# Algoritmy, datové struktury a přínos jejich studia pro vzdělání inženýra

**Jan M Honzík**

FEI VUT v Brně, Údolní 53, 602 00 Brno, Česká Republika

**Summary**

Presented paper overviews and evaluates the general knowledge, capabilities and skills which are resulting from the learning of advanced topics on algorithms and data structures. It is supposed that this knowledge dominantly influences and enriches the profil of engineering oriented professional regardless of his straight participation on building computer software products. The topic is quite actual at the Faculty of Electrical Engineering and Computer Science at TU Brno, where two different curricula are used in the first two years of study. Computer oriented courses are very weak at the electrical oriented curriculum. That situation is hardly acceptable for computer oriented stream of study. The compromise may be found in more balanced rate of electrical oriented and computer oriented courses, common for both streams - electrical enginering and computer science and engineering. That step has serious reason not only in necessity of stronger economical effectiveness, but in widening the engineer profile of future graduates. The paper is devoted to persuate the electrically oriented scholars that programming rather extends the problem solving capabilities of engineers and developes more then software building skills only.

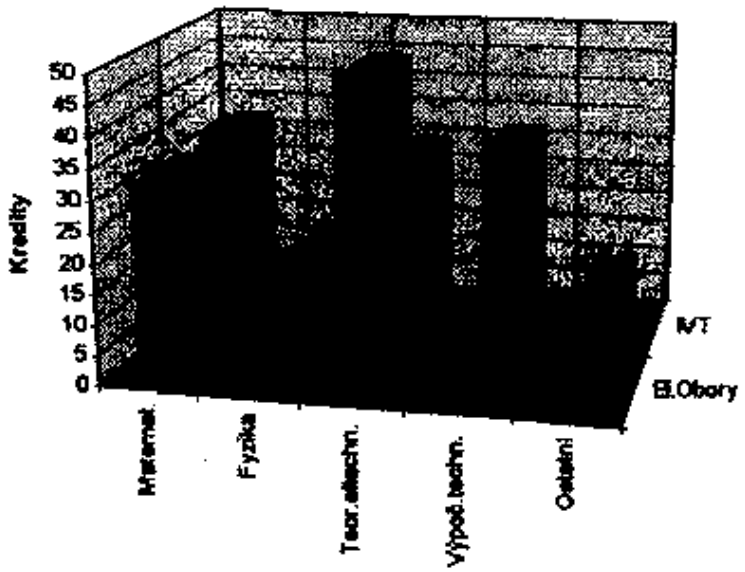
## ÚVOD

Po transformaci fakulty elektrotechnické na fakultu elektrotechniky a informatiky (FEI) VUT v Brně došlo k redukci celkového počtu oborů na fakultě a současně k oddělení studijního plánu (kurikula) oboru Informatika a výpočetní technika (IVT) od ostatních oborů počínaje 2. semestrem studia. Poměr skupin předmětů (matematika, fyzika, teoretické kurzy elektrotechnické, teoretické kurzy orientované na výpočetní techniku a programování a doplňkové kurzy) je znázorněn v obr.1.

Ekonomika vzdělávacího procesu, ale i nové aspekty hodnocení šíře profilu vzdělání vede k úvahám o sjednocení kurikula prvního stupně studia (první dva ročníky) pro všechny obory FEI. Jak je zřejmé z obrázku, podíl kurzů z oblasti výpočetní techniky u elektrotechnických oborů je velmi malý. Tento příspěvek si klade za cíl přesvědčit zejména zástupce elektrotechnického vzdělání, že vyváženější kurikulum s posílením výuky v kurzech výpočetní techniky a programování by bylo přijatelným kompromisem pro oba směry studia a že vzdělání v oblasti algoritmizace a pokročilejších datových struktur je významným obohacením schopností vnímat a řešit problémy i pro inženýra, který nebude tvůrcem programového vybavení. Příspěvek se bude zabývat některými *klíčovými pojmy* disciplíny, které jsou pro podobné obohacení nejpodstatnější. Lze je zařadit do některé z následujících oblastí disciplíny:

- *teoretický základ*
- *metodologie*

- *instrumentace*
- *aplikace*




Obr. 1 Poměr skupin kurzů na elektrotechnických oborech a na oboru IVT

## 1. FORMULACE VZDĚLÁVACÍHO CÍLE INŽENÝRSKÉHO VZDĚLÁNÍ

Odborný profil inženýrského vzdělání umožňuje, aby se absolvent tvůrčím způsobem zapojil do každé z etap produkčního cyklu zboží, určeného pro realizaci v tržním prostředí. Skutečnost, že se absolvent s inženýrským vzděláním uplatňuje v mimoprodukční sféře, lze považovat za nekonfliktní vedlejší jev.

