XVII

Kde bydlí skutečná složitost?

Po této kapitole přestaneš měřit obtížnost úlohy podle toho, jak cizí ti je syntaxe — a poznáš složitost tam, kde doopravdy bydlí: ne v příkazech, ale v tom, co skládají. A budeš vědět, že o kompozici lze požádat, i když ji sám neumíš napsat.

Předčasná optimalizace je kořen všeho zla.

— Donald E. Knuth, *„Premature optimization is the root of all evil." · Structured Programming with go to Statements,* Computing Surveys *6:4 (1974), §1 — v úplném znění: „...řekněme v 97 % případů; přesto bychom neměli promarnit příležitost v těch kritických 3 %."*

Deset let za krásnou stránku

Roku 1977 dostal Donald Knuth korektury druhého vydání druhého dílu své *The Art of Computer Programming*. Nakladatel mezitím přešel na novou fotosazbu — a stránky byly ošklivé. Matematické vzorce, na kterých Knuthovi záleželo ze všeho nejvíc, vypadaly rozházeně a lacině. Muž, který zasvětil život přesnosti algoritmů, se díval na svoje věty vysázené odbytě a rozhodl se, že si systém na sazbu krásné matematiky napíše sám.

13. května 1977 si napsal memo se základními rysy. Odhadoval, že to stihne za sabatikl — *šest měsíců až rok*. Trvalo to bezmála deset let. Zlom odstavce, o kterém by každý přísahal, že je triviální — kam dát konec řádku — se ukázal být plnou optimalizační úlohou; Knuth ji vyřešil až s Michaelem Plassem v roce 1981. Za sazbou se skrývaly ligatury, kerning, matematické odsazení — vrstvy neviditelného řemesla, které staletí dělali sazeči rukama a nikdo je nikdy nezapsal. Aby dostal správné litery, musel k sazbě přidat ještě program na kreslení písem (METAFONT) a celou rodinu fontů (Computer Modern). Syntaxe systému zamrzla teprve roku 1989.

Knuth nebyl líný ani neschopný. Byl to jeden z nejlepších programátorů své generace a odhad „šest měsíců" dal se vší vážností.

Přesto se spletl dvacetkrát. Ne proto, že by práci podcenil — proto, že *skutečná složitost úlohy nebyla vidět*. Vypadala jako „vysázet stránku hezky“. Bydlela ve vrstvách, které se ukázaly, teprve když se do nich člověk pustil. A přesně o tomhle je celá tahle kapitola: kde bydlí složitost, proč ji skoro nikdy nevidíme dopředu — a proč nás to systematicky vede k tomu, měřit obtížnost úplně špatně.

TeX a LaTeX — sázecí systémy, kterými se dodnes sází většina matematiky a fyziky na světě. Tahle kniha je vysázena příbuzným systémem. Krásná stránka není zdarma — někdo za ni zaplatil deseti lety.

Rozuzlení jedné noci

Teď musím splnit slib z první kapitoly. Vzpomínáš na Excel noc?

> Znám kluka — syna kamaráda — který jednou seděl celou noc nad Excelem. Měl v jednom souboru deset tisíc řádků objednávek, ve druhém seznam čísel, která z nich měl vytáhnout, a úkol znějící nevinně: najít, kolikrát se které číslo v tom velkém souboru objevuje. Nevěděl, jak na to jinak než ručně: najet myší, najít, přepsat, sečíst, další. Řádek po řádku. Když jsem ho ráno potkal, byl bílý jako stěna a měl hotovou možná pětinu. Prošvihl odevzdání. Té noci jsem nebyl u toho, abych mu ukázal to, co ukážu teď — a mrzí mě to dodnes, protože ta noc nemusela být.

