

Úvod do programování

Algoritmus. Vlastnosti algoritmu. Programovací jazyky.

Tomáš Bayer | bayertom@fsv.cvut.cz

Katedra geomatiky, fakulta stavební ČVUT.

Obsah přednášky

- 1 Úvodní informace o předmětu
- 2 Motivace
- 3 Problém a algoritmus
- 4 Algoritmus
- 5 Programovací jazyky a jejich dělení
- 6 Zápis a překlad programu

1. Plán přednášek

Tematické celky:

- (1) Problémy a algoritmy.
- (2, 3) Datové struktury a datové typy.
- (4) Podmínky.
- (5) Cykly.
- (6) Funkce.

Literatura:

[1] VIRIUS M.: Základy algoritmizace, 2004, Vydavatelství ČVUT.

Cvičení:

Implementace algoritmů v jazyce Python.

Literatura:

[1] Summerfield M.: Python 3, Computer Press, 2012

[2] Pilgrim M.: Ponořme se do Pythonu 3, CZ NIC, 2010:

<http://diveintopython3.py.cz/index.html>

[3] On line kurz: <https://naucese.python.cz/2017/pyladies-brno-jaro-po/>

2. Proč se učit programovat?

Steve Jobs:

“Everybody in this country should learn how to program a computer... because it teaches you how to think”.

Základní motivace:

- 1 Schopnost automatizace opakujících se činností.
- 2 Pochopení, jak a proč věci fungují.
- 3 Rozvoj logického a abstraktního myšlení.
- 4 Schopnost týmové práce, kreativita
- 5 Kontakt s moderními technologiemi.
- 6 Atraktivní, tvůrčí a dobře placené zaměstnání.
- 7 Možnost práce z domova.

Nutnost celoživotního vzdělávání.

Pravidlo 10-10: Za 10 let pouze 10% poznatků aktuálních.

3. Umím programovat...

Obečnější pohled:

- 1 Analýza, modelování, simulace procesů.
- 2 Podíl na řešení složitých problémů dnešního/budoucího světa.
- 3 Věda, výzkum, vývoj, inovace.
- 4 Aplikace v průmyslu, zemědělství, službách.

Konkrétnější uplatnění:

- 1 Automatizované zpracování a analýza dat.
- 2 CAD, počítačová grafika, digitální kartografie, mapové služby.
- 3 Modelování, analýzy, predikce v oblasti přírodních věd.
- 4 Umělá inteligence.
- 5 Návrh a tvorba mobilních řešení.
- 6 Cloudová infrastruktura, big data.

Front end vs. back end.

4. Problém

Termín mající více významů.

Definice 1 (Slovník spisovného jazyka českého)

“Věc k řešení, nerozřešená sporná otázka, otázka k rozhodnutí, nesnadná věc”

Definice 2 (Wikipedia).

“Podmínky nebo situace nebo stav, který je nevyřešený, nebo nechtěný, nebo nežádoucí.”

Problém zpravidla vyžaduje řešení.

Pro jeho nalezení nutné pochopit nejdůležitější aspekty problému.

Z hlediska informatiky, problémy musí splňovat formální definici.

Ne všechny problémy lze v současné době úspěšně a efektivně řešit.

Úlohy třídy P: snadno řešitelné.

Úlohy třídy NP: klasickými technikami neřešitelné.

Mnoho z nich spadá do oblasti geoinformatiky/geoinformatiky/dpz.

5. Popis problému

“Problém” z pohledu informatiky lze formalizovat:

NÁZEV: Slovní popis problému

IN: Popis přípustného vstupu (množina vstupních dat).

OUT: Popis výsledku, který je pro daný vstup očekáván.

Musí existovat funkce f přiřazující vstupním datům požadovaný výstup.

Nalezení řešení problému \Rightarrow nalezení příslušné funkce f .