## 2. TEORETICKÝ ZÁKLAD A METODOLOGIE

### 2.1 Pojmy algoritmus, algoritmizace, program, programování

Ranné období tvorby počítačových programů bylo spojeno s intuitivními postupy stanovení uspokojivého "ad hoc" řešení zadaného problému. S ohledem na technická omezení prvních počítačů a na skutečnost, že nejčastěji řešené problémy měly základ v numerickém zpracování čísel, měly programy často podobu přepisu početních postupů používaných bez použití samočinného počítače.

S rostoucí výpočetní silou počítačů, reprezentovanou velikostí použité paměti a operační rychlostí, rostla složitost a rozsah problémů řešených počítači a také složitost a rozsah jejich programů.

Jako důsledek zvětšující se existenční závislosti průmyslových i společenských aktivit na spolehlivě a bezchybně funkčních programech, začalo programování nabývat *charakter přírodovědných disciplín se všemi atributy vědního oboru*. S tvorbou programů úzce souvisí pojmy *algoritmus* a *algoritmizace*.

*Algoritmus* lze neformálně definovat jako *konečnou posloupnost dobře definovaných úkonů, účelně sestavených k řešení daného problému v konečném čase*. Mezi

základní vlastnosti *algoritmu patří obecnost, jednoznačnost a resultativnost*. Formální definice, jejichž použití překračuje rámec pojetí tématu, vycházejí z popisu Turingova stroje, Markovských řetězců nebo z rekurzivních funkcí.

*Algoritmizace* je systematické studium základních metod (technik) používaných pro návrh a analýzu účinných algoritmů. Je to disciplína aplikované matematiky, která učí analyzovat problém, hledat a stanovit jeho řešení a toto řešení přesně a srozumitelně specifikovat formou zápisu algoritmu. Má svou taxonomii, metodiku a teoreticky vychází z různých oblastí matematiky.

*Program* je účelná aplikace algoritmu určená pro řešení daného problému na daném výpočetním prostředku. Program má tvar zápisu ve vhodném programovacím jazyku.

*Programování* je tvůrčí intelektuální činnost, jejímž cílem je vytvoření nehmotného produktu - programu. "Nehmotnost" programu má zásadní význam pro odlišení programových produktů od jiných produktů.

## 2.2 Vývoj tvorby algoritmů

Významnější studium algoritmů bylo zahájeno ještě v předpočítačové době v třicátých letech tohoto století. Matematický výzkum se zaměřoval na formální definici pojmu algoritmus a byla navržena a zkoumána řada formálních modelů tohoto pojmu. Velký důraz se kladl na teorii vyčíslitelnosti, zabývající se analýzou problémů, které jsou řešitelné algoritmicky a specifikací těch, které nejsou. Významných výsledků dosáhl Alan Turing. *Turingův stroj* slouží k jedné z možností popisu algoritmů.

Praktické pojetí algoritmu se spojuje s jeho vyjádřením programem na počítači. Existuje řada praktických aplikací, jejichž řešení na počítači vyžaduje příliš mnoho času a/nebo příliš mnoho paměťového prostoru. Tato oblast se stala předmětem samostatné teoretické oblasti zabývající se *výpočetní složitostí*.

Vytváření zejména rozsáhlejších programů určených k samostatnému prodeji nebo programů, které jsou součástí rozsáhlejších celků, je stejně jako každá průmyslová produkce úzce spjata s metodologií tvorby programů, která se kromě matematicko-algoritmických atributů významně zabývá *ekonomicko-organizačními* aspekty. To je předmětem *softwarového inženýrství* nebo *technologie programování*, které ve studijním programu oboru IVT navazují na algoritmizaci.

