Predmet: **Algoritmizácia a programovanie**

2202 FEI Telekomunikácie

Ročník: 1.ročník

Rozsah: 3/2 ZS

Prednášajúci: Jiří Pospíchal

Anotácia Predmetu:

Úvod do algoritmizácie . Základne vlastnosti algoritmov. Vybrané algoritmy usporiadania a vyhľadávania. Úvod do programovacích paradigiem. Cieľom predmetu je zvládnutie tvorby jednoduchých algoritmov a tvorba jednoduchých programov v jazyku C. Zoznámiť sa s základnými údajovými typmi a štruktúrami, naučiť sa ich používať. Naučiť sa používať vybrané algoritmy usporiadania a vyhľadávania. Získať prehľad o súčasných programovacích paradigmách.

Z čoho študovať?

Jedna najlepšia kniha neexistuje

Wirth, N.: Algoritmy a štruktúry údajov. Bratislava, Alfa 1990.

Herout P.: Učebnice jazyka C . KOPP, České Budejovice, 1994

Topfer, P.: Algoritmy a programovací techniky, Prometheus, Praha 1995.

Čo máte ďalej z programovania a algoritmov?

2. ročník: Systémové programovanie a asemblery

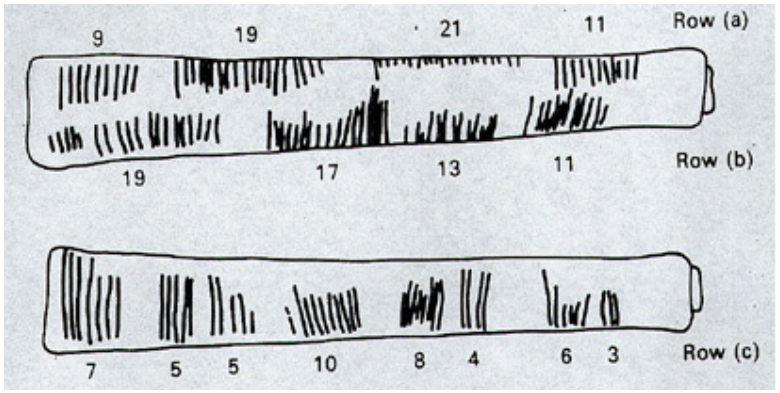
4. ročník: Programovacie techniky - Povinne voliteľný

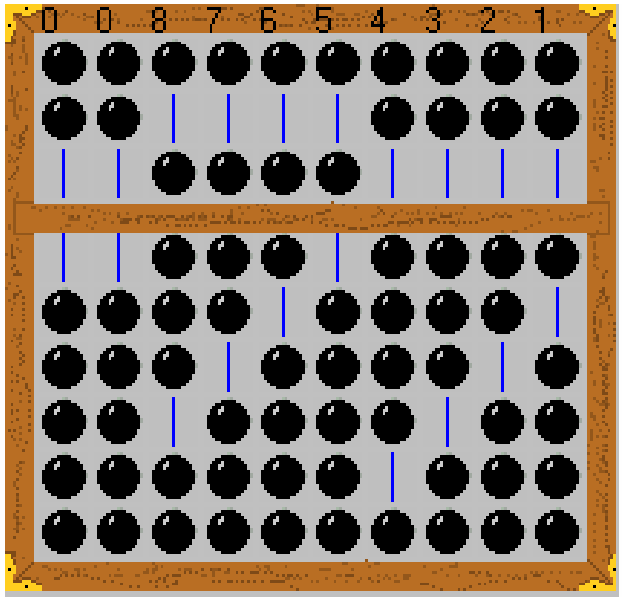
4. ročník: Grafové štruktúry a algoritmy - Povinne voliteľný

Trocha histórie:

kamienok = kus dobytka ang. „*calculator*“ z latinského „*calculus*“,

rováš, 30000 r. stará kosť s zárezmi	abakus z Babylónie, bežný 6. st. pnl. v Grécku (zmienený napr. Herodotom), doteraz v Rusku a na ďalekom východe. ("Program" je schovaný v mysli expertov)





Počítače a stroje - pomôcky na uľahčenie práce fyzickej aj duševnej

Základom strojov je usporiadanie, organizácia malých jednotiek (ľudí, koliesok, tranzistorov) a postupnosť ich práce

Fantázia:

V Ílias zlaté slúžky "Vedeli rozmýšľať, vravieť, dokonca mali aj silu".

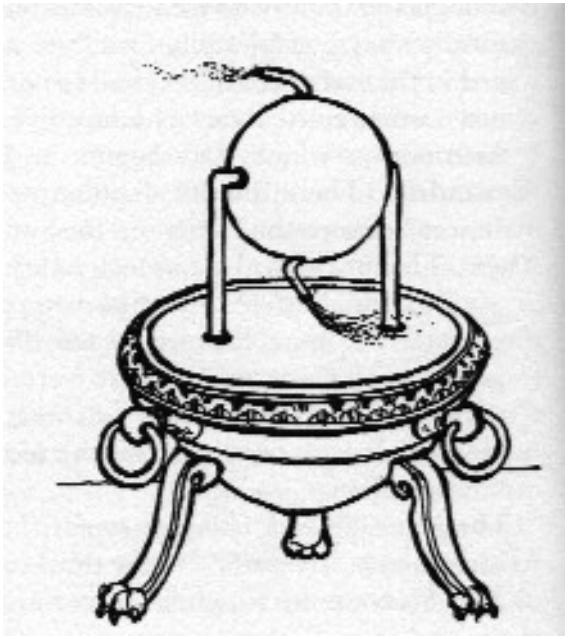
V Paríži navrhol Reymund Lullus (1235-1315) "stroj pravdy" na základe presvedčenia o formalizovateľnosti logických operácií.

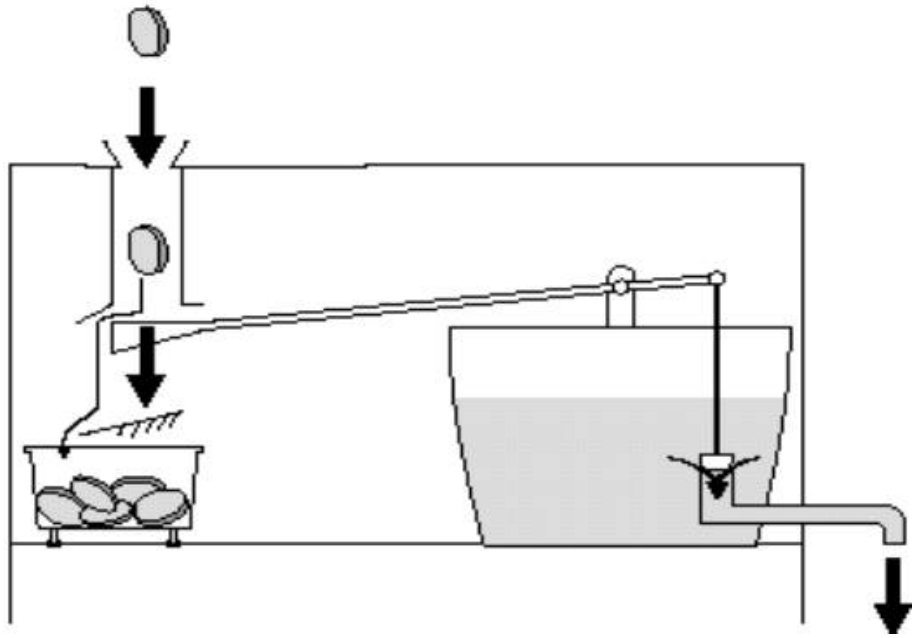
Albert Veliký spolu so svojím žiakom *Tomášom Akvinským* zhotovili pomocou alchýmie umelého človeka - androida.

V 14. storočí *Eliov Chelm* vytvoril z hliny golema, ktorého v 16 storočí oživil pražský rabín *Jehuda Lov ben Becalél*. Chýrny lekár a alchymista Theophrastus Bombastus von Hohenheim zvaný *Paracelsus*, ktorý tiež r. 1537 navštívil Bratislavu, opísal postup, ako v sklenenej nádobke zo spermy získať homunkula.