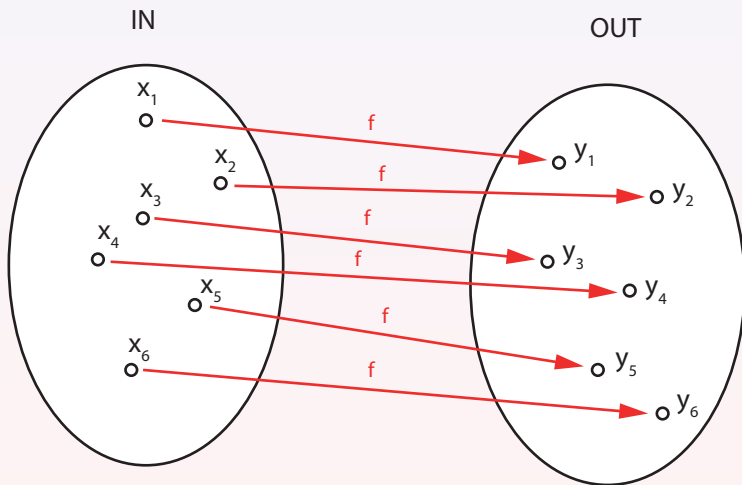
Každý problém P určen uspořádanou trojicí $P(IN, OUT, f)$

$$f : IN \rightarrow OUT,$$

IN množina přípustných vstupů, OUT množina očekávaných výstupů, f přiřazuje každému vstupu očekávaný výstup.

IN/OUT: Kombinace znaků, celých čísel či přirozených čísel představující kódování.

6. Znázornění problému

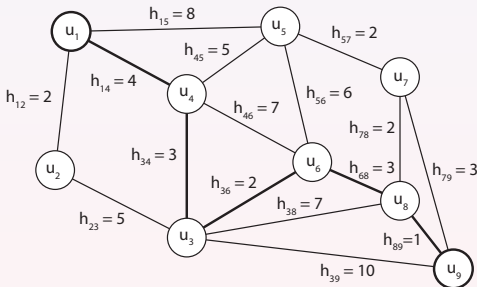


7. Příklady problémů

P1: Setřídění posloupnosti čísel:

$IN = \{16, 74, 8, 4, 11, -9, 38\},$

$OUT = \{-9, 4, 8, 11, 16, 38, 74\}$



P2: Nalezení nejkratší cesty v grafu:

$IN = \{G = (U, H, \rho)\}$

$U = \langle u_1, \dots, u_9 \rangle, H = \langle h_{12}, \dots, h_{79} \rangle$

$OUT = \langle h_{14}, h_{34}, h_{36}, h_{68}, h_{89} \rangle$

8. Algoritmus

Pojem cca 1000 let starý.

Poprvé použil perský matematik Abú Abdallah Muhammad ibn Musa al-Khwarizmi.

Algoritmus je obecný předpis pro řešení zadaného problému.

Posloupnost kroků doplněných jednoznačnými pravidly.

Analogie v běžném životě: kuchyňský recept, lékařský předpis.

Algoritmus A řeší problém P , tj. $A(P)$, pokud $\forall x \in IN$ přiřazuje v konečném počtu kroků (alespoň jeden) výstup y , $y \in OUT$, a $y = f(x)$.

Poznámky:

- Řešení vyhovuje vstupním podmínkám.
- Pro zadaný vstup x může existovat více řešení y , A by měl nalézt alespoň jedno.
- Množina vstupů zpravidla nekonečně velká.

9. Vlastnosti algoritmu

A) *Determinovanost*

Algoritmus jednoznačný jako celek i v každém svém kroku.

Nelze dosáhnout přirozenými jazyky, proto pro popis používány formální jazyky.

Algoritmus je invariantní vůči formálnímu jazyku !!!

B) *Rezultativnost*

Vede vždy ke správnému výsledku v *konečném* počtu kroků.

C) *Hromadnost*

Lze použít pro řešení stejné třídy problémů s různými vstupními hodnotami.

Pro jejich libovolnou kombinaci obdržíme *jednoznačné* řešení.

D) *Opakovatelnost*

Při opakovaném použití stejných vstupních dat vždy obdržíme *tentýž* výsledek.

E) *Efektivnost*

Každý krok algoritmu by měl být efektivní.

Krok využívá elementární operace, které lze provádět v *konečném* čase.

10. Řešení problému prostřednictvím algoritmu (1/2)

Fáze řešení problému:

1) Definice problému

Formulace problému společně s cílem, kterého chceme dosáhnout. Definujeme požadavky týkající se tvaru vstupních a výstupních dat.

2) Analýza problému

Stanovení kroků a metod vedoucích k jeho řešení.
Rozkládání problému na dílčí podproblémy: dekompozice.
Jejichž jednodušší než řešení problému jako celku.