Ráno by mu byl stačil jeden řádek. Ne trik, ne kouzlo — jeden řádek, který udělá přesně to, co on dělal myší celou noc, jenže za zlomek vteřiny a bez ohledu na to, jestli má řádků deset tisíc nebo deset milionů:

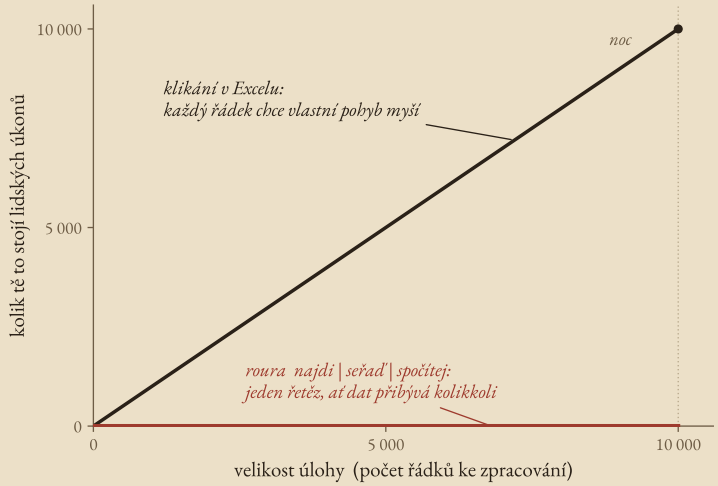
```
najdi | seřaď | spočítej
```

roura (unixová) — svislítko `|`, které pošle výstup jednoho příkazu na vstup dalšího. Příkazy se tak skládají *do řetězu:* `najdi | seřaď | spočítej`. *České pojmenování, ne „pajpa“.*

V praxi: `grep | sort | uniq -c`. *Tři nástroje staré půl století, které na tvém počítači už dávno jsou.*

Tři příkazy oddělené svislítkem. Každý sám o sobě umí jedinou hloupou věc: první *najde* řádky, které tě zajímají; druhý je *seřadí*; třetí *spočítá*, kolikrát se který opakuje. Žádný z nich není chytrý. Kouzlo není v příkazech — je v tom svislítku mezi nimi. Ta roura (`|`) říká: *vezmi to, co vyleze z prvního, a nasyp to rovnou do druhého.* Výstup jednoho je vstup dalšího. A protože všichni tři mluví stejnou řečí — obyčejný text, řádek po řádku — dají se navléknout za sebe jako korálky. Tomuhle navlékání se říká **kompozice**, a je to nejdůležitější slovo celé kapitoly.

Proč tohle porazí noc utrpení, a klikání ne? Protože *roura se skládá, a klikání ne.* Když v Excelu naklikáš pět kroků a přijde šestý, nemáš jak těch pět předchozích spojit do jednoho pohybu — každý řádek si žádá vlastní ruční tah znovu a znovu. Ale tři příkazy v rouře jsou napsané *jednou* a platí pro celý soubor. Přidej čtvrtý příkaz a řetěz se prostě prodlouží; potřebuješ z toho výsledku ještě vytáhnout jen ta čísla nad deset? Přilep pátý článek. Kompozice roste přidáváním, ne opakováním. To je ten rozdíl mezi nocí a vteřinou — a je to rozdíl *v druhu*, ne ve stupni.



Klikání roste s daty*: každý řádek stojí jeden lidský úkon, deset tisíc řádků deset tisíc úkonů — odtud ta noc. Roura je vodorovná: napíšeš ji jednou a je jí jedno, jestli běží na deseti řádcích, nebo na deseti milionech. Neškáluje s daty — proto vyhrává, jakmile dat přibude.*

Podívej se na ten obrázek pozorně, protože je to celá kapitola v jedné grafice. Osa dolů říká, jak velká je úloha — kolik řádků je třeba zpracovat. Osa nahoru říká, kolik tě to stojí *lidských* úkonů. Klikání je ta rostoucí čára: dvakrát víc dat, dvakrát víc noci. Roura je ta vodorovná u dna: pořád stejná, ať dat přibývá jakkoli. Nůžky mezi nimi se rozevírají a od jistého množství dat je propast mezi nocí a vteřinou tak hluboká, že už nejde přemostit vůlí ani kávou. Jde přemostit jedině *jiným druhem řešení.*

„Ve Wordu je to jednodušší“

Tady musím být poctivý, protože kdybych nebyl, byla by tahle kapitola gatekeeping — povýšené mávnutí rukou nad tím, kdo neumí příkazovou řádku. A to je přesně to, co tahle kniha dělat nesmí.