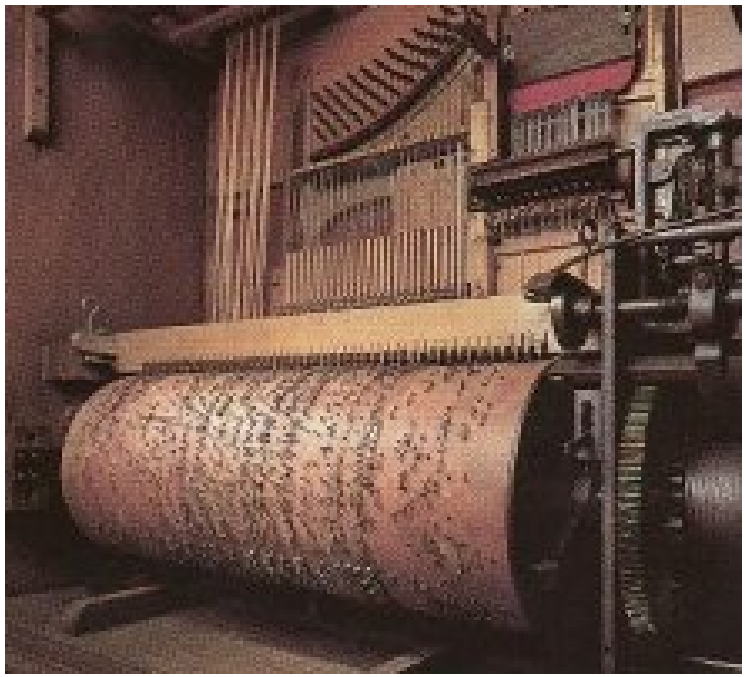
Realita

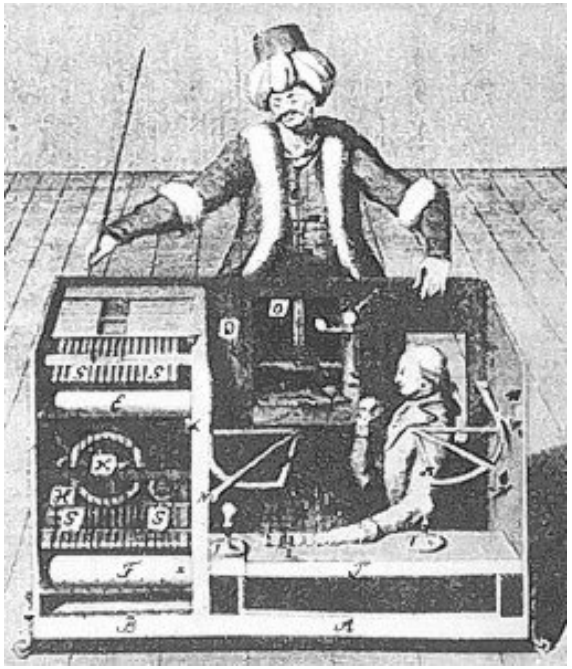
Heron Alexandrijský v prvom storočí pnl. vyrábal hydraulicky poháňané mechanizmy.





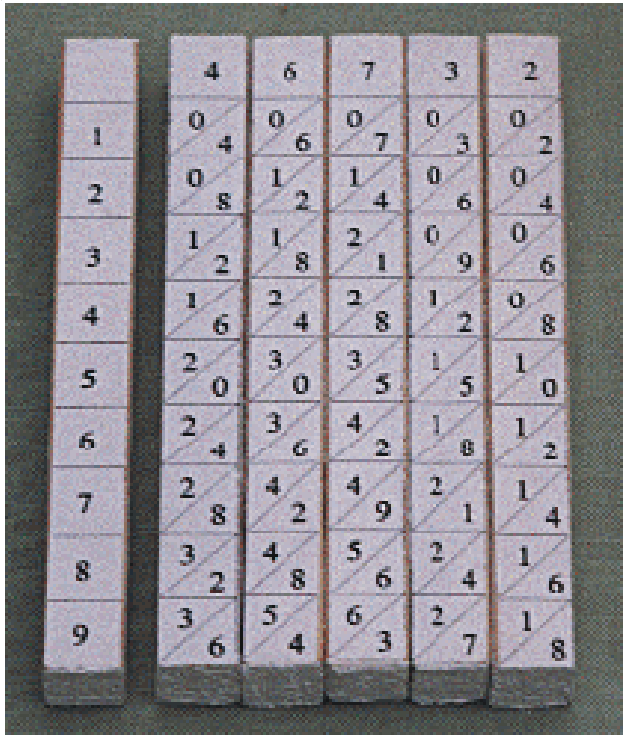
Programové riadenie automatov valčekom s klinčekmi sa používalo v 18. storočí v hračkách a orchestriónoch.

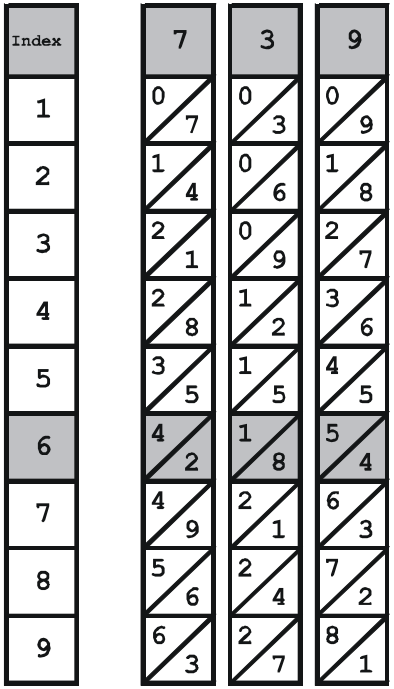




Šachový automat Johanna Wolfganga Kempelena, vytvorený v Bratislave a predstaveným r. 1770 Márii Terézii vo Viedni.
Predpokladá sa, že v stroji bol skrytý človek malého vzrastu, ovládajúci šach.

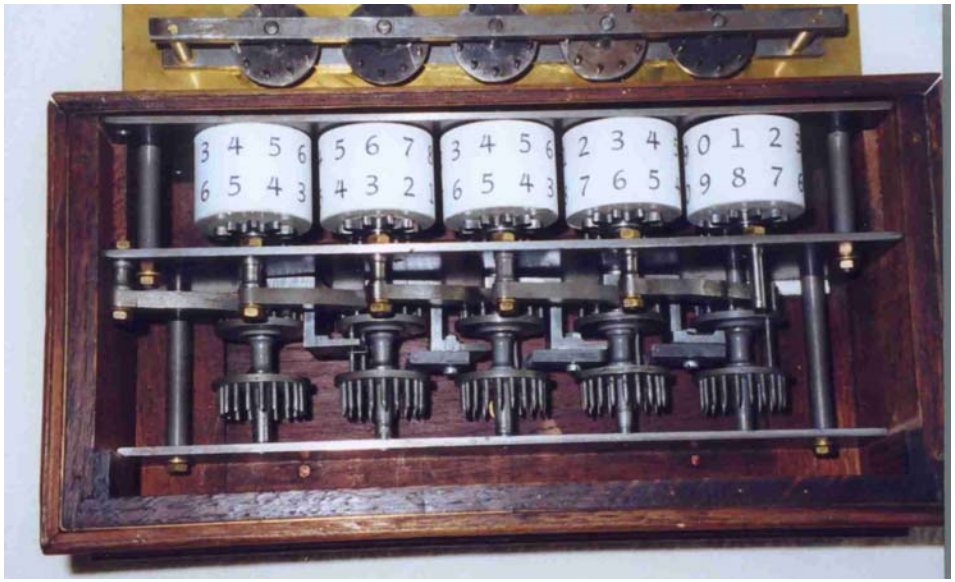
Uľahčeniu výpočtov v 15-16. storočí - doba zámorských obchodných ciest, veľa výpočtov v navigácii a pohybu planét, v 17 st. škótsky matematik John Napier "Napierove kosti" na násobenie, logaritmy e na prevod nás., del. A odm. Na sčítanie a odčítanie, Briggs logaritmy o zákl. 10, nasledovalo log. pravítko.

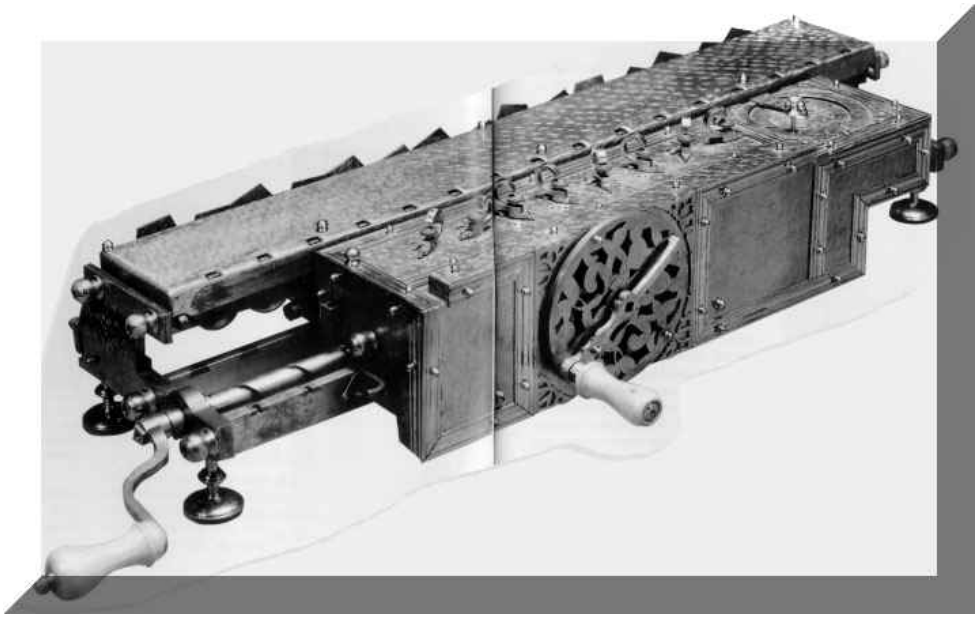





```
6 x 7   = 4 2      (6 x 700)
6 x  3  =   1 8    (6 x 30)
6 x   9 =     5 4  (6 x 9)
-------------------
6 x 739 = 4 4 3 4
          | | | |
          | | | jednotky
          | | desiatky
          | stovky
          tisícky
```

Prvé mechanické kalkulátory: nemec Wilhelm Schickard (1592-1635), sčítanie a odčítanie, ozubené kolieska, numerický displej. Zomrel na mor, zabudnutý. Francúzsky matematik a filozof Blais Pascal (1623-1662) pre otca zaťaženého účtovníckou robotou 19-ročný začal stavať stroj Pascaline (50 kusov vyrobených). V knihe *Myšlienky*: "*Aritmetický prístroj dáva výsledky, ktoré sa blížia mysleniu viac než všetko, čo robia živočíchy; nerobí však nič, aby sme mohli povedať, že má vôľu ako človek.*"





Gottfried Wilhelm von Leibnitz, matematik, logik a filozof, spolutvorca diferenciálneho a integrálneho počtu s Newtonom a autor v súčasnosti používaného symbolizmu, postavil r. 1673 stroj, ktorý dokázal automaticky tiež násobiť a deliť. Leibnitz je tiež autorom myšlienky o výhodách dvojkovej sústavy.