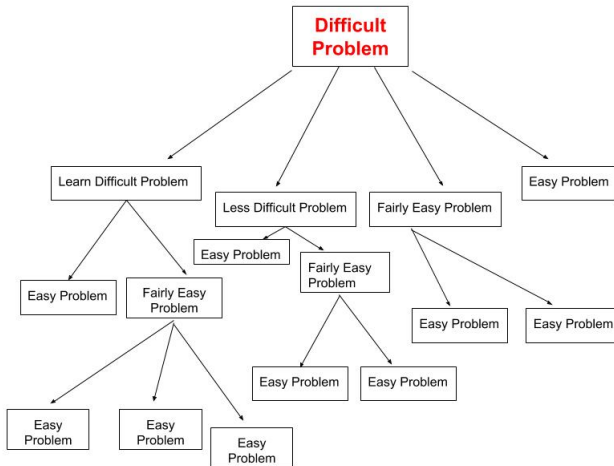
Top-Bottom Approach.

3) Sestavení algoritmu

Navržení a sestavení algoritmu řešící požadovaný problém.
Dílčí komponenty skládány do složitějších: kompozice.

Bottom-Up Approach.

11. Top-Down decomposition



12. Řešení problému prostřednictvím algoritmu (2/2)

4) Kódování algoritmu

Převod algoritmu do formálního jazyka.

Zpravidla se jedná programovací jazyk.

Výsledkem “*Proof of Concept*”.

5) Ověření správnosti algoritmu

Ověření funkčnosti algoritmu na konečném vzorku dat.

Zahrnuje běžné situace + singulární případy (mnohdy obtížné).

3 množiny: Worst, Best, Average.

Různé strategie testování: např. *Unit Testing*.

6) Nasazení algoritmu

Splňuje-li algoritmus požadavky, lze ho nasadit do „ostrého provozu“.

Poznámky:

Algoritmus by měl být dokázán pro všechny varianty vstupních dat.

Bod 5 nejnáročnější (90% času/nákladů)!

13. Paradigmata programování

Souhrn základních domněnek, předpokladů, představ.
Zahrnuje způsob formulace problému, metodologických prostředků k řešení.

Paradigma v programování ovlivňuje styl programování.
Definuje jakým způsobem vnímá programátor problém, a jak ho řeší.

Různá paradigmatata:

Procedurální (imperativní) programování	Základem algoritmus, přesně daná posloupnost kroků.
Objektově orientované programování	Založeno na objektově orientovaném přístupu k problému.
Deklarativní programování	Opak imperativního programování (říkáme co, ne jak).
Funkcionální programování	Varianta deklarativního programování, založeno na lambda kalkulu.
Logické programování	Kombinace procedurálního / deklarativního přístupu.
Paralelní programování	Pro dekomponovatelné úlohy, paralelizace zpracování.
Distribuované programování	Zpracování probíhá po částech na více počítačích.

14. Znázornění algoritmu

Algoritmus lze znázorňovat mnoha způsoby:

● Grafické vyjádření

Popsán formalizovanou soustavou grafických symbolů:

- vývojové diagramy,
- strukturogramy.

(+) přehlednost, názornost,

(+) znázornění struktury,

(+) informace o postupu řešení.

(-) náročnost konstrukce symbolů a vztahů,

(-) obtížná možnost dodatečných úprav, vede k “překreslení” celého postupu,

(-) není vhodné pro rozsáhlé a složité problémy.

● Textové vyjádření

Nejčastěji používané varianty:

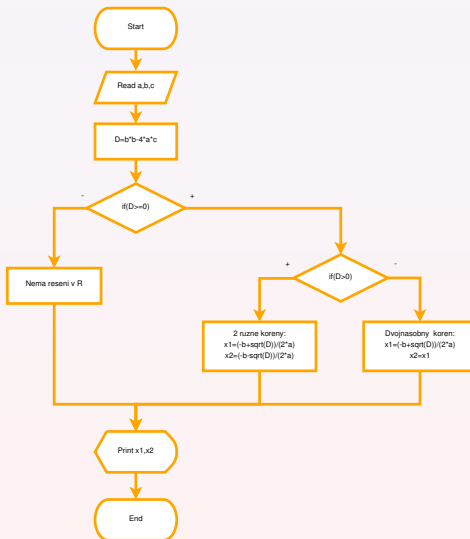
- slovní popis, pseudokód,
- PDL,
- formalizovaný jazyk.

(+) přehlednost, jednoznačnost,

(+) snadný přepis,

(+) možnost modifikace postupu.