## 2.3 Strukturované programování

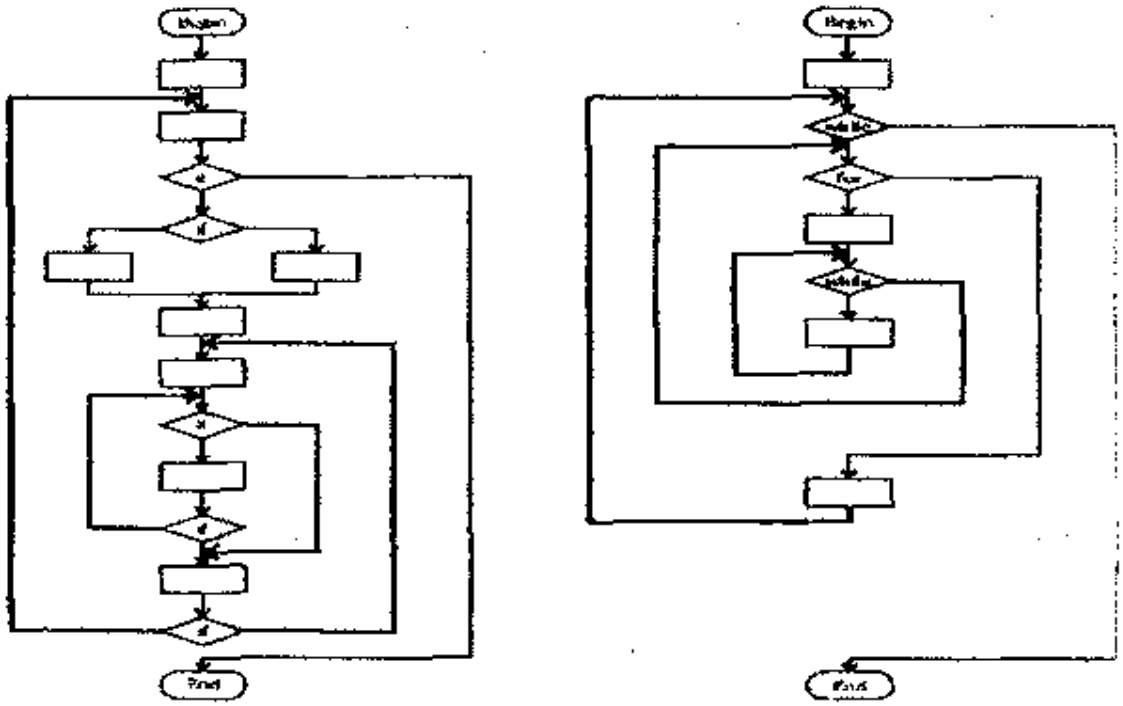
Na přelomu 60. a 70. let se v řadě odborných publikací z oblasti programování začaly formovat závěry prvních ucelených zkušeností teoretického výzkumu algoritmizace i z jeho aplikací ve tvorbě rozsáhlejších komerčně používaných programování. Böhm a Jaccopini svými poznatky o úplnosti algoritmické stavebnice s řídicími strukturami typu "sekvence", "alternativa" a "iterace" dali základ *strukturovanému programování*.

Následující příklad ukazuje rozdíl mezi strukturovaným a nestrukturovaným zápisem algoritmu. Ukázka nestrukturovaného vývojového diagramu Shell sortu je převzata z

Judd,D.R.:" Použitie súborov", Alfa Bratislava 1975, kde autor monografie o algoritmech uvádí, cit.:"...pokud se někdo domnívá, že algoritmu porozumí, ať to zkusí...". Přepis algoritmus výhradním použitím zásad strukturovaného programování je zcela srozumitelný. Každá řídicí struktura je ohraničená a provádí sémanticky samostatně popsatelnou aktivitu.

Zatím co smysl nestrukturovaného diagramu je jen ztěží pochopitelný, na strukturovaném vývojovém diagramu lze zřetelně pozorovat, že nejvnějšnější cyklus snižuje krok sousedních prvků, vnitřnější cyklus obsluhuje jednotlivé sekvence prvků s daným krokem a nejvnitřnější cyklus provádí výměnu prvků sousedních dvojic v dané sekvenci.

Vývoj počítačového programování ovlivnily mimo jiné především díla Niclause Wirtha *"Systematické programování"* a *"Algoritmy + datové struktury = programy"*, Donalda E. Knutha *"Umění programovat"*, Edsgara W. Dijsktry *"Disciplína programování"*. V oblasti nástrojů k zápisu algoritmů pro počítačovou realizaci -



Obr. 2. Vývojové diagramy strukturovaného a nestrukturovaného zápisu algoritmu řadicí metody Shell-Sort

programovacích jazyků, měl v té době zásadní význam jazyk *Pascal* navržený N.Wirthem.

Mezi nejvýznamnější nové principy a nástroje algoritmizace, které přinesl vývoj posledních dvou desitiletí uvedené publikace lze uvést především:

- metodiku *dokazování správnosti* algoritmu
- metodiku *vytváření dokázaných algoritmů* (Dijkstra)
- pojem *abstrakce, struktury a hierarchie*
- *metodiku vytváření programu "shora-dolů"*, postupným zjemňováním abstrakce (top-down, step-wise refinement), která je komplementem k dříve častěji používané metodice *"zdola-nahoru"*

- hodnocení *časové a prostorové složitosti* algoritmů
- *taxonomie paradigmat algoritmů*

## 2.4 Dokazování správnosti algoritmů

Důkaz správnosti algoritmu lze provést více způsoby. Iterační algoritmy lze dokazovat matematickou indukcí, v níž se prověří platnost podmínky pro počáteční stav a pro stav inkrementovaný o 1. Z něj se ustaví podmínka pro konečný stav, která musí splňovat zadání řešeného problému.