16-17. st. - prielom vo vývoju mechanických zariadení k výpočtom, princípy mechanických kalkulátorov do polovice dvadsiateho storočia. Nedostatky:

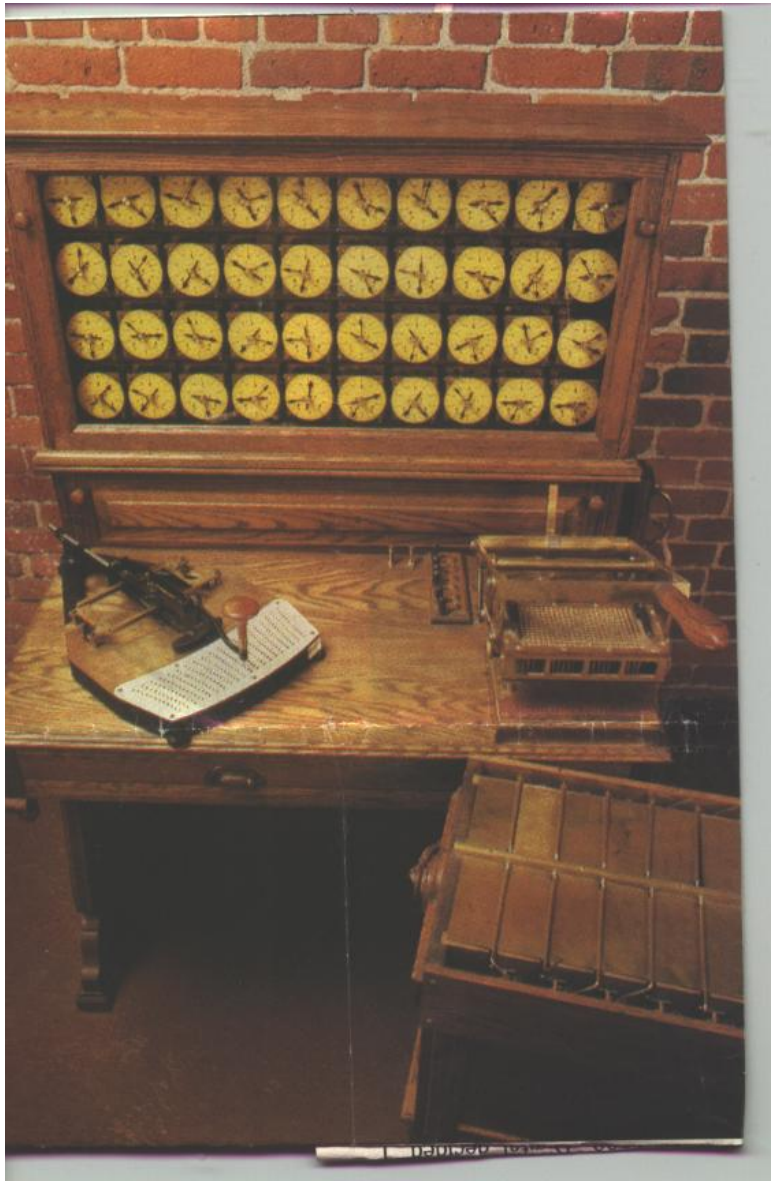
1. Neexistoval koncept programu, takže sada rovnakých krokov pre jedny vstupné dáta musela byť opakovane ručne zadávaná pre iné vstupné dáta.
2. Väčšinou neexistuje pamäť, medzivýsledky na papier a ručne zadávané
3. Z technologického hľadiska ide čisto o mechanické stroje bez použitia elektroniky.

Vy ste moje najlepšie pamäťové médium, Marienka!

Tkáčske stroje a program

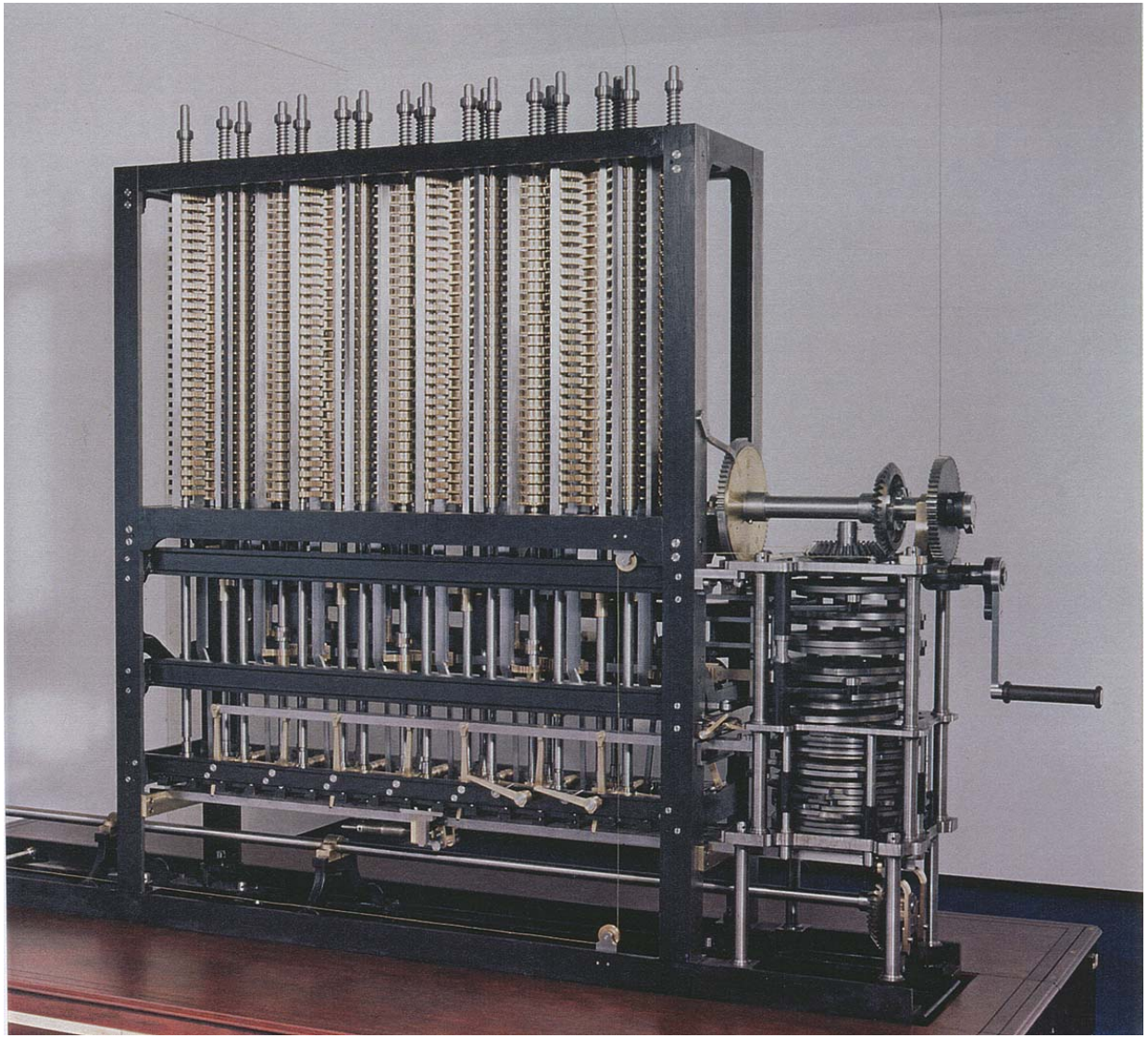
Jacque de Vaucanson navrhol opakovať vzory na látkach pomocou papierových kariet v podobe dierkovanej šablóny, po 70 rokoch r. 1801 realizoval Joseph Marie **Jacquard**. Ihly systémom dierok na kartónových kartách preťahovali nite (napríklad hodvábny portrét Jacquarda bol vytvorený pomocou "programu" z 10000 kariet).

Americký inžinier **Hermann Hollerith** použil r. 1890 pri **sčítaní obyvateľov** USA pre každého občana ručne dierkovaný štítok. Otvor na určenom mieste znamenal ÁNO pri odpovedi na danú otázku a v stroji ohmataním štítkov drôtenými kefkami určoval prechod elektrického prúdu, čo sa prenášalo na panel tabelačného stroja. Stroj umožňoval triedenie dát a jednoduché výpočtové operácie. 43 strojov spracovalo údaje o 62 miliónoch ľuďoch za pár týždňov, predtým ručne 500 pracovníkov po dobu 7 rokov. Holleritova firma Tabulating Machine sa r. 1924 premenovala na International Bussiness Machines - slávne **IBM**.

