................####E#####........#
#.........P........P.................#
#....................................#
#....................................#
#...P................................#
#....#####E#####.....................#
#....###########......########......#
#....##.......##......########......#
#....##.......##......########......#
#....##......P##......########......#
#....##.......##......########......#
#....##.......##......########......#
#....##.......##......###E####......#
#....#####.##E##......P..............#
#....#####.#####.....................#
#....................................#
#....................................#
#....................................#
#....................................#
#....................................#
#....................................#
#....................................#
########################################
```

Obrázek 3: Ukázková mapa města – vstupní textový soubor `SampleCityMap.txt`

- Ještě před hledáním optimálního rozmístění ověřte, zda má každý vchod alespoň jedno kandidátní stanoviště ve vzdálenosti nejvýše D. Pokud existuje alespoň jeden vchod, pro který žádné takové P neexistuje, program musí oznámit, úloha nemá řešení. V takovém případě není potřeba pokračovat v optimalizaci.
- Ve všech testovacích příkladech uvažujte stejnou docházkovou vzdálenost $D = 20$.
- Není nutné hledat všechna řešení, stačí nyjít jedno řešení.
- Výstupem z aplikace bude textový soubor obsahující následující informace:
    1. rozměry mapy,
    2. počet vchodů,
    3. počet stanovišť,
    4. informaci, zda má úloha řešení,
    5. počet vybraných stanovišť a
    6. souřadnice vybraných stanovišť.
    7. Dále vypište výslednou mapu. Vybraná stanoviště pro popelnice označte znakem 0. Ostatní znaky ponechejte beze změny.

4 Výšková mapa

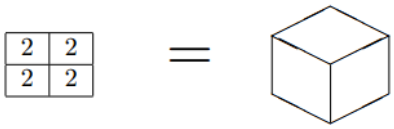
Problém

V tomto zadání budeme pracovat s nepravidelnými mnohostěny. V našem případě má mnohostěn vždy obdélníkovou podstavu[4], o rozměrech $m \times n$ délkových jednotek. Mnohostěn je zadán takzvanou výškovou mapou. Výšková mapa je matice $\mathbf{H}$ přirozených čísel typu $m \times n$, kde každý prvek matice udává výšku jisté části mnohostěnu. Prvek matice $h_{i,j}$ představuje sloupec ve tvaru kvádru se základnou o rozměrech 1×1 a výšce $h_{i,j}$. Velice zjednodušeně řečeno, jde o sloupec z $h_{i,j}$ kostek o hraně délky 1 postavených na sebe. Celý mnohostěn je složen z $m \times n$ takových sloupců položených těsně vedle sebe. Vznikne tak mnohostěn s podstavou tvořenou obdélníkem o rozměrech $m \times n$ a několika stěnami nahoře a po stranách. Vaším úkolem je implementovat algoritmus, který pro mnohostěn zadaný výškovou mapou, určí počet jeho stěn.

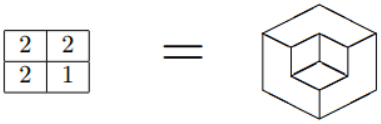
Ukázkový příklad

Například matice 2×2 se všemi hodnotami rovnými 2 představuje krychli o straně 2, viz obrázek 4a. Pokud je však jedna z hodnot rovna 1, je reprezentovaným mnohostěnem tatáž krychle s jedním vyříznutým rohem, viz obrázek 4b. Komplexnější mnohostěny můžeme vidět na obrázcích 4c a 4d. V obou případech dva horní čtverce, jevící se z pohledu čtenáře jako kosočtverce, tvoří dvě stěny.

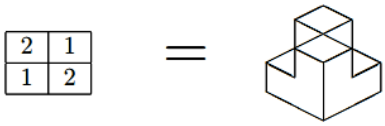
[4] Čtverec považujme za speciální druh obdélníka.