15. Vývojový diagram řešení kvadratické rovnice



16. Textové vyjádření řešení kvadratické rovnice

Algoritmus 3: Kvadratická rovnice (a, b, c)

```
1: read (a,b,c)
2:  $D = b^2 - 4ac$ 
2: if  $D > 0$ :
3:      $x_1 = (-b + \sqrt{D}) / (2a)$ 
4:      $x_2 = (-b - \sqrt{D}) / (2a)$ 
5: else if  $D = 0$ 
6:      $x_1 = (-b + \sqrt{D}) / (2a)$ 
7:      $x_2 = x_1$ 
8: else
9:      $x_1 = \emptyset, x_2 = x_1$ 
10: print ( $x_1, x_2$ )
```

17. Jazyky a jejich vlastnosti

Dělení do 2 skupin:

- **Přirozené jazyky**

Komunikační prostředek mezi lidmi.

Skládání slov do vyšších celků není řízeno tak striktními pravidly.

Nevýhodou významová nejednoznačnost některých slov.

Synonyma/homonyma omezují jejich použití při popisu problémů.

- **Formální jazyky**

Vznikají umělou cestou.

Skládání symbolů do vyšších celků řízeno striktnějšími pravidly.

Zamezení významové nejednoznačnosti.

Zástupci:

- programovací jazyky,
- matematická symbolika,
- chemické značky,
- jazyk mapy,
- ...

18. Dělení programovacích jazyků

Kritéria pro dělení programovacích jazyků:

- *Podle způsobu zápisu instrukcí*
Nižší programovací jazyky.
Vyšší programovací jazyky.
- *Podle způsobu překladu*
Kompilované jazyky.
Interpretované jazyky.
- *Podle způsobu řešení problému*
Procedurální jazyky.
Neprocedurální jazyky.

19. Dělení programovacích jazyků

Nižší programovací jazyky:

Programování prováděno ve strojových instrukcích.

(+) Výsledný kód velmi rychlý.

(-) Na programátor ukládá značné nároky (přehlednost, srozumitelnost).

(-) Program je závislý na architektuře procesoru.

Zástupci: strojový kód, Assembler.

```
mov al, 61h
```

Vyšší programovací jazyky:

Používají zástupné příkazy nahrazující skupiny instrukcí.

Tyto následně překládány do strojového kódu.

(+) Struktura programu je přehlednější.

(+) Vývoj programu je jednodušší.

(+) SW není závislý na architektuře procesoru.

(-) O něco pomalejší kód.

První vyšší programovací jazyk: Fortran.

Většina současných programovacích jazyků: Basic, Pascal, C, C++, C#, Java, Python, Ruby.

20. Interpretované jazyky

Využívají *virtuální stroj*, klíčovou součástí *interpreter*.

Program přímo vykonávající instrukce jiného programu z jeho zdrojového kódu.

3 typy interpreteru:

- *Přímé vykonávání zdrojového kódu*
Neprováděna kompilace, nevýhodou pomalost. Zástupce: Basic.
- *Překlad do mezikódu*
Zdrojový kód přeložen do mezikódu (bytekód), nezávislý na operačním systému.
Ten následně vykonán interpreterem.
Zástupce: Python.
- *Just in Time*
Zdrojový kód přeložen do mezikódu (bytekód).
Celek, resp. části, při překladu kompilovány do strojového kódu.
Nevýhoda: prodleva při spuštění programu.
Zástupce: Java, C#.

- (-) Delší spuštění.
- (-) Pomalejší běh programu.
- (-) Větší spotřeba hardwarových prostředků.
- (+) Nezávislost na cílové platformě (kompatibilita).
- (+) Snadnější správa paměti.
- (+) GUI standardem jazyka.

21. Kompilované jazyky

Program je přeložen do strojového kódu.

Poté může být opakovaně spouštěn již bez přítomnosti interpreteru.

Překlad prováděn kompilátorem.

Výsledkem bývá spustitelný soubor (např. exe na platformě Windows).

(+) Kód rychlejší než u jazyků interpretovaných.

(+) Nižší hardwarové nároky.

(+) Běh bez interpreteru (nic se nemusí instalovat).

(-) Přeložený kód není univerzálně použitelný pod různými OS.

(-) Nutnost kompilace pro různé platformy.

(-) Složitější správa paměti.