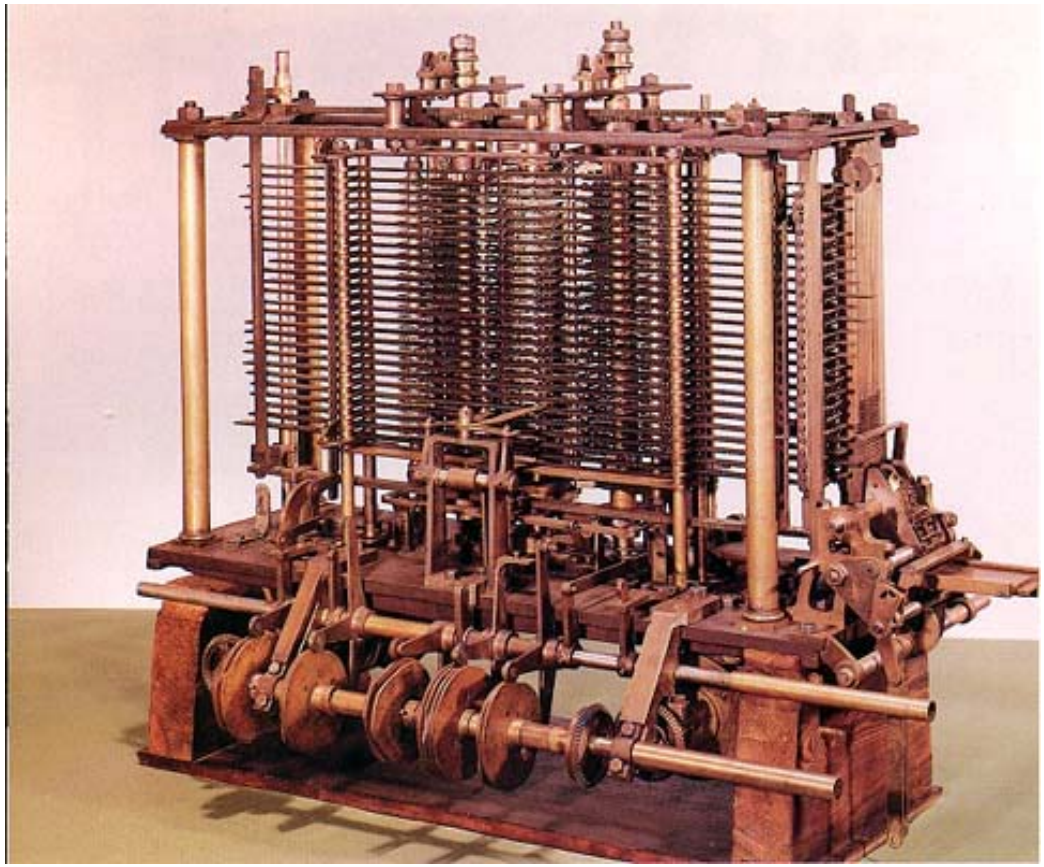


Differential engine

V roku 1830 Charles Babbage (1791-1871), anglický matematik navrhol stroj poháňaný parou na výpočet a tlač matematických tabuliek, využívajúci metódu diferencií. Táto metóda prevádzala výpočet polynómov na sčítanie. Pre technické problémy pokročilá verzia nikdy nezrealizovaná.



Babbage: Návrh stroja pre univerzálne použitie - **Analytical Engine**. Výpočet plánoval riadiť pomocou diernych štítkov, kedy jeden druh štítkov obsahoval určenie operácie a druhý adresu čísla; stroj mal mať mechanickú adresovateľnú pamäť (až 1000 čísel po 50 cifrách) a "mlynček" = centrálnu aritmetickú jednotku, v ktorej sa mali vykonávať základné operácie. Programovanie komplikovaných algoritmov. Babbage: „bude predstavovať mechanickú inteligenciu“. Cieľom bolo, aby si stroj dokázal sám flexibilne meniť svoj program v priebehu výpočtov.



Prvý programátor na svete: Babbageho spolupracovníčka, dcéra anglického básnika Byrona - grófka **Augusta Ada King of Lovelace**.

Odhalila základy programovania a význam podmieneného vetvenia programu podľa výsledku predchádzajúceho kroku a význam cyklov alebo slučiek v programovaní, napísala aj program na riešenie sústavy lineárnych rovníc a na generovanie Bernoulliho čísel. Špekulovala tiež o možnosti použiť stroj na generovanie hudobných diel pomocou zakódovania zákonov harmónie a kompozície na dierne štítky. Taktiež uvažovala o možnosti použiť analytický stroj v úlohe manipulátora s algebraickými výrazmi.

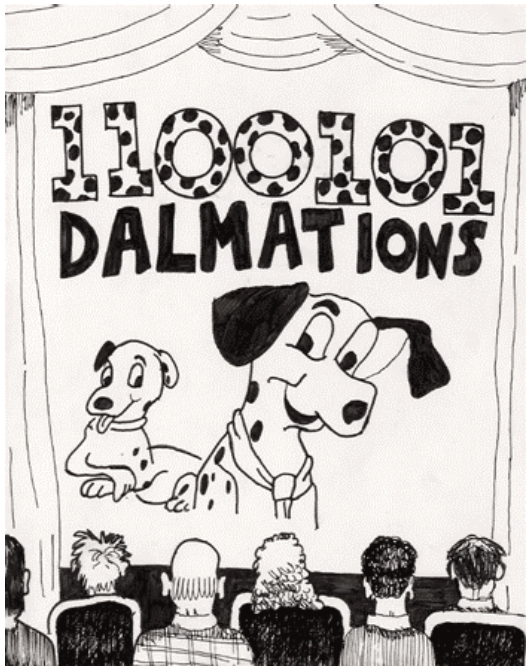
Lady Ada videla ale aj obmedzenia počítačov, v dopise svojmu známemu napísala: "Analytický stroj nemá ambície vymyslieť niečo originálne. Dokáže urobiť iba čokoľvek, o čom vieme, ako mu prikázať, aby to vykonal. Môže postupovať podľa výsledkov analýzy riešenia; nemá ale žiadnu schopnosť vymyslieť akékoľvek analytické vzťahy alebo tvrdenia."

V druhej polovici 20. storočia bol na jej počesť pomenovaný programovací jazyk ADA.

Binárna logika

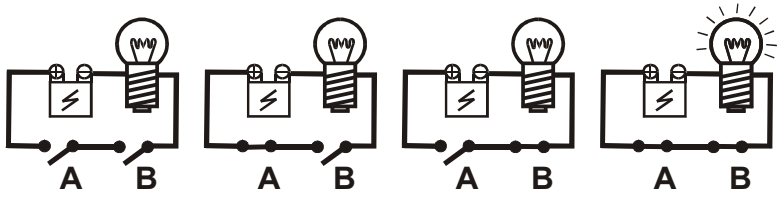
Niektoré z prvých počítačov pracovali analógovým spracovaním signálu.

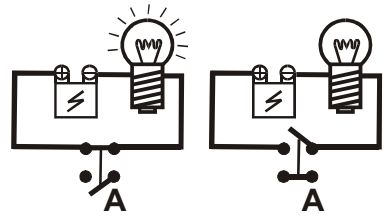
Zvíťazili ale počítače s binárnou logikou (na najnižšej úrovni iba nuly a jednotky). Základy binárnej logiky položil anglický matematik George Boole (1815-64), vymyslel systém na ohodnotenie pravdivostných hodnôt výrazov zložených z logických spojok AND, OR, NOT a logických premenných nadobúdajúcich iba dve hodnoty – 1 (PRAVDA) a 0 (NEPRAVDA).

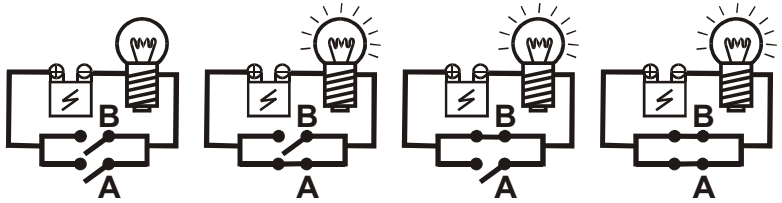


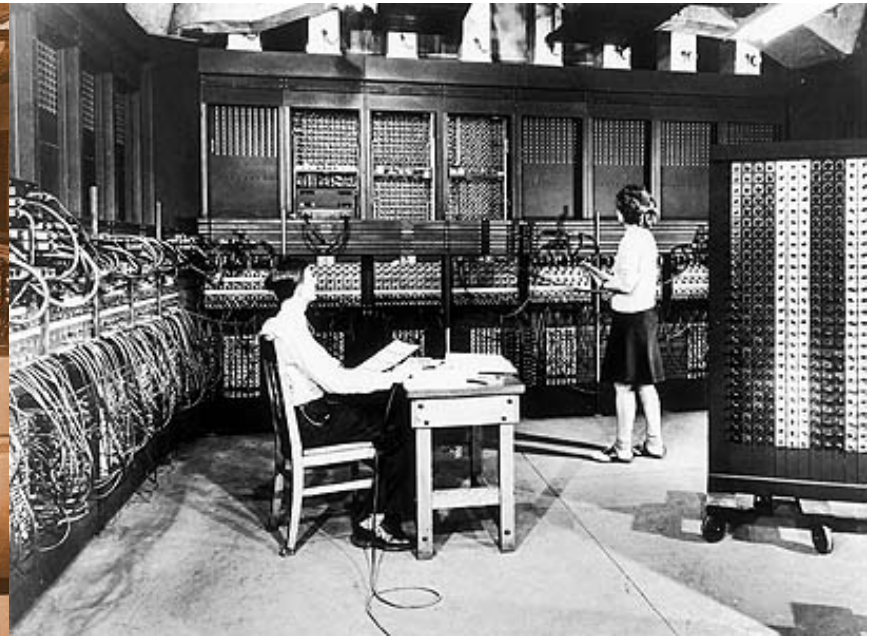
Boolovská algebra našla použitie pri návrhu technologickej súčasti moderných počítačoch, a to v binárnych klopných obvodoch. Pomocou elektronicky ovládaných hradiel sú reprezentované čísla v binárnej reprezentácii, robia sa výpočty, od sčítania a násobenia až po zložité funkcie. Vhodnou kombináciou klopných obvodov vznikajú programy, ktoré sú schopné počítať pravdivostné hodnoty zložitých funkcií. Manipulácia s binárnymi číslami je ľahko realizovateľná, a rýchla.