Pro dokazování správnosti programu krok po kroku byla zavedena pravidla sémantiky příkazu na základě precedence vstupní a výstupní podmínky příkazu. Dijkstra tyto podmínky rozšířil o pojem *nejslabší počáteční podmínka* - *"weakest precondition"* - wp(S,R), což je nejslabší podmínka, která zaručuje, že mechanismus příkazu S v konečné době splní výstupní podmínku R. S použitím těchto podmínek pak definuje sémantiku všech příkazů - mechanismů, algoritmického jazyka. Malou ukázkou je tzv. sémantická definice středníku, která definuje mechanismus složený ze dvou dílčích mechanismů oddělených středníkem, tedy wp("S1;S2",R) = wp(S1,wp(S2,R)). S jeho pomocí lze dokázat správnost sekvence příkazů, provádějících vzájemnou výměnu hodnot dvou proměnných typu, nad nimž jsou definovány aritmetické aditivní operace: "x:=x+y; y:=x-y; x:=x-y;"

*Příklad:*

wp("x:=x+y; y:=x-y; x:=x-y", (x=X) and (y=Y)) =
= wp("x:=x+y; y:=x-y", wp("x:=x-y", (x=X) and (y=Y))) =
...
= (y=X) and ((x+y)-y)=Y =
= (y=X) and (x=Y) Q.E.D.

Nejvýznamnějším teorémem z oblasti dokazování programů je teorém *invariantní podmínky cyklu,* který umožňuje, aby důkaz správnosti cyklického algoritmu byl co do velikosti úměrný délce zápisu cyklu a nikoli délce provádění cyklu.

## 2.5 Metodika vytváření dokázaných programů

Zatím co předchozí postup dokazoval intuitivně vytvořený algoritmus, E.W. Dijstra vytvořil metodiku, která z výroků o počátečním a konečném - výsledném stavu dovozovala transformační algoritmus. Pro zajištění konečnosti cyklu zavedl Dijkstra pojem *"nejslabší dekrement"* (weakest decrement) wdec(S, t>t), který zajišťoval, že mechanismus S snižuje kladnou celočíselnou hodnotu funkce t, jejíž nulovou hodnotou cyklus končí.

Použití metodiky pro odvození algoritmu binárního vyhledávání v seřazeném poli s možností více shodných klíčů a s vyhledáním krajního ze shodných (tzv. Dijsktrova varianta binárního vyhledávání) je ukázáno v následujícím příkladu:

*Příklad:* "Dijkstrovo" binární vyhledávání v seřazeném poli.
Nechť je dáno pole celých čísel, pro které platí:

A[0] <= A[1] <= ... <= A[N-1] < A[N]

a nechť je dána hodnota klíče x, pro který platí A[0] <= x <A[N].
Nalezněme algoritmus, který ustaví pravdivost Booleovské proměnné SEARCH v případě, že x se rovná hodnotě některého prvku zadaného pole, tedy řešení ve tvaru:

R : SEARCH=(Exist i:(0 <= i< N))(x=A[i])

Vzhledem k seřazenosti pole skončí repetiční proces dosažením podmínky

R' : A[i] <= x < A[i+1]

Pak tedy platí (R' and SEARCH=(x=A[i])) => R

Invariantní relaci zavedeme pomocí proměnné j a předchozího vztahu s cílem, aby P and (j=i+1) => R'. Pak tedy bude o invariantní relaci P platit:

P : (A[i] <= x <A[j]) and (0 <= i <j <= N)

Cílem cyklu je upravovat hodnoty i a j při zachování platnosti invariance P tak, aby se dosáhlo platnosti relace j:=i+1. Program bude mít tedy následující strukturu:

```
i,j:=0,N;   (* Ustavení P *)
do
j<>(i+1) -> *úprava i a j při zachování platnosti invariance P *
od;         (* P and (j=(i+1)) *)
SEARCH:= (x=A[i]);          (* Ustavení řešení R *)
```

Formálním odvozením se dospívá k mechanismu *úpravy* hodnot i a j , který zachovává invarianci podmínky P a vede k ukončení cyklu. Celý program, jehož správnost je formálním postupem jeho tvorby zaručen, má konečný tvar:

```
i,j:=0,N;                   (* ustavení podmínky P *)
do
j<>(i+1) -> m:=half(i+j);   (* platí i<m<j *)
if
A[m]Ăx -> i:=m              (* P and (A[m]Ăx) and (m<j) *)
! x<A[m] -> j:=m            (* P and (x<A[m]) and (i<m) *)
fi
od;                             (* P and j=(i+1) ustavuje R'*)
SEARCH:=(x=A[i])            (* ustavuje R *)
```

## 2.6 Abstrakce, struktura, hierarchie.

*Abstrakce* je jedním ze základních nástrojů, jak zvládat složitost. Podle Hoara *"abstrakce vychází s rozpoznání podobností mezi určitými objekty, situacemi a procesy reálného světa, zaměřuje se na tyto podobnosti a zanedbává, po jistý čas, vzájemné rozdíly"*.