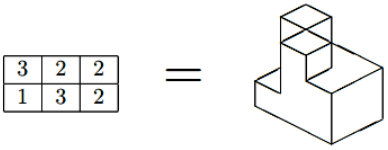
(a) krychle, 6 stěn



(b) krychle s vyříznutým rohem, 9 stěn



(c) komplexnější mnohostěn, 13 stěn



(d) komplexnější mnohostěn, 14 stěn

Obrázek 4: Příklady výškových map a jim odpovídajících mnohostěnů

Poznámky

- Každá výšková mapa bude uložena v samostatném textovém souboru. Na prvním řádku textového souboru jsou uložena dva celá kladná čísla m a n, definující rozměry výškové mapy. Dále následuje celkem m řádků, na každém řádku je uloženo n celých kladných čísel představující jednotlivé prvky výškové mapy. Čísla na řádcích jsou vždy oddělena jednou mezerou. Ukázkový mnohostěn z obrázku 4d bude uložen takto:

```
2 3
3 2 2
1 3 2
```

- Výsledek, tj. počet stěn mnohostěnu, vypište na standardní výstup.
- Stěnu chápeme tak, že je to jeden rovinný mnohoúhelník. Mezi stěny se počítá i podstava.
- Prvky matice **H** jsou přirozená čísla. Na každé pozici výškové mapy tak vždy leží aspoň jedna kostka.

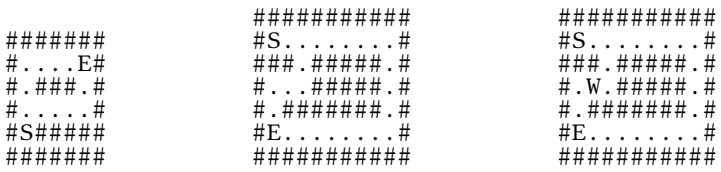
5 Sobí bludiště

Problém

Sob se účastní závodu v bludišti. Bludiště je vytvořeno ve čtvercové síti, každé políčko je buď zeď, nebo volné políčko. Jedno volné políčko je označeno jako start, jedno jako cíl. Některá volná políčka jsou zaplavena vodou; sob jimi může procházet, ale je to namáhavější.

Sob stojí na startovním políčku a je otočen *na východ* (doprava). V každém okamžiku může provést jednu ze dvou akcí:

- udělat krok vpřed na sousední políčko ve směru, kterým je otočen, pokud toto políčko není zeď; krok na obyčejné volné políčko stojí T bodů, krok na políčko s vodou stojí $10T$ bodů,
- otočit se na místě o 90° doleva nebo doprava; tato akce stojí 1000 bodů bez ohledu na to, kde sob stojí.



Obrázek 5: Ukázková sobí bludiště

Cena kroku T je kladné celé číslo zadané na vstupu. Skóre závodu je součet bodů za všechny provedené akce.

Vaším úkolem je implementovat algoritmus, který pro zadané bludiště a cenu kroku T určí *nejmenší možné skóre*, s nímž se sob dostane ze startu do cíle. Na tom, kterým směrem je sob otočen po dosažení cíle, nezáleží. Pokud se sob do cíle dostat nemůže, je výsledkem -1.

Ukázkový příklad

Bludiště zapisujeme tak, že znak `#` je zeď, `.` volné políčko, `W` volné políčko s vodou, `S` start a `E` cíl.

Příklad 1. Uvažujme cenu kroku $T = 1$ a bludiště na obrázku 5a. Sob stojí na `S` otočen na východ, kde je ale zeď. Do cíle vedou dvě trasy, obě se sedmi kroky:

- otočit se doleva (na sever), jít 3 políčka na sever, otočit se doprava (na východ) a jít 4 políčka na východ; celkem 2 otočení a 7 kroků, tj. skóre $2 \cdot 1000 + 7 \cdot 1 = 2007$,
- otočit se doleva, jít 1 políčko na sever, otočit se doprava, jít 4 políčka na východ, otočit se doleva a jít 2 políčka na sever; celkem 3 otočení a 7 kroků, tj. skóre 3007.

Nejmenší možné skóre je tedy 2007.