(-) GUI (knihovny třetích stran).

Zástupci: C, C++, Fortran, Pascal.

22. Procedurální jazyky

Vycházejí z paradigmatu *procedurálního (imperativního) programování*.
Řešení problému popsáno pomocí posloupnosti příkazů na základě algoritmu.
Program tvořen kombinací proměnných, podmínek, cyklů, procedur.
Většina programovacích jazyků umožňuje imperativní přístup.

Dělení do dvou skupin:

- *Strukturované jazyky:*
Algoritmus rozdělen dílčí kroky realizované podprogramy.
Tyto následně integrovány v jeden celek.
Zástupce: C.
- *Objektově orientované jazyky:*
Popis a hledání řešení úloh postupy blízkými lidskému uvažování.
Využití objektového modelu, každý objekt má určité chování a vlastnosti.
Objekty vytvářeny ze "šablon", tzv. tříd.
Mohou spolu komunikovat prostřednictvím rozhraní.
Zástupci: C++, Java, C#.

23. Neprocedurální jazyky

Vycházejí z paradigmatu *deklarativního (neprocedurálního) programování*.

Snaha o eliminaci chyb, které vznikají při vývoji algoritmů.

Definován pouze problém (cíl), popř. vztahy, a nikoliv jeho řešení.
Vlastní algoritmizaci problému řeší překladač.

Příkazy pro opakování (cykly) nahrazeny rekurzí.
Méně časté používání proměnných.

Kód nemusí být zpracováván lineárně (počátek->konec).

Zástupci: SQL, Scheme, Prolog.

S výjimkou SQL jsou méně často používány.

24. Srovnání 4 programovacích jazyků

```
1 def quadratic(a, b, c):
2     D = b * b - 4 * a * c
3     if (D > 0):
4         x1 = (-b + math.sqrt(D)) / (2 * a)
5         x2 = (-b - math.sqrt(D)) / (2 * a)
6         print('x1=' + str(x1) + ', x2=' + str(x2))
7     elif (D == 0):
8         x1 = (-b + math.sqrt(D)) / (2 * a)
9         print('x1=x2' + str(x1))
10    else:
11        print('No solution in R')
```

```
1 void quadratic(double a, double b, double c)
2 {
3     double D = b * b - 4 * a * c;
4     if (D > 0)
5     {
6         double x1 = (-b + sqrt(D)) / (2 * a);
7         double x2 = (-b - sqrt(D)) / (2 * a);
8         cout << "x1=" << x1 << ", x2=" << x2;
9     }
10    else if (D == 0)
11    {
12        double x1 = (-b + sqrt(D)) / (2 * a);
13        cout << "x1=x2" << x1;
14    }
15    else
16        cout << "No solution in R";
17 }
```

```
1 function [] = quadratic(a, b, c)
2     D = b * b - 4 * a * c
3     if (D > 0)
4         x1 = (-b + sqrt(D)) / (2 * a)
5         x2 = (-b - sqrt(D)) / (2 * a)
6         sprintf('x1 = %6.2f, x2 = %6.2f', x1, x2)
7     elseif (D == 0)
8         x1 = (-b + sqrt(D)) / (2 * a)
9         sprintf('x1=x2 %6.2f', x1)
10    else
11        sprintf('No solution in R')
12    end
13 end
```

```
1 public void quadratic(double a, double b, double c)
2 {
3     double D = b * b - 4 * a * c;
4     if (D > 0)
5     {
6         double x1 = (-b + sqrt(D)) / (2 * a);
7         double x2 = (-b - sqrt(D)) / (2 * a);
8         System.out.println("x1=" + x1 + ", x2=" + x2);
9     }
10    else if (D == 0)
11    {
12        double x1 = (-b + sqrt(D)) / (2 * a);
13        System.out.println("x1=x2" + x1);
14    }
15    else
16        System.out.println("No solution in R");
17 }
```

Language	Lines	Words	Characters
Python	11	56	385
Matlab	13	58	392
C++	17	58	430
Java	17	65	473

25. Zápis zdrojového kódu

Editor:

Textový soubor, ASCII bez formátování, nepoužívat textové editory.
Syntax Highlighting, Code Completion.