Prvé boli elektromechanické spínacie relé, pozostávajúce z dvoch kovových kontaktov spojených kovovým jazýčkom. Spínače môžu byť ľahko použité napríklad v AND a OR obvodoch, kedy "otvorený" (vypnutý) spínač predstavuje hodnotu NEPRAVDA (FALSE alebo nulu v binárnom kódovaní, neprechádza prúd) a "uzatvorený" (zapnutý) spojený spínač hodnotu PRAVDA (TRUE alebo jednotku v binárnom kódovaní, prechádza prúd). Obvod NOT mení pozíciu "otvorený" alebo "uzatvorený" na opačnú. Ľubovoľný výraz Boolovej algebry sa dá previesť na usporiadanie spínačov, pri pravdivosti výrazu bude obvodom prechádzať prúd. Pre použitie premennej súčasne v niekoľko obvodoch by ale boli potrebné "vetviace" spínače.

AND 

NOT 

OR 

John Prosper Eckert a John Mauchly r. 1943: vláda U.S.A. financuje **ENIAC** Electronic Numerical Integrator and Calculator, postavený z elektroniek, asi prvý mnohoúčelový elektronický počítač, využívaný na výpočty spojené s výrobou atómovej bomby. V r. 1944 - 150 m^2, 30 ton, 100 000 súčiastok, chladený vrtuľami dvoch leteckých motorov. Dokázal násobiť dve čísla za 0,003 sekundy. Pracoval paralelne, ale čísla ale v desatinnej sústave namiesto binárnej. Vyžadoval po každom výpočte hodiny prepínania drôtov a nulovania.

ENIAC plánoval John von Neumann, svetoznámy návrhom počítačovej architektúry. Pamäť je rozdelená na rovnako veľké bunky, ktoré sú očíslované, cez číslo (adresu) bunky sa dá prečítať alebo meniť jej obsah. Program je uložený do buniek idúcich za sebou, nasledujúca inštrukcia sa zoberie vždy z nasledujúcej bunky s adresou o jednotku vyššou, pokiaľ nepredchádzala inštrukcia skoku. Program dokáže upravovať sám seba v priebehu výpočtu. Všetky dáta aj program sú binárne kódované.

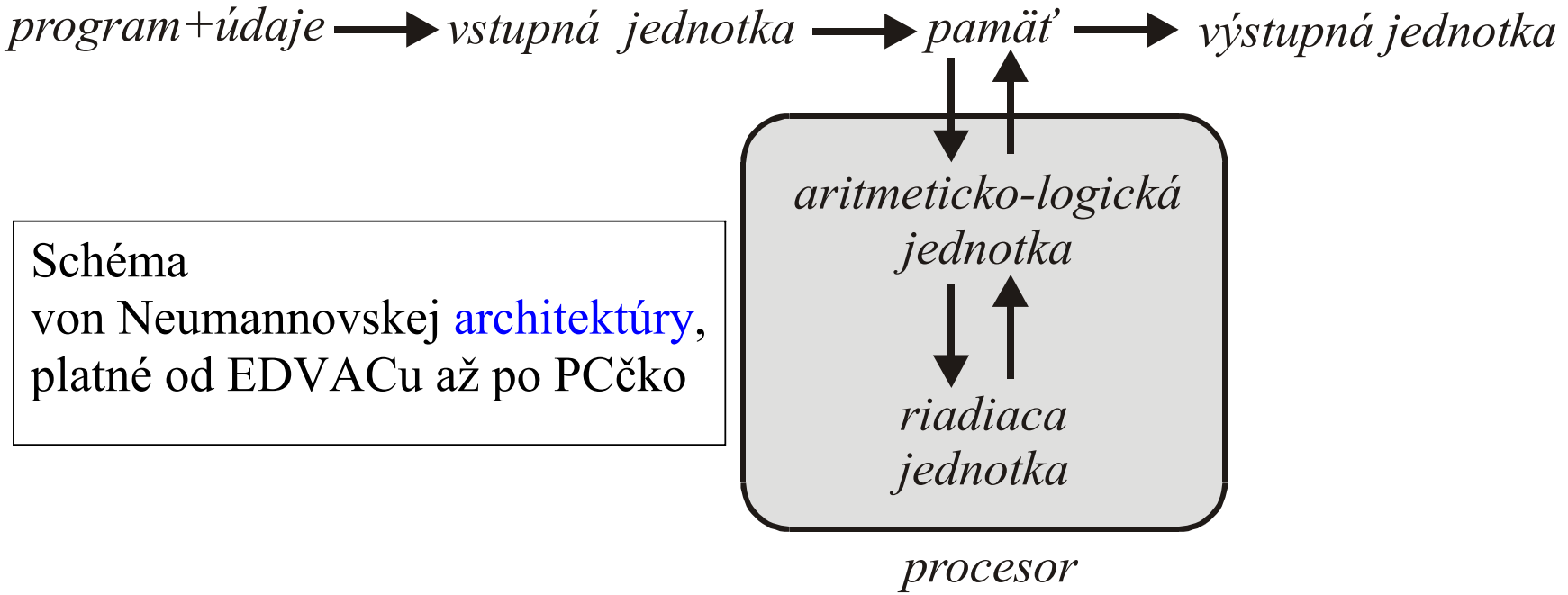


Schéma
von Neumannovskej architektúry,
platné od EDVACu až po PCčko

Vývoj softvéru

Operačný systém – určuje, čo sa robí, keď stlačíme klávesu klávesnice.
UNIX (spolu s jeho verziou Linux pre osobné počítače)
Microsoft Windows

Programovacie jazyky:

- do 50-tych rokov 20. storočia program aj dáta ako sadu núl a jedničiek.
- r. 1950 Maurice Wilkes: **assembler**, jednoduché skratky príkazov v angličtine (ako "pričítaj" alebo "ulož do pamäti")
- **jazyk vyššej úrovne**, jeden príkaz na niekoľko strojových príkazov riadiacej a aritmeticko-logickej jednotke počítača pomocou kompilátora

Prvý kompilátor r. 1952: námorná admirálka USA pani Grace Murray Hopper (Aj "debugovanie" počítača: mora v reléovom prepínači Mark I).

FORTRAN vedecko-tech. aplikácie: (FORmula TRANslation)
COBOL, obchod a riadenie podniku.
LISP (LISt Processing, inak Lots of Idiotic, Silly Parentheses) umelá inteligencia, MIT
PROLOG Európe v viac používaný jazyk logického programovania.
PASCAL Niklaus Wirth, na výuku
BASIC jednoduchý, nekompatibilný
C a C++ vedecko-technické výpočty
JAVA, C# v internetu

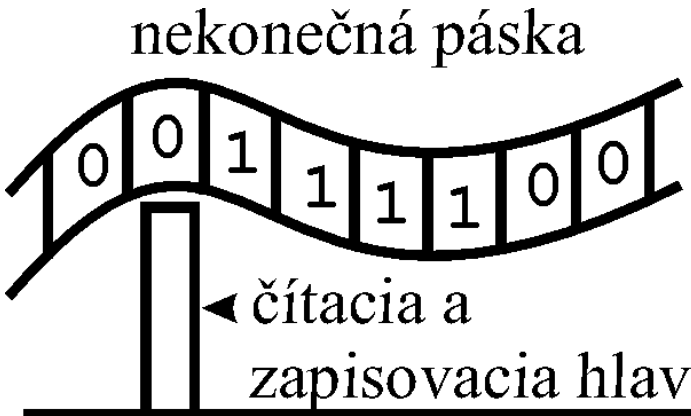
Teória algoritmov

Chyba architektúry procesoru?

Chyba algoritmu?!? Formálna analýza, alebo porovnanie výpočtov.

Formálne je algoritmus tzv. vypočitateľná funkcia. Model:

Turingov stroj (Alan Mathison Turing, britský matematik, 1912-54)



Prepisovacie pravidlá:

vnútorný stav	číta z pásky	nový vnút. stav	zapisuje na pásku	akcia
0	0	0	0	vpravo
0	1	1	1	vpravo
1	0	0	1	stop
1	1	1	1	vpravo

Turingov stroj

Je to automat s konečným počtom stavov, o ktorom sa dajú dokazovať matematické vety.

Turingov stroj, ktorý pričíta jednotku k unárnemu číslu (prirodzenému číslu *n* odpovedá unárne číslo rovné *n* jedničkám, napr. 2=11, 5=11111).