Příklad 2. Nejlevnější trasa závisí na ceně kroku. V bludišti na obrázku 5b vedou do cíle právě dvě trasy:

- *objížďka*: 8 kroků na východ, otočení, 4 kroky na jih, otočení, 8 kroků na západ; celkem 2 otočení a 20 kroků, skóre $2000 + 20T$,
- *klikatá cesta*: 2 kroky na východ, otočení, 2 kroky na jih, otočení, 2 kroky na západ, otočení, 2 kroky na jih; celkem 3 otočení a 8 kroků, skóre $3000 + 8T$.

Pro $T = 1$ je nejmenší možné skóre 2020 (objížďka), pro $T = 100$ je to 3800 (klikatá cesta).

Příklad 3. Zaplavíme-li v bludišti z příkladu 2 jedno políčko na klikaté cestě vodou, viz obrázek 5c, stojí klikatá cesta $3000 + 7T + 10T = 3000 + 17T$, zatímco objížďka zůstává za $2000 + 20T$. Pro $T = 1$ je nejmenší možné skóre opět 2020, pro $T = 100$ je nyní 4000, protože objížďka je levnější než klikatá cesta přes vodu (4700).

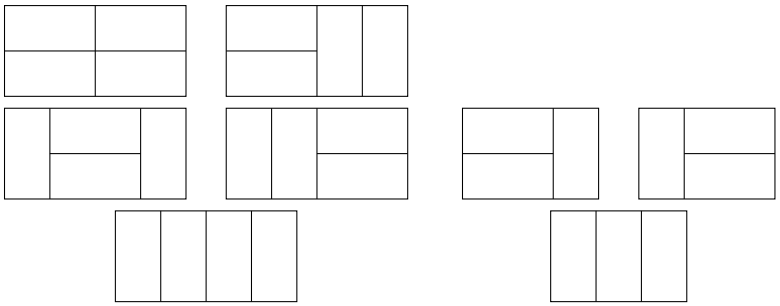
Poznámky

- Vstupem programu je textový soubor v následujícím formátu:
  - na prvním řádku je uvedena cena kroku T, kladné celé číslo,
  - na druhém řádku je uveden počet řádků čtvercové sítě bludiště r,
  - na třetím řádku je uveden počet sloupců čtvercové sítě bludiště c a
  - na dalších r řádcích je uveden řetězec tvořený c znaky, kde
    - `#` odpovídá zdi,
    - `.` odpovídá volnému políčku,
    - `W` odpovídá volnému políčku s vodou,
    - `S` odpovídá startu a
    - `E` odpovídá cíli.

  Bludiště z příkladu 3 s cenou kroku $T = 100$ bude uloženo takto:

```
100
7
11
###########
#S........#
###.#####.#
#.W.#####.#
#.#######.#
#E........#
###########
```

- Cesta k vstupnímu souboru je programu předána jako první argument příkazové řádky.
- Celý okraj bludiště tvoří vždy zeď. Start i cíl jsou v bludišti právě jednou a nejsou zaplaveny vodou.
- Cena kroku se vždy řídí políčkem, *na které* sob vstupuje. Otočení stojí 1000 bodů i na políčku s vodou.
- Bludiště může obsahovat volná políčka, na která se sob ze startu nedostane. Cesta ze startu do cíle nemusí existovat.
- Na standardní výstup vypište jediné celé číslo: nejmenší možné skóre, nebo -1, pokud cesta ze startu do cíle neexistuje.
- Nepovinně můžete vypsat i jednu trasu s nejmenším skóre, například bludiště, v němž jsou políčka trasy nahrazena šipkami `^`, `>`, `v`, `<` udávajícími směr soba. Tras s nejmenším skóre může být více; výpis trasy se nekontroluje proti vzorovému řešení, ale musí odpovídat vypsanému skóre.