Definici abstrakce lze podle Booche vyjádřit také takto: *"Abstrakce označuje základní charakteristiky objektu, které jej odlišují od ostatních objektů a které odrážejí pohled řešitele"*. Existuje celé spektrum druhů abstrakcí, od nejdůležitějších, až po málo významné:

- *Abstrakce entit* representuje užitečný model entity problémové domény
- *Abstrakce akcí* poskytuje generalizovaný soubor operací, z nichž každá provádí stejný druh funkce
- *Abstrakce virtuálního stroje* sdružuje operace použité na určité vyšší úrovni řízení nebo operace, které všechny používají určitý soubor nižších operací
- *Koincidenční abstrakce* sdružuje operace, které nemají žádný jiný vzájemný vztah než ten, že byly umístěny do téhož modulu Všechny abstrakce mají *statické a dynamické vlastnosti*.

*Struktura* je dalším významným nástrojem k ovládnutí složitých jevů. *Struktura není vrozenou vlastností entit reálného světa. Je to účelově zvolené členění, kterým člověk-řešitel nazírá na složitou soustavu, aby ji snadněji pochopil a ovládnul. Abstraktní datová struktura*, která patří mezi abstrakce entit, je prvkem specifikovaným *abstraktním datovým typem* (ADT). Ten je definován *množinou hodnot, kterých mohou prvky tohoto typu nabývat a množinou operací, které jsou nad prvky tohoto typu ustaveny*.

Mezi významnější vlastnosti, které u ADT sledujeme patří:

- *možnost hierarchického uspořádání*
- *homogennost-heterogennost*
- *statičnost/dynamičnost*
- *lineárnost-nelineárnost uspořádání*
- *přímý-sekvenční přístup* k prvkům

*Hierarchie v datových strukturách* dovoluje, aby komponentou struktury byla opět struktura. Tento rekurzivní mechanismus významným způsobem obohacuje možnosti vytváření abstraktních datových typů mapujících libovolný objekt problémové domény.

Studium vlastností a návrhu ADT patří k nejvýznamnějším partiím moderní algoritmizace. Vývoj programovacích jazyků lze sledovat na vývoji prostředků nabízených k použití i vlastní výstavbě ADT.

S pojmy *abstrakce, struktura a hierarchie* je úzce svázáno vytváření programu metodou *"shora-dolů"*, postupným zjemňováním abstrakce. Tvorba rozsáhlejších programových celků vyvolala potřebu zvládnout stále se zvětšující složitost programů. Významným pokrokem byla tvorba programů metodou "shora-dolů" s postupným zjemňováním abstrakce navržených bloků. Tato metoda úzce souvisí se strukturovaným přístupem a aplikuje strategii *"rozděl a panuj"*. Výsledný produkt měl

jasně definovanou strukturu, v s nízkou spřažeností stavebních modulů a s jejich jasně vymezeným hierarchickým uspořádáním. Tato metodika umožňovala účelnou dělbu práce v týmu při analýze, návrhu i ladění rozsáhlého programu.

## 2.7 Výpočetní složitost algoritmu

Prostor a čas potřebný k výpočtu programu se s rostoucí velikostí stal významným kritériem kvality algoritmu. Počáteční, většinou experimentální způsoby hodnocení daly vzniknout teoretickým úvahám o složitosti algoritmů.

Jedním ze způsobů hodnocení časové složitosti je použití virtuálního stroje, s určováním počtu provedení jeho základních operací. Pro vzorek vstupních dat o kardinalitě "n" lze pak stanovit minimální, maximální a průměrný počet jednotlivých operací, a z toho i odpovídající časové nároky. Původní výsledky analýzy řadicích algoritmů publikované v [5] prokázaly vysokou korelaci mezi experimentálními a analyticky získanými hodnotami.

Asymptotickou notaci, využil Knuth (1976) pro klasifikaci časové složitosti algoritmu. "Velké Omikron" se stalo synonymem pro horní omezení časové náročnosti algoritmů. Odvozování asymptotické složitosti je pro řadu algoritmů náročným matematickým procesem.