Algoritmy a počítanie

Počítač transformuje vstupné údaje na výstupné údaje. Potrebuje

(1) *reprezentáciu*, ktorá určuje symbolické kódovanie použitej informácie (tak vstupnej, ako aj výstupnej, a samozrejmé aj medzivýsledkov), napr. vo forme čísel alebo mien, a

(2) *transformáciu* (predpis, algoritmus, program), ktorá je použitá na výpočet požadovaného výsledku, t.j. transformáciu vstupných údajov na výstupné údaje.

Univerzálnosť algoritmu znamená, že je použiteľný pre riešenie veľkej skupiny úloh toho istého typu, líšiacich sa vstupnými údajmi (tak napr. algoritmus pre hľadanie koreňov kvadratickej rovnice je použiteľný pre každú kvadratickú rovnicu). Algoritmus je popis transformácie, ktorý je určený nielen pre pochopenie človekom, ale slúži aj ako podklad pre napísanie programu, a umožňuje vznik technickej pomôcky, ktorá na seba preberá časť procesu, ako sú prístroje a tabuľky. Počítač bez algoritmov = komplikovaný kus kovu.

Algoritmus je predpis, metóda, alebo technika, ktorý špecifikuje postup úkonov potrebných na dosiahnutie riešenia nejakej úlohy (ako aj napr. usporiadanie zoznamu mien podľa abecedy alebo recept na bábovku).

Pôvod slova od arabského matematika al-Chwárizmí (v angličtine al-Khowarizmi) z ôsmeho storočia n. l., vo svojej knihe „*Al-jabr wa´l muqabala*“ (podľa ktorej vznikol názov Algebra) používal desiatkovú sústavu a nulu (čo prevzal z indickej matematiky).

Algoritmus v informatike je jednoznačná, presná a konečná postupnosť operácií, ktoré sú aplikovateľné na množinu objektov alebo symbolov (čísiel, šachových figúrok, ingredencií na bábovku). Počiatočný stav týchto objektov je vstupom, ich koncový stav je výstupom. Počet operácií, vstupy a výstupy sú konečné (aj keď bežne počítame napr. s iracionálnym číslom π, vždy jeho číselnú reprezentáciu obmedzíme pri numerických výpočtoch na konečnú presnosť, napr. π=3.14).

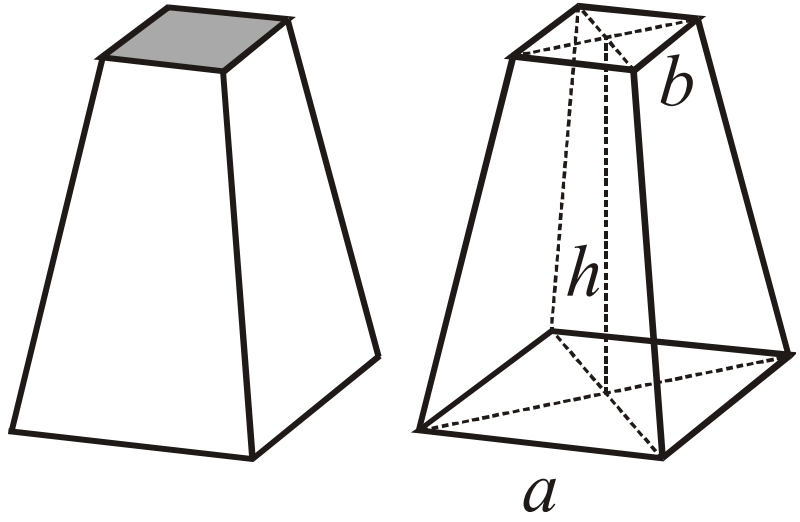
Algoritmus berie vstupné hodnoty problému (funkcie) a mapuje ich na výstup. Jeden problém môže byť riešený veľa algoritmami.

Jednoznačnosť znamená, že každý krok algoritmu musí byť presne definovaný. Nesmie dovoľovať viac výkladov, jednoznačne je určený krok za ním nasledujúci.
Rezultatívnosť znamená, že algoritmus vždy musí po konečnom počte krokov dojsť k nejakému riešeniu.
Ďalšou základnou požiadavkou na algoritmus je **správnosť**, čo znamená, že odpoveď algoritmu je pre každý vstup korektná.
A samozrejme, algoritmus by mal byť efektívny. **Efektívnosť** - potrebný výpočtový čas a kapacita pamäti nutná na výpočet - často protichodné, uvažuje sa iba výpočtový čas.

Prvé algoritmy

V starom Egypte, napr. návod na výpočet objemu zrezaného ihlanu (ihlan so štvorcovou podstavou, ktorého vrchol je zrezaný rovinou rovnobežnou s podstavou) je podľa tzv. Moskovského papyrusu (Egypt, 1850 p. n. l.) nasledujúci: Je daná orezaná pyramída, ktorej výška je 6, strana podstavy je 4 a strana hornej základne je 2. Vypočítaj objem tejto pyramídy:

1. Umocni číslo 4 na druhú, dostaneš 16.
2. Číslo 4 zdvojnásob, dostaneš 8.
3. Umocni na druhou číslo 2, dostaneš 4.
4. Tieto čísla 16, 8 a 4 sčítaj, dostaneš 28.
5. Urči tretinu z čísla 6, dostaneš 2.
6. Zdvojnásob číslo 28, dostaneš 56.
7. Celkový výsledok je 56, počítal si dobre.



Použitie symbolov až u starých Grékov u Aristotela, $V=1/3\ h\ (a^2 + ab + b^2)$.

Formalizácia konceptu algoritmu v Euklidových *Základoch* (3. st. p. n. l.),

Primitívne problémy? Úloha zadaná Euklidom – **kvadratúra** kruhu – skonštruovať pomocou kružidla a pravítka štvorec s rovnakou plochou ako má zadaný kruh - až koncom 19 storočia (23 storočí po zadaní úlohy) bolo dokázané, že nemá riešenie.

Známy Euklidov algoritmus:

výpočet maximálneho spoločného deliteľa dvoch čísel *m* a *n*:

1. Keď *m* je menšie ako *n*, vymeň ich hodnoty
2. Do *m* daj hodnotu zvyšku po delení *m*/*n* (v modernej terminológii sa to zapisuje ako *m*:=*m* modulo *n*)
3. Keď sa *m* nerovná nule, choď na krok 1 s novými hodnotami *m* a *n*
4. Vráť *n* ako výsledok

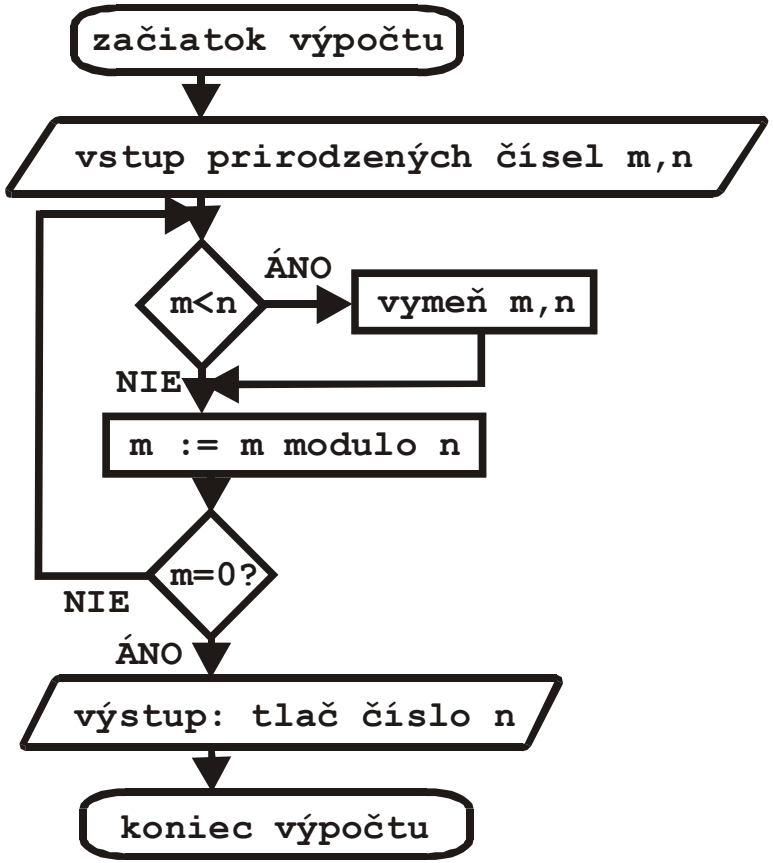
(56,21)

Ďalším známym starovekým algoritmom je **Eratosthenovo sito** (z 3 storočia p. n. l.) na nájdenie všetkých prvočísiel menších ako *n* (prvočíslo je celé číslo väčšie ako 1, ktoré je bezo zvyšku deliteľné iba sebou samým a číslom 1, v súčasnej dobe sa prvočísla často využívajú napr. pri šifrovaní správ). Výsledok algoritmu pre *n*=50 je na obr. 2.3, Eratosthenov predpis znie:

1. Zapíš do zoznamu všetky čísla od 2 do *n*
2. Nech *k*=2
3. Pre každé číslo *m* medzi *k*+1 a *n* skontroluj, či je presným násobkom *k*, keď áno, vyškrtni *m* zo zoznamu.
4. Do *k* daj najmenšie ešte nevyškrtnuté číslo zo zoznamu
5. Keď *k* je menšie ako *n*, opakuj od kroku 3
6. Akékoľvek číslo zo zoznamu, ktoré nebolo vyškrtnuté, je prvočíslo