(a) 2×4 (b) 2×3

Obrázek 6: Možné způsoby položení parket pro různé rozměry pokojů

6 Parkety

Problém

V tomto zadání se vžijete do role návrháře interiéru, který má vytvořit parketovou podlahu v obdélníkovém pokoji o rozměrech $a \times b$ délkových jednotek. Protože návrhář pracuje pro náročného a vrtošivého zákazníka, musí vytvořit mnoho různých návrhů, jak parkety poskládat do nejrůznějších vzorů. Zákazník by nejraději viděl všechny možné způsoby pokládky parket ve svém pokoji. Návrhář však oponuje, že tento počet může být velmi vysoký. Vaším úkolem je implementovat algoritmus, který pro pokoj o rozměrech $a \times b$ vypočítá počet všech možných způsobů pokládky parket. Předpokládejte, že používané parkety mají jednotný rozměr 1×2 jednotky. Podlaha pokoje musí být parketami pokryta beze zbytku a parkety nelze řezat.

Ukázkový příklad

Všechny možné způsoby pokládky parket pro pokoje o rozměrech 2×4 a 2×3 jsou uvedeny na obrázku 6.

Poznámky

- Vstupem do vašeho programu bude textový soubor obsahující rozměry různých pokojů, na kterých budete testovat implementovaný algoritmus. Na každém řádku textového souboru budou tři celá čísla a, b, c oddělena mezerou. Čísla a a b představují rozměry pokoje, číslo c udává počet všech možných způsobů pokládky parket pro daný pokoj. Toto číslo je zde pro kontrolu správnosti Vámi implementovaného algoritmu. Ukázka vstupního souboru:

```
1 2 1
1 3 0
1 4 1
2 2 2
2 3 3
2 4 5
2 11 144
4 11 51205
```

Všimněte si, že úloha nemusí mít řešení, např. pro pokoj o rozměrech 1×3.

- Na standardní výstup vypište vždy rozměry pokoje a počet všech možných způsobů pokládky parket.

Rozdělení zadání mezi studenty

Prezenční forma studia

	Login	Student	Zadaný projekt
1	ABI0008	Abilmazhinova Aruzhan	1
2	AMR0013	Amruz Josef	2
3	AXM0015	Axman Filip	3
4	BAK0055	Bakoš Andrej	4
5	BAL0344	Balcar David	5
6	BAL0346	Balajka Timotej	6
7	BAN0075	Baňas Marek	1
8	BAR0866	Bartoš Ámos	2
9	BAR0888	Baron Jeroným	3
10	BAR0892	Bartulec Lukáš	4
11	BEL0178	Belyayeva Yelizaveta	5
12	BEL0200	Belko Adam	6
13	BEN0362	Beniak Adam	1
14	BER0348	Bernát Michal	2
15	BIA0040	Bialas Patrik	3
16	BLA0399	Blažej Matěj	4
17	BLA0401	Bláha Vojtěch	5
18	BOC0093	Boček David	6
19	BRA0255	Brachtl Jakub	1
20	BUD0081	Budoš Matúš	2
21	BUG0021	Bugr Jaroslav	3
22	BUI0040	Bui Vu Phuong Linh	4
23	BUK0064	Bukovan Tomáš	5

pokračování na další stránce...

	Login	Student	Zadaný projekt
24	BZD0007	Bzdilíková Michaela	6
25	CAJ0038	Čajánek Mikuláš	1
26	CEC0184	Čech Lukáš	2
27	CIA0014	Ciahotná Lenka	3
28	CIE0138	Cieslar Jiří	4
29	CMI0036	Čmiel Damián	5
30	CUP0032	Čupa Lukáš	6
31	CZE0047	Czech Jakub	1
32	DAN0229	Daníček Tadeáš	2
33	DAU0020	Dautov Bekezhan	3
34	DEJ0015	Dejlová Kristýna	4
35	DIH0016	Dihel Lukáš	5
36	DLU0048	Dluhoš Richard	6
37	DOB0183	Dobešová Tereza	1
38	DOS0246	Dostál Filip	2
39	DRO0116	Drobyna Oleksandr	3
40	DRO0123	Drobík Matyáš	4
41	DRZ0012	Držka Samuel	5
42	DVO0283	Dvořák Šimon	6
43	FAB0083	Fabisz Samuel	1
44	FAR0103	Farník Michal	2
45	FAR0104	Farkašovský Lukáš	3
46	FIE0038	Fiedor Matyáš	4
47	FIL0207	Filip Eduard	5
48	FOL0195	Folvarčný Martin	6

pokračování na další stránce...