Slyším tu námitku a je oprávněná: *„Ve Wordu je to jednodušší.“* A často je. Když píšeš dopis, pozvánku nebo leták, Word je správný nástroj a příkazová řádka by byla směšná okázalost. Když děláš jednu tabulku o dvaceti řádcích, Excel je rychlejší než jakákoli roura a myš je tvůj přítel. *Na dopis Word stačí* — a kdo ti tvrdí opak, prodává ti svou ješitnost.

Lež není ve „stačí“. Lež je ve „stačí **vždycky**“.

Protože „jednodušší“ platí do první stovky rovnic — a do prvních deseti tisíc řádků. LaTeX vypadá vedle Wordu jako pedantské trápení: místo abys tučné písmo udělal jedním klikem, píšeš podivné povely se zpětnými lomítky. Na první stránku je to nesmysl. Jenže napiš ve Wordu dizertaci se stovkou rovnic, křížovými odkazy a bibliografií — a on ti pod rukama začne přeskakovat čísla, rozhazovat sazbu a přečíslovávat, co nemá. Přesně tam, kde Word taje, LaTeX teprve začíná zpívat: to, co vypadalo jako zbytečná buzerace, byla neviditelná investice do řemesla, které se vyplatí, až úloha přeroste hračku. LaTeX je škola pokory. Učí

tě, že „ošklivě složité“ a „hluboce správné“ vypadají zvenčí stejně — a rozezná je až měřítko.

Kde přesně je ta hranice, se nedá napsat obecně, a právě proto ji tolik lidí mine. Neleží u konkrétního počtu řádků ani rovnic — leží tam, kde se z úlohy udělané *jednou* stane úloha, kterou děláš *pořád*, nebo z úlohy o desítkách položek úloha o desetitisících. Slovo „vždycky“ je zrádné z obou stran: kdo tvrdí, že příkazová řádka je vždycky lepší, je snob; kdo tvrdí, že klikání stačí vždycky, si sám před sebou zamyká celou třídu úloh, o kterých pak bude věřit, že „prostě nejdou“. Poctivá odpověď zní: záleží na měřítku — a úkolem téhle kapitoly je naučit tě to měřítko *vidět*, ne ti předepsat nástroj.

A tady je ta dobrá zpráva, kvůli které tuhle knihu píšu: dnes už si ten děsivý řádek nemusíš pamatovat. `najdi | seřaď | spočítej` ti napíše Claude během vteřiny — správně, i s tou verzí přepínačů, kterou by sis musel roky pamatovat. Nástroj je opravdu úžasný. Ale napíše ti ho jen tehdy, když víš, *o co si říct* — když víš, že úloha „spočítej výskyty“ je roura, ne noc.

→ *kap. 6: LaTeX byl* předepsaný odpor — *bolest, která se vyplatila. Co vypadá jako zbytečná dřina, bývá neviditelné řemeslo. Vyhnout se mu vždycky znamená nikdy nepřerůst hračku.*

To nás vede k tezi téhle kapitoly, a je to teze celého dějství: **problém nikdy nebyl v tom, že neznáš syntaxi. Problém je nevědět, že to jde.** Kdo prožije celý pracovní život v grafickém rozhraní — v okně, kde se kliká — neušetřil čas. Přišel o možnosti. Nikdy se nedozvěděl, že celá třída jeho nocí byla ve skutečnosti roura, protože rouru nikdy neviděl a neměl jak tušit, že existuje. A tady je ta zrada: klikací rozhraní vypadá *snadněji*, a přesně proto je nebezpečné. Nabídne ti tlačítko na každou obvyklou věc a tím tě přesvědčí, že věci, na kterou tlačítko není, prostě nejde — místo aby tě naučilo, že si ji můžeš složit sám. Myš tě vede za ruku a přitom ti bere ze zorného pole celý kraj, do kterého ta ruka nemíří. Syntaxe je dnes zadarmo; napíše ji stroj. Vzácné je vědět, *že něco jde* — poznat v úloze tvar, který se dá složit. To ti stroj nedodá; to je přesně ten úsudek, který se v téhle knize snažíme vybrousit.