Porozumění smyslu časové složitosti algoritmu vede k pochopení skutečnosti, že počítač s vyšším řádem výpočetní rychlosti je pro pomalý algoritmus méně významný, než algoritmus řešící stejný problém s nižším řádem složitosti. Klasickým příkladem je rekurzívní a iterační zápis algoritmu pro výpočet Fibonacciho čísla, kdy pro rekurzívní algoritmus lze odvodit časovou složitost řádu Omikron($\Psi^n$) (kde $\Psi$ je hodnota "zlatého řezu"), zatím co pro iterativní algoritmus je složitost lineární. S využitím principu *dynamického programování* je známá verze tohoto algoritmu s logaritmickou složitostí.

## 2.8 Taxonomie paradigmat algoritmů

Algoritmy lze klasifikovat různými způsoby. Lze je rozdělovat do tříd podle základního principu, který je použit k řešení zadaného problému. Vnímáme-li algoritmus jako proces transformující vstupní údaje na výstupní, pak algoritmus úzce souvisí s údaji, které reprezentují stav procesu v jeho jednotlivých krocích - s použitou datovou strukturou. Objektově orientovaný přístup chápe algoritmus jako representaci jistého chování objektu v jeho modelové doméně. Nahlédneme-li do monografií nejznámějších autorů v oblasti algoritmů a datových struktur, pak nejčastějšími kritérii, podle nichž tito autoři člení obsah své publikace a nazývají jejich kapitoly jsou:
a) princip - paradigma, na němž je algoritmus založen
b) typ datové struktury, s nímž algoritmus pracuje
c) aplikační zařazení algoritmu

Porovnáním uspořádání obsahu neznámějších publikací o algoritmech, není snadné rozčlenit algoritmy jediným z uvedených kritérií. Jako příklad *tříd podle prvního kritéria* jsou uvedeny tři skupiny algoritmů.

*Nenasytné algoritmy* ("greedy algorithms") zahmují nejčastěji grafově orientované algoritmy používané k optimalizačním úlohám. Do této kategorie patří řada algoritmů, v jejichž principu se vyskytuje abstraktní nebo reálná grafová struktura. Podmnožinou jsou algoritmy "s návratem", které většinou neaplikují poslední krok - kritérium výběru optimálního řešení.

Algoritmy třídy *"rozděl a panuj"* ("divide and conquer"), řeší daný problém dekompozicí jeho instance na řadu menších subinstancí téhož problému. Kombinací získaných dílčích řešení získávají řešení celku. Tyto algoritmy často používají k postupné dekompozici až po hraniční velikost subinstance mechanizmu rekurze. Účinnost této techniky závisí na účinnosti nalezení řešení subinstanci. Mezi nejznámější algoritmy této třídy patří např. binární vyhledávání, řazení setřiďováním nebo Quicksort.

Algoritmy třídy "rozděl a panuj" jsou typické svým postupem "shora-dolů", který aplikují při dekompozici. Algoritmy třídy označované jako *algoritmy dynamického programování* se naopak vyznačují postupem "zdola-nahoru". Začínají s nejmenšími možnými subinstancemi. Jejich kombinací získávají řešení subinstanci zvětšujícího se rozměru, až dospějí k řešení celé zadané instance. Tak lze např. postupovat při výpočtu binomického koeficientu (postup známý z Pascalova trojúhelníku), nebo při výpočtu prvku Fibonacciho posloupnosti, kdy algoritmus "shora-dolů" vede k exponenciální časové složitosti, zatím co při postupu "zdola-nahoru" lze získat časovou složitost logaritmického řádu. S pojmem dynamické programování jsou nejčastěji spojovány některé algoritmy pro hledání nejkratší cesty mezi dvěma uzly orientovaného grafu nebo problém "obchodního cestujícího", který hledá nejkratší cyklický průchod (smyčku) všemi danými uzly počínaje (a konče) jedním zadaným uzlem. Do této skupiny však patří také řada jiných algoritmů, jako např. zřetězený součin matic aj.

*Druhé kritérium* lze charakterizovat např. výčtem různých datových struktur, operace nad nimiž jsou vyjádřeny základní nebo složitější algoritmy. Tak lze definovat algoritmy pracující s celými čísly, s grafy, seznamy, stromovými strukturami, soubory, textovými řetězci atd.

*Třetí kritérium*, vytváří skupiny orientované do stejné aplikační oblasti. Toto kritérium je jedno z nejčastěji používaných. Podle něho lze vymezit např. třídy numerických algoritmů. Celý jeden svazek Knuthova díla byl pojmenován podle dvou tříd - vyhledávání a řazení. Jsou známy algoritmy grafického zobrazování, algoritmy pro rozpoznávání vzorků, algoritmy pro práci s texty, algoritmy her atd. Existují i další, méně významná kritéria taxonomie algoritmů.