	Login	Student	Zadaný projekt
49	FOL0196	Foldyna Tomáš	1
50	GAB0104	Gabriel Radim	2
51	GEN0024	Genči Jaroslav	3
52	GLO0070	Glončák Maroš	4
53	GRE0168	Gregořica Adam	5
54	GRY0127	Grygar Jan	6
55	GRY0128	Grygařík Ondřej	1
56	GUB0007	Gubayev Aslan	2
57	GUR0065	Gürtler Matěj	3
58	HAL0268	Halfar Matej	4
59	HAV0358	Havránek Václav	5
60	HAV0359	Havlásek Matyáš	6
61	HLO0068	Hloch Michal	1
62	HNI0031	Hnietnov Danylo	2
63	HOL0621	Holčák Jan	3
64	HOL0622	Holba Kristián	4
65	HOL0624	Holánik Martin	5
66	HOR0640	Horňák Maxmilián	6
67	HOR0642	Hořínka Jakub	1
68	HOR0643	Horáček Ondřej	2
69	HRO0111	Hrozová Julie	3
70	HUJ0022	Hujdus Jan	4
71	CHA0289	Chabreček Stanislav	5
72	CHL0124	Chlebanová Kristína	6
73	CHM0088	Chmel Daniel	1

pokračování na další stránce...

	Login	Student	Zadaný projekt
74	IHN0018	Ihnát Jan	2
75	IWO0009	Iwoniak Lukáš	3
76	JAL0045	Jalůvka Štěpán	4
77	JAN1030	Janeczko Jiří	5
78	JAN1032	Janča Stanislav	6
79	JAN1033	Janarbyek Tilyekbyerdi	1
80	JAN1034	Jantula Marko	2
81	JEZ0086	Ježková Nikol	3
82	JEZ0113	Ježek Jonáš	4
83	JUK0013	Jukl Vojtěch	5
84	JUR0567	Juriš Zachariáš	6
85	KAD0236	Kaduch Jan	1
86	KAL0382	Kala Luboš	2
87	KAL0383	Kalivoda Jiří	3
88	KAM0117	Kamshybek Damir	4
89	KAR0336	Karolczyk Tomasz	5
90	KAS0301	Kassymbekova Diana	6
91	KAT0031	Katona Petr	1
92	KLA0136	Klapetek Jakub	2
93	KNE0063	Knebl Tobiáš	3
94	KNI0030	Knichal Jiří	4
95	KOB0082	Kobiela Filip	5
96	KOC0413	Kocůrek Adam	6
97	KOL0661	Kolek Ondřej	1
98	KOM0143	Komendát Erik	2

pokračování na další stránce...

	Login	Student	Zadaný projekt
99	KON0530	Kondratov Vladyslav	3
100	KON0542	Kononenko Artem	4
101	KON0544	Konvičný Michal	5
102	KOR0350	Korsa Jan	6
103	KOS0445	Košnar Pavel	1
104	KOT0339	Kotula Matúš	2
105	KOV0449	Kovařčík Vladimír	3
106	KOV0450	Kovalenko Artem	4
107	KOZ0380	Kožiarik Marko	5
108	KOZ0393	Kozačka Ondřej	6
109	KOZ0394	Kozar Martin	1
110	KRA0775	Krayzel Jan	2
111	KRC0099	Krchňák Boris	3
112	KRE0463	Kreutz Adam	4
113	KRE0487	Kresa Ondřej	5
114	KRE0528	Krejčí František	6
115	KRU0239	Krüger Jan	1
116	KUB0813	Kubát Tomáš	2
117	KUB0838	Kubányi Robert	3
118	KUC0451	Kuczera Jan	4
119	KUC0452	Kuczera Tomáš	5
120	KUC0453	Kučera Filip	6
121	KUC0456	Kuchler Matěj	1
122	KUP0152	Kupka Michal	2
123	LAD0018	Ladňák Adam	3

pokračování na další stránce…

	Login	Student	Zadaný projekt
124	LAL0020	Ľaľová Klaudia	4
125	LAT0059	Látka Tomáš	5
126	LIB0058	Libosvár Jiří	6
127	LIS0168	Lisztwan Vojtěch	1
128	LUK0202	Lukaštík Filip	2
129	LUZ0037	Lužný Jakub	3
130	MAC0722	Macura Matěj	4
131	MAR1021	Marek Šimon	5
132	MAR1080	Maryniok Adam	6
133	MAR1081	Martynková Veronika	1
134	MAT0607	Matyáš Jakub	2
135	MAY0031	Mayer Tomáš	3
136	MEI0035	Meinhard Michael	4
137	MEL0097	Melnyk Nikita	5