Podívej se na tu noc ještě jednou, tentokrát v pseudokódu, protože ten rozdíl je v něm vidět nejostřeji. Klikání je ve skutečnosti tohle — smyčka, kterou místo počítače protáčíš vlastníma rukama, řádek po řádku:

```
pro každý řádek v objednávky:          # 10 000×
ručně, celou noc
    najdi_myší(řádek)
    přepiš(řádek)
    přičti_k_součtu(řádek)
```

Roura je tatáž práce, ale smyčku *nepíšeš* — je schovaná uvnitř každého nástroje, který ji zvládne na miliony řádků sám a bez tebe:

```
najdi("číslo") | seřaď | spočítej    # napsáno
jednou, platí pro vše
```

Není v tom žádná magie. Je v tom jen posunutá hranice: co ve verzi nahoře děláš ty, dělá ve verzi dole nástroj — a ty jsi z otroka smyčky povýšil na toho, kdo ji zadá a zkontroluje výsledek.

Malé nástroje, které se skládají

TEENAGER · MECHANISMUS

Roura z Excel noci není náhoda ani trik jednoho operačního systému. Je to přímý důsledek filozofie, kterou tvůrci Unixu zapsali před půlstoletím a která dodnes drží. Tři pravidla, ze kterých plyne všechno ostatní:

```
# 1) Každý nástroj dělá JEDNU věc, a tu pořádně.
#    (najdi jen hledá; seřaď jen řadí; spočítej
jen počítá)

# 2) Nástroje spolu mluví TEXTEM — společné
rozhraní pro všechny.
#    (výstup jednoho = řádky textu = vstup
dalšího)

# 3) Skládej je rourou; nestav jeden velký
program, který umí vše.
najdi "číslo" objednávky.txt | seřaď | spočítej
```

Klíč je pravidlo 2. Kdyby každý nástroj mluvil jiným jazykem — jeden by vracel tabulku, druhý chtěl na vstupu obrázek — nedaly by se navléknout za sebe. To, že *všichni* mluví obyčejným textem, je ta „společná řeč“, díky které jde libovolný výstup poslat na libovolný vstup. Grafické rozhraní tuhle společnou řeč nemá: tlačítko v Excelu a tlačítko ve Photoshopu si nemají jak nic předat, protože žádný společný výstup neexistuje. Proto se roury komponují a klikání ne — ne kvůli šikovnosti, ale kvůli *rozhraní*.

Unixová filozofie (Doug McIlroy, 1978): „Piš programy, které dělají jednu věc a dělají ji dobře. Piš programy, které spolu spolupracují. Piš programy, které pracují s textovými proudy, protože text je univerzální rozhraní.“

A jak o to požádáš dnešní stroj? V tom je celý fígl — a je snadný, jakmile ho jednou uvidíš. Neříkej „napiš mi program, který spočítá výskyty“. Řekni: *„Postav mi rouru z existujících nástrojů, která spočítá výskyty.“* Rozdíl je propastný. V prvním případě dostaneš padesát řádků nového kódu, který musíš číst, testovat a udržovat. V druhém dostaneš `najdi | seřaď | spočítej` — tři osvědčené nástroje, které na tvém počítači jsou půl století a fungovaly, než ses narodil. Žádat o kompozici z hotového znamená stavět z cihel, které už někdo vypálil a vyzkoušel. Žádat o nový program znamená vypalovat cihly znovu — a tvoje budou horší.

DEMO

`lesk-a-bida-vibecodingu.gaussalgo.com/d/postav-rouru`

Postav rouru: skládej `najdi | seřaď | spočítej | ...` nad opravdovým logem a závoď s klikáním. Uvidíš na vlastní oči, kde se nůžky rozevřou.

Slavný duel: Knuth versus McIlroy

TEENAGER · MECHANISMUS

Roku 1986 dal programátor Jon Bentley ve svém sloupku v *Communications of the ACM* dvěma velikánům stejnou úlohu: *načti text a vypiš n nejčastějších slov i s jejich počty.* Donalda Knutha požádal, ať ji vyřeší ve svém stylu *literátského programování* — kód proložený souvislým výkladem, aby se program četl jako esej. Knuth napsal skvost: přes deset stran promyšleného programu s vlastní datovou strukturou (hašovaný trie), vysázeného jako literatura.