## 3. INSTRUMENTACE A APLIKACE

Výuka algoritmů a tvorba programů má ojedinělou bohatost forem inženýrské práce. Teoretické poznatky se bezprostředně prověřují na implementaci algoritmů v podobě ucelených malých programů. Součástí výukového cyklu je projekt, který má všechny podstatné náležitosti programového díla.

### 3.1 Programovací jazyky a vývojové prostředí

Programovací jazyk je podstatným nástrojem pro zápis algoritmu do jeho proveditelné podoby ve tvaru programu. S ohledem na tradice, srozumitelnost, sebeobranné rysy i aplikační použitelnost je stále Pascal považován za jazyk vhodný pro výuku algoritmizace těch studentů, jejichž oborové zaměření nepředpokládá přímou účast na tvorbě profesionálního programového vybavení. Studenti, kteří se naopak zaměřují na tvorbu programového vybavení, musí kromě jazyků Pascal, C nebo C++ , ovládnout nebo seznámit se s řadou dalších obecných i specializovaných programovacích jazyků a systémů.

### 3.2 Tvorba rozsáhlých programů, nástroje a metodika

Tematikou bezprostředně navazující na algoritmy a datové struktury je tvorba rozsáhlých programů. Je to již plně inženýrská disciplína s akcentem na ekonomicko-manažerské aspekty tvorby programů. S tvorbou algoritmů a datových struktur nejvíce souvisí metodika *modulárního programování a objektově orientovaného programování.*

*Programový modul* je definován jako "schránka" na abstraktní entity, která poskytuje možnost samostatného (nikoliv nezávislého) překladu, ukrytí vnitřní implementace algoritmů a dat a dobře definovaného rozhraní. Modul se stává, podobně jako v ostatních technických oblastech, "fyzickou" stavební jednotkou programu. Předmětem modulárního programování je seznámení se s nástroji tvorby modulů a s racionálním stanovením obsahu a rozsahu jednotlivých modulů. Nástrojem modulárního programování je některá verze Pascalu z produkce firmy Borland nebo Wirthův jazyk Modula-2.

*Objekt* je definován jako datová nebo funkční abstrakce, která má základní vlastnosti objektové orientace a to:

- ochranu a neviditelnost vnitřní struktury - *zapouzdření*
- *dědičnost*
- *polymorfismus*, realizovaný nejčastěji pozdní vazbou

S ohledem na skutečnost, že tvorba rozsáhlých programů je disciplínou, která navazuje na algoritmizaci a datové struktury, nebude zde tato problematika pojednána šířeji. Modulární i objektově orientované programování je předmětem samostatného kurzu nabízeného především v kurikulu oboru IVT.

### 3.3 Dokumentace a prezentace programů

Neoddělitelným požadavkem již v počátcích programování je tvorba *dobře dokumentovaných programů*. Dokumentací se rozumí nejen věcné komentování zdrojového textu včetně všech identifikačních náležitostí textového souboru v jeho záhlaví, ale i pomocné vytvoření pomocného souboru jako "Read.me" nebo "Cti.me", které detailně popisují soustavu souborů vytvářejících řešení a usnadňují dobrou navigaci.

*Prezentace výsledků a komunikace* programu s uživatelem patří k základním vlastnostem dobře vytvořeného programu. Mezi prioritní požadavky patří jednoduchost, robustnost a srozumitelnost komunikace a zobrazení, která dává přednost vtipu a kreativitě před grafickou nabubřelostí a nabobtnalostí snadno získatelnou z nejrůznějších komerčně dodávaných vývojových nástrojů.

### 3.4 Etika inženýra informatika

Po lékařství je informatika a výpočetní technika snad prvním dalším přírodovědně orientovaným oborem, který se vážně zabývá etikou své profese. Hippokratova přísaha adeptů lékařství má historickou tradici. O informatické verzi Hippokratovy přísahy pro absolventy oborů výpočetní techniky se diskutuje na řadě univerzit. Tam, kde v kurikulu není zařazen samostatný kurz etiky a psychologie programování, je nutné dílčí problémy zařadit do všech kurzů zabývajících se programováním a to i pro studenty, jejichž vlastní profesí nebude tvorba software. Informační systémy realizované výpočetní technikou se staly nervovou soustavou lidské společnosti. Možnost jejich zneužití je tak závažným problémem, že každý i neformální rozbor možných příčin a následků, posilující morální aspekty a zodpovědnost budoucích mladých inženýrů, je důležitou složkou jejich vzdělání.