138	MIK0596	Miko Jozef	6
139	MIK0598	Mikulík Jakub	1
140	MLC0053	Mlčák Jan	2
141	MOR0296	Morcinek Sebastian	3
142	MRA0130	Mrázek Jakub	4
143	MUK0014	Mukhambetkali Ayan	5
144	MYL0012	Myloserdov Kyrylo	6
145	NEK0032	Nekoranec Andrej	1
146	NEM0303	Nemčok Patrik	2
147	NES0071	Nespěšná Viktorie	3
148	NEU0131	Neuwirth Ondřej	4

pokračování na další stránce…

	Login	Student	Zadaný projekt
149	NGU0406	Nguyen Van An	5
150	NIK0073	Nikolov Dominik	6
151	NOS0046	Noska Matyáš	1
152	NOV0818	Novotný Matěj	2
153	NOV0820	Novák Adam	3
154	NOV0821	Novák Tomáš	4
155	OLE0063	Oleksyk Vasyl	5
156	OND0325	Ondra Tomáš	6
157	OTA0039	Otáhal Vojtěch	1
158	OZA0022	Ožana Bernard	2
159	PAV0634	Pavlíček Felix	3
160	PAV0636	Pavelek Filip	4
161	PER0228	Perďoch Vladislav	5
162	PER0230	Perďoch Jakub	6
163	PET0519	Petřík Vojtěch	1
164	PET0520	Petruška Ondřej	2
165	PIC0039	Pícha Patrik	3
166	PIN0110	Pinček Pavel	4
167	PLE0121	Pleva David	5
168	POD0207	Podkul Adam	6
169	POL0622	Polzer Jakub	1
170	POT0077	Potůček Daniel	2
171	PRA0186	Pravda Daniel	3
172	PRO0419	Procházka Tomáš	4
173	PRY0031	Prydatko Vladyslav	5

pokračování na další stránce...

	Login	Student	Zadaný projekt
174	PUP0023	Pupík Aki	6
175	REH0153	Řehák Michael	1
176	REZ0150	Rezner Jan	2
177	RUC0078	Ručková Kristýna	3
178	RYB0136	Rybka Jakub	4
179	SAV0037	Savchenko Oleksandr	5
180	SED0334	Sedlíková Viktorie	6
181	SED0335	Sedlářová Veronika	1
182	SEI0097	Seifert Lukáš	2
183	SEI0103	Seitkaziyev Nurdaulet	3
184	SEN0108	Šenkeřík Vojtěch	4
185	SER0086	Serafín Petr	5
186	SEV0193	Ševčuk Filip	6
187	SHI0035	Shipilova Yekaterina	1
188	SHY0003	Shylov Kyrylo	2
189	SCH0470	Schaffartzik Patrik	3
190	SCH0471	Schuchert Jeremy	4
191	SCH0472	Schwan Lukáš	5
192	SCH0492	Schee Radim	6
193	SCH0493	Schmeister Karel	1
194	SIK0250	Sikora Šimon	2
195	SIM0441	Šimek Filip	3
196	SKR0171	Škrhla David	4
197	SLA0395	Slavík Marek	5
198	SLA0398	Šlachta Lukáš	6

pokračování na další stránce…

	Login	Student	Zadaný projekt
199	SLA0399	Slazyk Jakub	1
200	SLA0400	Slanina Radoslav	2
201	SLI0148	Slíva Martin	3
202	SME0169	Smékal Jiří	4
203	SOB0143	Sobolová Klára	5
204	SOB0144	Sobol Petr	6
205	SOC0089	Sochacki Daniel	1
206	SPA0211	Španiel Michal	2
207	SRU0045	Šrubař Václav	3
208	STE0637	Štěpánek Martin	4
209	STO0257	Stonawski Tobijas	5
210	STO0268	Stopka Adam	6
211	STR0678	Stránský Jan	1
212	STU0221	Stuchlík Michal	2
213	STY0028	Štýbar Vojtěch	3
214	SUC0151	Suchánek Jan	4
215	SUG0022	Šugárek Lukáš	5
216	SVA0226	Švancár Lucián	6
217	SVE0176	Švecová Rozálie	1
218	SVE0177	Švehelka Patrik	2
219	TAH0016	Taha Adam	3
220	TOM0473	Tomis Tadeáš	4
221	TOM0535	Tomussyak Bogdan	5
222	TVA0030	Tvardek Pavel	6
223	TVR0054	Tvrdý Vojtěch	1

pokračování na další stránce...