Pak Bentley požádal Douglase McIlroye — jednoho z otců Unixu — o recenzi. McIlroy Knutha ocenil, a pak celý ten skvost shrnul do *šesti příkazů* spojených rourami:

```
tr -cs A-Za-z '\n' | tr A-Z a-z | sort |
    uniq -c | sort -rn | sed ${1}q
```

Rozsekej text na slova, převeď na malá písmena, seřaď, spočítej výskyty, seřaď podle počtu, vypiš prvních n. Šest hotových nástrojů, žádný nový kód.

Je lákavé číst ten příběh jako výprask — génius napsal deset stran, mistr to smázl šesti řádky. Ale takhle ho číst je nefér a míjí to pointu. Knuthovým *zadáním* bylo předvést literátské programování, a on ho předvedl mistrovsky; kdyby dostal za úkol „udělej to co nejkratší rourou“, napsal by ji sám, unixové roury zná lépe než většina lidí na světě. Duel není o tom, kdo je chytřejší. Je o tom, že *stejnou úlohu lze vidět dvěma způsoby* — jako věc, kterou postavíš od základů, nebo jako věc, kterou složíš z hotového. A pro drtivou většinu každodenních úloh je ta druhá cesta správná: ne proto, že bys neuměl napsat program, ale proto, že *nejlepší kód je ten, který jsi psát nemusel*. McIlroyho lekce nebyla „Knuth je horší“. Byla „než začneš stavět, podívej se, jestli to už nejde složit“.

Tahle jediná otázka — *nejde to složit z toho, co už existuje?* — je možná nejužitečnější návyk celé části knihy. Nejde o to naučit se nazpaměť `tr` a `uniq`. Jde o to nabýt reflexu podívat se na každou opakovanou ruční práci a zeptat se: *má tohle tvar roury?* Zbytek, včetně syntaxe, dnes obstará stroj.

Pramen: J. Bentley, „Programming Pearls: A Literate Program“, CACM *29(6), červen 1986, s hostujícími D. Knuthem a M. D. McIlroyem. Knuthův program ve WEBu (Pascal); McIlroyova roura přesně šest příkazů.*

Zákon knihy: žádné ponížení géniů. Knuth splnil své zadání dokonale; lekce je o kompozici*, ne o tom, kdo koho porazil.*

A ten reflex není jen pro programátory — právě naopak. Účetní, který každý měsíc ručně slepuje tři exporty do jednoho přehledu; asistentka, která přepisuje adresy z jednoho systému do druhého; učitel, který ručně počítá průměry ze sto padesáti testů — všichni dělají tutéž chybu jako kamarádův syn nad Excelem. Ne že by byli hloupí; jen nevidí tvar roury, protože jim ho nikdo neukázal a klikací nástroj je přesvědčil, že jinak to nejde. Přitom stačí to opakování popsat vlastními slovy dnešnímu stroji — „mám tři soubory, potřebuju je spojit podle jména a sečíst částky, každý měsíc znovu“ — a on ti vrátí tu rouru, kterou bys jinak hledal půl života. Nemusíš znát `tr` ani `uniq`. Musíš jen vědět, že tvoje měsíční otročina *má tvar*, který se dá složit a zautomatizovat. Rozdíl mezi tím, kdo se ptá, a tím, kdo klape myší, dnes není ve znalosti syntaxe. Je v tom jediném vědění: *že to jde.*

Věž: proč se roury skládají a co je vlastně „složité“

EINSTEIN · MATEMATIKA — přeskočitelné

Kompozice je skládání funkcí. Řekněme to formálně. Příkaz je funkce: vezme vstup a vrátí výstup. Roura `f | g` znamená „proveď f, výsledek podej g“ — a to je přesně matematické *skládání funkcí*:

$$(g \circ f)(x) = g(f(x)).$$

Aby to dávalo smysl, musí obor hodnot f ležet v oboru definice g — výstup prvního musí být přípustným vstupem druhého. V Unixu je tahle podmínka splněná triviálně, protože *každý* příkaz bere text a vrací text: obor hodnot i obor definice jsou tentýž typ, řádky textu. Proto lze skládat cokoli s čímkoli. Skládání je navíc *asociativní* —

$$(h \circ g) \circ f = h \circ (g \circ f),$$

takže na uzávorkování nezáleží a řetěz `najdi | seřaď | spočítej` je jednoznačný. Množina věcí se společným typem rozhraní a asociativním skládáním má v matematice jméno; není náhoda, že roury tvoří *monoid* (s prázdným příkazem `cat` jako neutrálním prvkem).