	~~1~~	2	3	~~4~~	5	~~6~~	7	~~8~~	~~9~~	~~10~~
deliteľné	def.			2		2		2	3	2
	11	~~12~~	13	~~14~~	~~15~~	~~16~~	17	~~18~~	19	~~20~~
deliteľné		2		2	3	2		2		2
	~~21~~	~~22~~	23	~~24~~	~~25~~	~~26~~	~~27~~	~~28~~	29	~~30~~
deliteľné		2		2		2		2		2
	31	~~32~~	~~33~~	~~34~~	~~35~~	~~36~~	37	~~38~~	~~39~~	~~40~~
deliteľné		2	3	2	5	2		2	3	2
	41	~~42~~	43	~~44~~	~~45~~	~~46~~	47	~~48~~	~~49~~	~~50~~
deliteľné		2		2	3	2		2	7	2

Zápis algoritmov Vývojový diagram a pseudokód Euklidovho algoritmu na výpočet maximálneho spoločného deliteľa dvoch čísel *m* a *n*.




```
Vstup dvoch celých čísel: m, n
  rob
    keď m < n vymeň m,n
    m := m modulo n
  dokial n≠0
  vytlač n
```

Základné prvky: vstup, výstup, podmienka, skok, slučka

Zápis algoritmu pomocou pseudokódu

V 60. rokoch 20 storočia jazyk **Algol**, v 70. rokoch **Pascal** – v praxi zabudnuté (okrem Delphi), v teórii vládnu, na zápis algoritmov **Pseudopascal**

Premenná meno konkrétneho pamäťového miesta s údajmi, obsah sa môže meniť.
Typ premennej: číselná, logická, textová
Označenie postupnosťou písmen, číslic a podčiarnikov, napr. prva_faza
Jednoduchá premenná – x, y
Štrukturovaná premenná – napr. matice, pole X=X[1],X[2],X[3],…
Priradenie
x:=konštanta
x:=1
x:=y
vstup(x) do premennej x užívateľ uloží hodnotu, napr. z klávesnice
výstup(x) obsah premennej sa vypíše, napr. na obrazovku
výstup(„Ahoj“) vypíše Ahoj

Kľúčové slová tučne

začiatok	**begin**
koniec	**end**
ak	**if**
potom	**then**
inak	**else**
pokiaľ	**while until dokiaľ je/dokiaľ nie je** splnená podmienka opakuj
opakuj	**repeat, do**
pre	**for**
od	
do	**to**

Príkaz ukončujeme bodkočiarkou

Sekvencia (blok) je postupnosť akcií uzatvorená slovami začiatok, koniec

začiatok Determinant … (pomenovanie, ako napr. Determinant, nemusí byť)
    príkaz1;
    príkaz2; (príkazy vnútri môžu obsahovať aj vetvenia a cykly)
    príkaz3;
koniec Determinant

Vetvenie rozhodujeme sa podľa splnenia, či nesplnenia podmienky, ktorá nadobúda hodnoty TRUE alebo FALSE

Úplné vetvenie

ak podmienka **potom**
    príkaz1;
inak
    príkaz2;

Neúplné vetvenie

ak podmienka **potom**
    príkaz1;

Cyklus je akcia, v ktorej po splnení podmienky je opakovane vykonávaná rovnaká akcia volaná telo cyklu

pokiaľ podmienka **opakuj**

    príkaz1;

Cyklus ktorý sa vždy vykoná aspoň raz a splnenie podmienky pre opakovanie tela cyklu je vykonávané na konci

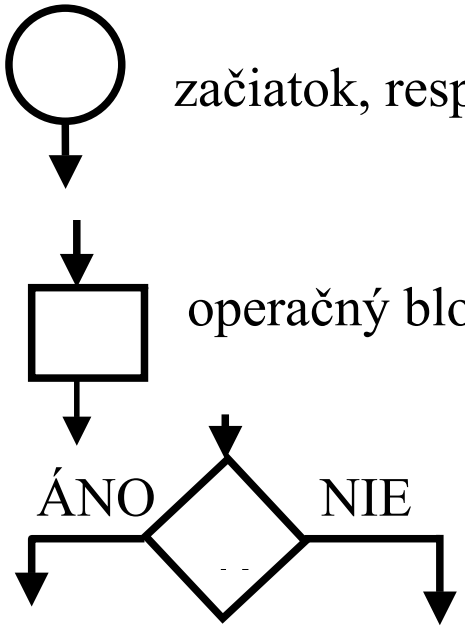
opakuj

    príkaz1;
    príkaz2;

pokiaľ podmienka

Jazyk vývojových diagramov

Popisuje tok riadenia od jedného k ďalšiemu kroku algoritmu (bežne odzhora dole) pomocov orientovaných hrán (šípok) spájajúcich činnosti opísané v blokoch. V praxi sa už veľmi nepoužíva, ale pre vizualizáciu jednoduchých algoritmov je užitočný