	Login	Student	Zadaný projekt
224	ULE0014	Ulenfeld Jakub	2
225	VAC0312	Vaculík Lukáš	3
226	VAL0552	Valger Kirill	4
227	VAL0554	Valenta Tomáš	5
228	VAL0555	Vala Michael	6
229	VAL0556	Válek Kristián	1
230	VAN0393	Vaněk Antonín	2
231	VAN0394	Vandra Matyáš	3
232	VAN0396	Vanca Michal	4
233	VAN0397	Vaňková Kateřina	5
234	VAN0399	Vančura František	6
235	VAS0293	Vasilevi Davit	1
236	VAV0279	Vavrys Marek	2
237	VIL0103	Vilášek Matěj	3
238	VLC0133	Vlček Jan	4
239	VOJ0168	Vojtuš Roman	5
240	VOJ0169	Vojár Fabián	6
241	VRA0189	Vrána Jakub	1
242	VYS0087	Vysloužil Matyáš	2
243	WAL0128	Walach Matěj	3
244	WRU0009	Wrubel Ondřej	4
245	YEV0008	Yevsieiev Dmytro	5
246	ZAK0132	Zakrevskyi Vitalii	6
247	ZAR0077	Zarchshikov Davyd	1
248	ZBO0061	Zbojek Matej	2

pokračování na další stránce…

	Login	Student	Zadaný projekt
249	ZDR0059	Zdražila Jakub	3
250	ZEN0038	Zenchenko Artemii	4
251	ZID0108	Židek Matěj	5
252	ZIZ0058	Žižková Lenka	6
253	ZLA0049	Zlámal Jan	1
254	ZNA0012	Žňava Jan	2
255	ZUS0013	Zušťák Lukáš	3

Kombinovaná forma studia

	Login	Student	Zadaný projekt
1	BAL0320	Baláž Thomas Ladislav	1
2	BAR0889	Barčuk Jakub	2
3	BEL0153	Bělohoubková Sandra	3
4	BRO0167	Broda Zbyněk	1
5	DRI0033	Dřízga Samuel	2
6	FOJ0151	Fojtík Petr	3
7	HAR0200	Harang Václav	1
8	JOS0041	Joštiak Matej	2
9	KAL0402	Kaluža Dominik	3
10	KAR0336	Karolczyk Tomasz	1
11	KON0572	Konečná Veronika, Bc.	2
12	KRE0410	Krepčík Martin	3
13	KUL0172	Kulíšek Dušan	1
14	LUK0201	Lukačovič Vít	2
15	MAR1120	Martikán Dominik	3

pokračování na další stránce…

	Login	Student	Zadaný projekt
16	MIC0123	Michalčík Ondřej	1
17	OBS0020	Obšivač Martin	2
18	PFE0013	Pfeffer Steven	3
19	PIA0022	Piątek Jakub Tomasz	1
20	POR0087	Porubský Tomáš, Ing.	2
21	SZO0046	Szotek David	3
22	VAL0575	Valová Denisa	1
23	ZOU0031	Zoubek Martin	2