Grafické rozhraní žádnou takovou algebru nemá: „klik na tlačítko A“ a „klik na tlačítko B“ nesdílejí typ vstupu a výstupu, protože žádný explicitní výstup neexistuje. Bez společného typu není skládání — a bez skládání každou úlohu stavíš od nuly. To není vlastnost lenosti uživatele; je to vlastnost rozhraní.

EINSTEIN · MATEMATIKA — přeskočitelné

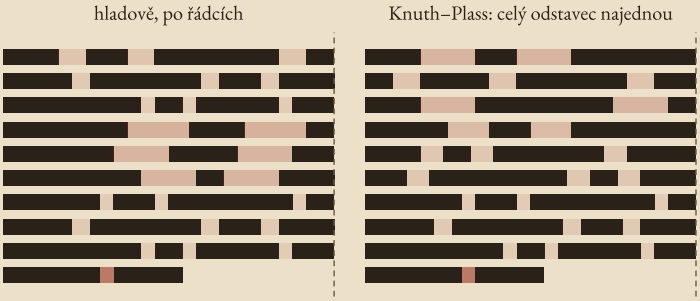
Knuth–Plass: zlom odstavce jako dynamické programování. Vraťme se ke Knuthově deseti letům. Proč je „kam dát konec řádku“ těžké? Naivní, *hladový* postup bere slova, dokud se vejdou, a pak zalomí — rozhoduje se lokálně, řádek po řádku. Jenže lokálně dobrý zlom může vynutit ošklivý řádek o pár řádků níž. Knuth s Plassem (1981) proto neminimalizují jeden řádek, ale *součet* trestů (demeritů) přes celý odstavec najednou:

$$D^* = \min_{\text{zlomy}} \sum_{i=1}^{L} d(r_i), \quad d(r) = (1 + \gamma \cdot \text{napětí}(r))^2,$$

kde napětí(r) měří, jak křečovitě je řádek r natažený nebo stlačený. Klíčové slovo je „najednou“: hledá se posloupnost zlomů s *globálně* nejmenším součtem. Vypadá to jako nemožné prohledávání všech kombinací zlomů — ale není, protože úloha má *optimální podstrukturu*: nejlepší zlom odstavce končící na slově j se poskládá z nejlepšího zlomu končícího na nějakém dřívějším slově i plus jeden řádek $i \to j$. To je přesně rekurence *dynamického programování*:

$$D(j) = \min_{i<j} [D(i) + d(\text{řádek } \ i \to j)].$$

Místo exponenciálního počtu kombinací stačí projít každý zlom jednou. Tak se z „triviálního“ konce řádku vyklube algoritmus — a tak se ukázalo, kde ta složitost celou dobu bydlela.



tmavší mezera = křečovitěji natažený řádek; pravá varianta minimalizuje součet napětí přes všechny zlomy

Vlevo hladově: první řádky se cpou do plna, na pozdější zbude křeč a „řeky“ bílé (tmavší mezery). Vpravo Knuth–Plass: rozhoduje o všech zlomech naráz, napětí rozdělí rovnoměrně, odstavec dýchá. Tatáž slova, tatáž šířka — jiná globální *volba zlomů.*

EINSTEIN · MATEMATIKA — přeskočitelné

Kolmogorovská složitost: kde složitost bydlí doopravdy. Poslední patro. Existuje míra, která říká, jak je úloha *vnitřně* složitá, nezávisle na tom, jakým nástrojem ji řešíš. *Kolmogorovská složitost* $K(x)$ řetězce x je délka nejkratšího programu, který x vytiskne:

$$K(x) = \min\{|p| : U(p) = x\},$$

kde U je pevně zvolený univerzální stroj a $|p|$ délka programu p. Podstatné je tohle: $K(x)$ je vlastnost *úlohy*, ne nástroje. Ať klikáš myší, píšeš rouru, nebo prosíš jazykový model — pod tím vším leží nejkratší popis toho, co má vzniknout, a ten nejde ošidit. „Bezkódové" (no-code) nástroje neubírají složitost; jen ji *přesouvají* z tvého editoru do konfigurace, klikání a skrytých předpokladů. Skutečně nesníží K tím, že schováš kód — jen ho přestaneš vidět. A co nevidíš, to nedokážeš ověřit. Tady se uzavírá kruh s Knuthem: jeho „šest měsíců" podcenilo právě K úlohy „krásná sazba" — složitost, kterou žádné rozhraní nezmenší, jen zviditelní, nebo skryje.