## 4. SHRNUTÍ OBECNÝCH PŘÍNOSŮ DISCIPLÍNY PRO VZDĚLÁNÍ INŽENÝRA

Algoritmy a datové struktury jsou aplikovanou matematickou disciplínou. Její postavení v kurikulu inženýra vzdělávaného na fakultě informatiky a výpočetní techniky je podobné disciplínám teoretických základů elektrotechniky. Příspěvek záměrně uvádí více příkladů z teoretických oblastí algoritmizace, aby demonstroval příspěvek kurzu z této oblasti do teoretické výbavy inženýra. O praktičnosti programování nemá totiž většina zúčastněných pochybnosti.

Mezi nejvýznamnější přínosy studia algoritmizace a datových struktur lze uvést: *analytické hodnocení i syntetizující postupy* při práci se *strukturami a hierarchií* jejich vrstev a komponent. Významně se uplatňuje *prostoročasové vlastnosti struktur* (statičnost-dynamičnost, přístup ke komponentě). Teoreticky i prakticky se studují *iterace a rekurze*, což jsou praprincipy přírodních jevů i života jako takového. Řešení problému je podrobováno *časoprostorovým kritériím*, což je základ budoucího ekonomického hodnocení každého produktu. Počítač poskytuje nesmírně účinnou *laboratoř*, která dovoluje těsné, rychlé a efektivní *ověření postulátu experimentem*. Kreativitu tvorby základních algoritmů doplňují exaktní *metody verifikace správnosti* řešení. Skryté prostředí vlastního řešení problému v počítači vyžaduje tvorbu efektivní *komunikace mezi člověkem a počítačem*. Tato problematika se úzce váže na *psychologii i estetiku*, které výrazně ovlivňují výsledný efekt použití programů. *Modelování reality programem* prohlubuje poznání vlastní reality. Tato skutečnost se projevuje již při začátečnických pokusech o realizaci algoritmů her, jejichž intelektuální náročnost je velmi malá, ale algoritmizace není jednoduchá (např. Člověče nezlob se!). Instrumentář algoritmizace a programování poskytuje stavební kameny, které zdaleka nejsou ve svých makro rozměrech tak fyzikálně omezeny, jako každé jiné stavebnice pro inženýrské činnosti jiných oborů. Lze bez nadsázky říci, že v oblasti programového vybavení již existují a lze dále *tvořit nejsložitější a nejrozsáhlejší produkty lidského intelektu*. To zákonitě vede k vážným úvahám o

*spolehlivosti, robustnosti, bezpečnosti, vedlejších jevech a společenských dopadech* programových produktů.

Všechny tyto, a mnohé další fenomény, jsou probírány nebo zmiňovány v rámci pokročilého kurzu algoritmů a datových struktur a lze tedy vyjádřit domněnku, že tento kurz, který je nezbytný pro budouciho tvůrce programových systémů, je výrazným přínosem k obecně formulovanému profilu inženýra kteréhokoli oboru. O to více to platí v době, kdy v nejvyspělejších zemich, pod záštitou nejautorizovanější celosvětové organizace IEEE, vzniká nové a klíčové mezioborové studijní zaměřeni nazývané *"Inženýrství systémů ovládaných počítačí"* (Engineering of Computer Based Systems). To bude vyžadovat úzkou spolupráci inženýrů dokonale ovládajících počítače a rozumějících i ostatním technickým principům a inženýrů pro nejrůznější technické obory, kterým jsou základní vlastnosti počítačů a algoritmů známy důvěrněji, než z konzumně uživatelského úhlu.

**LITERATURA**


[1]Cormen,T.H.,Leiserston,E.C.,Rivest,R.: Introduction to Algorithms: The MIT Press. McGraw-Hill,1990

[2]Kruse,R.L.: Data Structures and Program Design. Prentice-Hall, 1987

[3]Brassard,G.,Bratley.: Algorithmics, Theory & Practice. Prentice-Hall,1988

[4]Baase,S.: Computer Algorithms, Introduction to Design and Analysis. Addison-Wesley,1988

[5]Honzik J.: Programovací techniky. Ediční středisko VUT v Brně, 1985

[6]Honzik,J: Dokazováné správnosti programu. In. Sborník konference "Programování 83", DT ČSVTS, Ostrava 1983

[7]Honzik,J.,Hruška,T.: Abstraktní typy dat a jejich využití ve výuce programování. In. Sborník konference "Ladění 85", DT ČSVTS, České Budějivice 1985

[8]Honzik,J.M.: Etika programování. In. Sborník konference "Programování 93", DT ČSVTS Ostrava,1993