Python Clustering facility_location.py processResultsOfClustering.py

```
55
56 return U
57
58 def generateCone(a, b, n):
59     #Generate cone
60     t = 0
61     U = []
62     for i in range(n):
63
64         #Point on the base
65         if random.random() < b / (b + (a * a + b * b)**0.5):
66             h, r, t = baseConePoint(a, b)
67
68         #Point on the side
69         else:
70             h, r, t = sideConePoint(a, b)
71
72         #Cone coordinates
```

26. Proces překladač (1/2)

Odlišný přístup pro kompilované/interpretované jazyky.

Preprocesor:

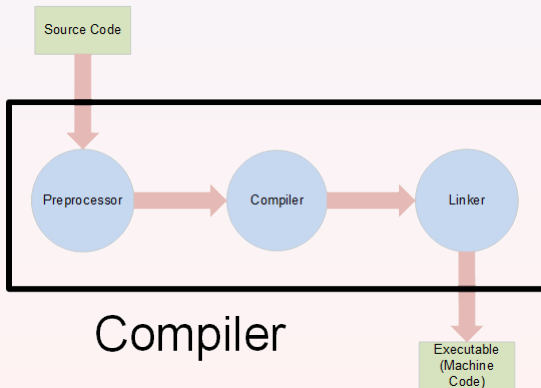
Předzpracování zdrojového kódu, vynechání komentářů, includování souborů, zpracování maker. Výsledkem opět textový soubor.

Kompilátor:

Překlad textového souboru vytvořeného preprocesorem do assembleru.

Následně překlad do relativního kódu (absolutní adresy proměnných a funkcí nemusí být ještě známy), tzv. Object code, *.obj.

Každý programový modul tvořen samostatným *.obj souborem.



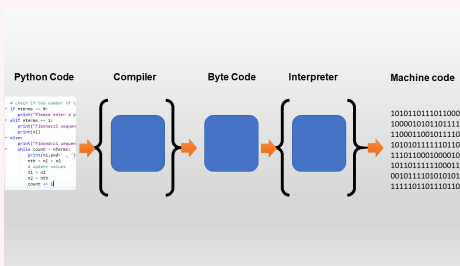
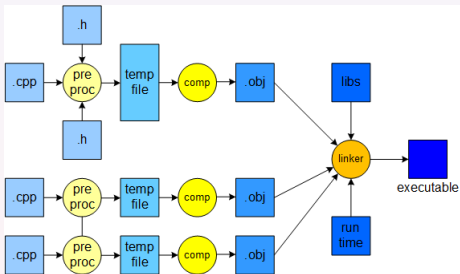
27. Proces překladu (2/2)

Linker:

Spojení jednotlivých *.obj souborů generovaných kompilátorem.

Relativní adresy nahrazeny absolutními.

Výsledkem přímospustitelný kód.



28. Odladění aplikace

Debugger:

Slouží pro ladění programu.

Hledání chyb nastávající za běhu programu.

Nástroje Break Point (Conditional), Watch, Evaluate, Step Into/Over, Run To Cursor...

The screenshot shows the PyCharm IDE with the following components:

- Main Editor:** Displays the code of `parser4.py`. The code is a recursive function `interpret` that processes a tree structure. It handles different operators: `+`, `-`, `*`, `/`, and `:`. The current line of execution is line 120, which is a conditional breakpoint `elif node.data is '-':`.
- Debugger Window:** Located at the bottom, it shows the current state of the program.
 - Frames:** Lists the call stack, including `MainThread`, `interpret, parser4.py:120`, `interpret, parser4.py:118`, and `<module>, parser4.py:140`.
 - Variables:** Shows the current values of variables:
 - `node`: `{(Node) <_main__Node object at 0x0000013f204156a0>}`
 - `left`: `{(float) 1.0}`
 - `right`: `{(NoneType) None}`
 - `self`: `{(Parser) <_main__Parser object at 0x0000013f20302190>}`
 - `cur_token`: `{(NoneType) None}`
 - `i`: `{(int) 8}`
- Evaluate Window:** Located on the right, it shows the result of evaluating the expression `node.data`. The result is `Nothing to show`.
- Terminal:** At the bottom, it shows the output of the program, including the message `Invalid VCS root mappings: The following directories are registered as VCS roots, but they are not: // <Project> \CRLF // <Project> \Bility Location-Problem // <Project> \Facility Location-s // <Project> \gt\Facility location // <Project> \FacilityProblem3 // <Project> \Location-NeurIP2019 // <Project> \Heuristics-2FLP ... (55 minutes ago) 1:16 CRLF UTF`.