$K(x)$ je nevyčíslitelná (nejde ji spočítat pro obecné x) — přesto je to ta správná pojmová *míra. Pointa pro praxi: složitost úlohy je dolní mez, kterou žádné rozhraní nepodkročí. Nástroj rozhoduje jen o tom, jestli tu složitost* uvidíš*, ne o tom, jestli existuje.*

Kde tedy bydlí

Poskládej to dohromady. Knuth odhadl šest měsíců a strávil deset let, protože složitost sazby nebyla vidět — bydlela ve vrstvách řemesla pod hladkým povrchem „hezké stránky". Kamarádův syn probděl noc nad Excelem, protože neviděl, že jeho úloha má tvar roury — ne proto, že by neuměl napsat `grep`. Grafické rozhraní se komponovat nedá a příkazová řádka ano — ne kvůli šikovnosti, ale protože jednomu chybí a druhé má společnou algebru rozhraní. A žádné „bez kódu" vnitřní složitost úlohy nesníží; jen rozhodne, jestli ji uvidíš, nebo ne.

A kompozice, kterou jsme tu potkali v malém, je zárodek myšlenky, na které stojí celé zbývající dějství. Roura skládá *příkazy* do řetězu textem. O pár kapitol dál uvidíš, že tatáž idea — vzít hotové kusy se společným rozhraním a navléknout je za sebe — postaví celou infrastrukturu: kontejner (kapitola 21) je roura pro provoz, standardizovaná krabice, kterou lze složit s libovolnou jinou, protože všechny mají stejné rohy, za které je vezme jeřáb. Kompozice se nezastaví u tří příkazů; je to páka, která zvedá věci od jedné řádky bashe po celé nasazení. Vidět tvar roury je první krok; ostatní kapitoly jen zvětšují, co se do ní vejde.

→ kap. 21: kontejner a jeřáb. Standardizované rozhraní (rohy krabice) je přesně to, co dělá z příkazů rouru — jen o dvě řády výš. Kompozice pro infrastrukturu.

Skutečná složitost tedy nebydlí v syntaxi. Syntaxe je dnes zadarmo — napíše ji stroj. Bydlí v tom, *co* příkazy skládají, ve vrstvách, které nejsou vidět dopředu, a v nejkratším poctivém popisu úlohy, který nejde ošidit. A protože syntaxe je levná a složitost skrytá, měříme obtížnost systematicky špatně: bojíme se toho děsivého řádku a nevšímáme si noci, kterou

nahrazuje. Tahle kapitola tě učí obrátit to. Neboj se řádku — ten ti napíše stroj. Boj se noci, kterou nevidíš, že jsi jí mohl uniknout.

„Jak bych poznal, že se pletu?“

Konkrétní test k této kapitole: až budeš příště něco dělat *opakovaně ručně* — přepisovat řádky, klikat tentýž sled kroků potřetí, čistit tabulku po tabulce — zastav se a polož si jedinou otázku: *nejde tohle složit z existujících nástrojů jedním řádkem?* Když odpověď zní ano a ty ten řádek napíšeš (nebo si o něj řekneš stroji) a on tvou noc nahradí vteřinou, právě jsi viděl kompozici na vlastní oči a kapitola platí. A když neznáš syntaxi, na tom nezáleží — stačí vědět, *že to jde*, a o zbytek požádej. Kapitola se plete jen tehdy, když se ukáže, že tvoje opakovaná ruční práce *nemá* tvar, který by šel složit — pak ji dělej dál rukama a nestyď se za Word; jen si buď jistý, že jsi tu otázku doopravdy položil, než ses vrátil ke klikání.