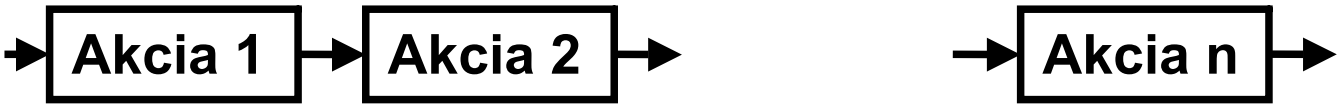


začiatok, resp. koniec algoritmu

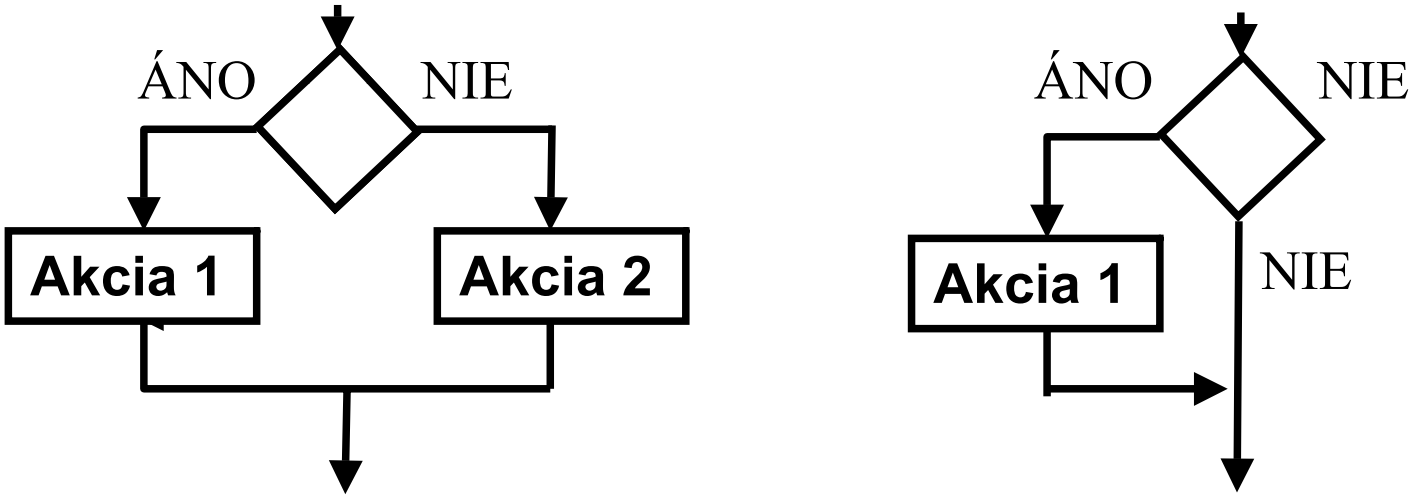
operačný blok, obsahuje akcie

rozhodovací blok, splnenie/nesplnenie podmienky B
Určí nasledujúci krok riadenia

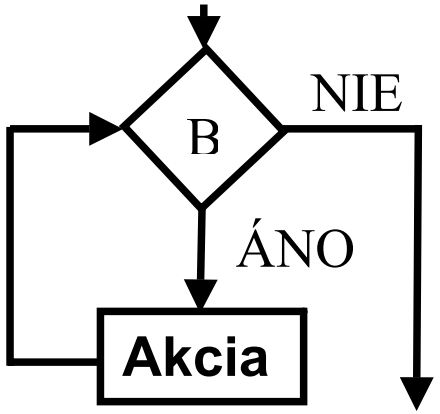
Sekvencia



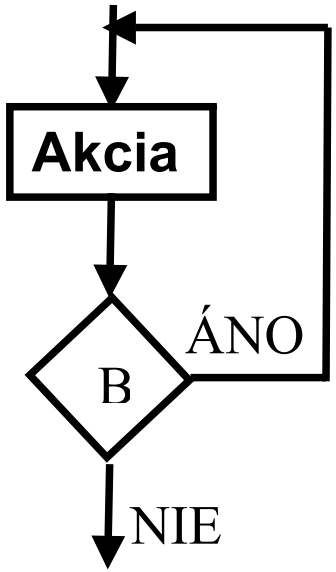
Vetvenie



Cyklus s testom na začiatku



Cyklus s testom na konci

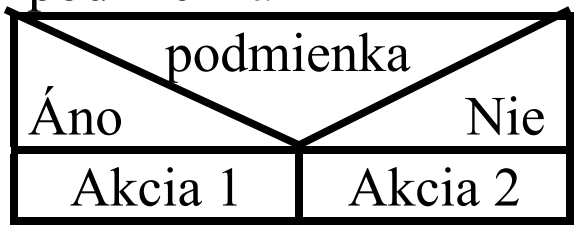


Nassi-Schneidermanové diagramy
=jazyk štruktúrovaných vývojových diagramov

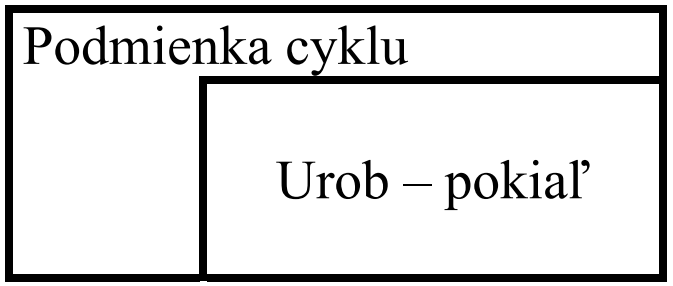
operačný blok

sekvenčný blok

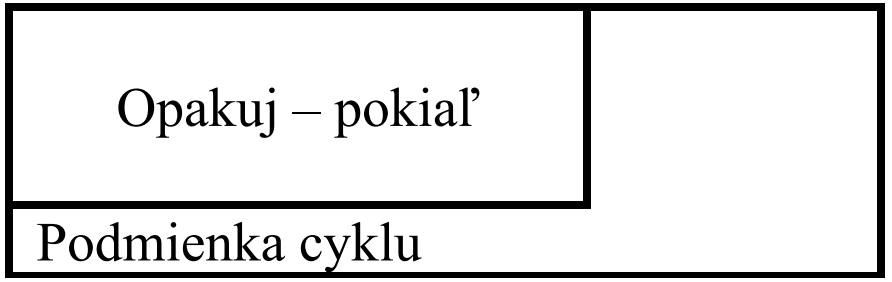
podmienka



Cyklus s testom na začiatku



Cyklus s testom na konci



Efektívnosť algoritmu často rozhoduje o prijateľnosti alebo nemožnosti riešenia problému danej veľkosti.

Druh analýzy:

Najhorší prípad: $\Theta(n)$ = maximálny čas algoritmu pre veľkosť vstupu n.

Priemerný čas: $\Theta(n)$ = očakávaný čas algoritmu pre náhodne vybraný vstup veľkosti n, treba štatistické spracovanie.

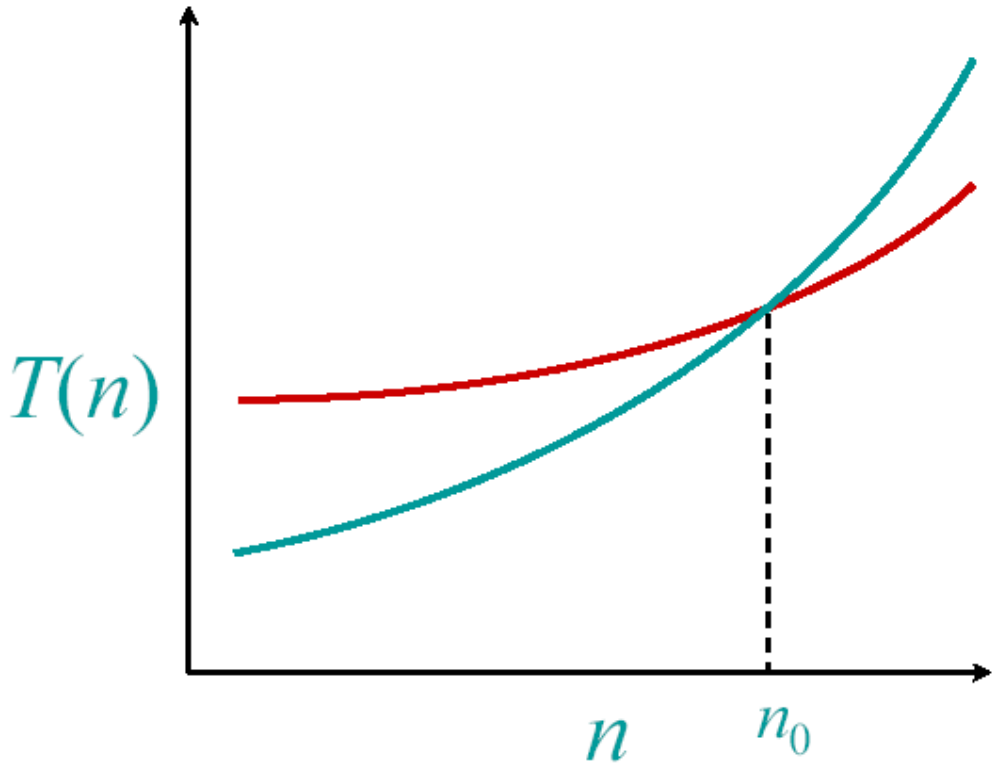
Najlepší prípad: (podraz) Aby ste dokázali pri diplomke, že Váš algoritmus je lepší ako iné, vyberiete príklad, na ktorom funguje najlepšie.

Čas - strojový – závisí na počítači BIG IDEA:

Ignoruje konštanty závislé na typu počítače, pozerajte sa iba na rast $\Theta(n)$ pre $n \to \infty$.

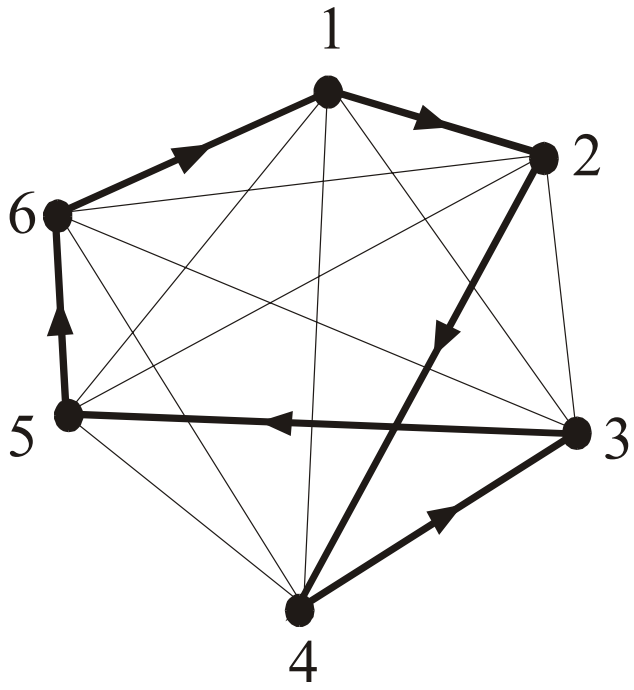
Odstránte členy polynomu nižšieho rádu; ignorujte konštanty u najvyššieho rádu

Pr.: $3n^3 + 90n^2 - 5n + 6046 = \Theta(n^3)$



Výber vhodného algoritmu závisí na veľkosti problému, pre dostatočne veľké n je $\Theta(n^2)$ algoritmus *vždy rýchlejší* akob Θ (n^3) algoritmus

Algoritmy sa z hľadiska **efektívnosti** delia na **NP-úplné** a **polynomiálne**. Jedným z najznámejších NP- úplných problémov je napríklad problém obchodného cestujúceho, kde hypotetický obchodný cestujúci má navštíviť *n* miest (každé len raz, pričom na záver sa vracia do východzieho mesta), pričom si musí navrhnúť takú cestu, aby mala minimálnu vzdialenosť. Počet všetkých možných usporiadaní postupností *n* miest s daným prvým mestom je (*n*-1)!



P=(1,2,4,3,5,6)

Cyklická cesta (tzv. hamiltonovský cyklus) na úplnom grafe, ktorý obsahuje 6 vrcholov (vrcholy - mestá, hrany - cesty ohodnotené vzdialenosťou dvojice miest; v grafe záleží iba na tom, či sú mestá prepojené, nie na tom, ako sa cesta krúti). Cestu (nie najkratšiu) reprezentuje permutácia P=(1,2,4,3,5,6).

Pre 1000 miest nenájdeme najlepšie riešenie, ani keby sme superpočítač nechali bežať miliardy rokov.
Povedzme, že máme iba 100 miest a pridáme do rozpisu dve mestá, počet možností sa zvýši 100×101 krát ≅ 10000.

Polynomiálny algoritmus znamená, že čas na výpočet sa dá vyjadriť ako polynóm premennej danej veľkosťou problému. Napríklad aj ten najjednoduchší algoritmus pre zoradenie n čísel sa dá vyjadriť polynómom čas_na_výpočet=$a\ n^2 + b\ n + c$ a teda sa o ňom hovorí, že má zložitosť $O(n^2)$, pre veľmi veľké n má význam iba najväčšia mocnina a koeficient a závisí na implementácii.

U problému obchodného cestujúceho s algoritmom zložitosti $O(2^n)$ by to bolo exponenciálne zvýšenie 2^n krát (pri prechodu zo 100 na 200 miest sa počet operácií zvýši viac ako 1000000000000000000000000000000 krát).
U zoradenia čísel by zvýšenie času nebolo viac ako štyrikrát.

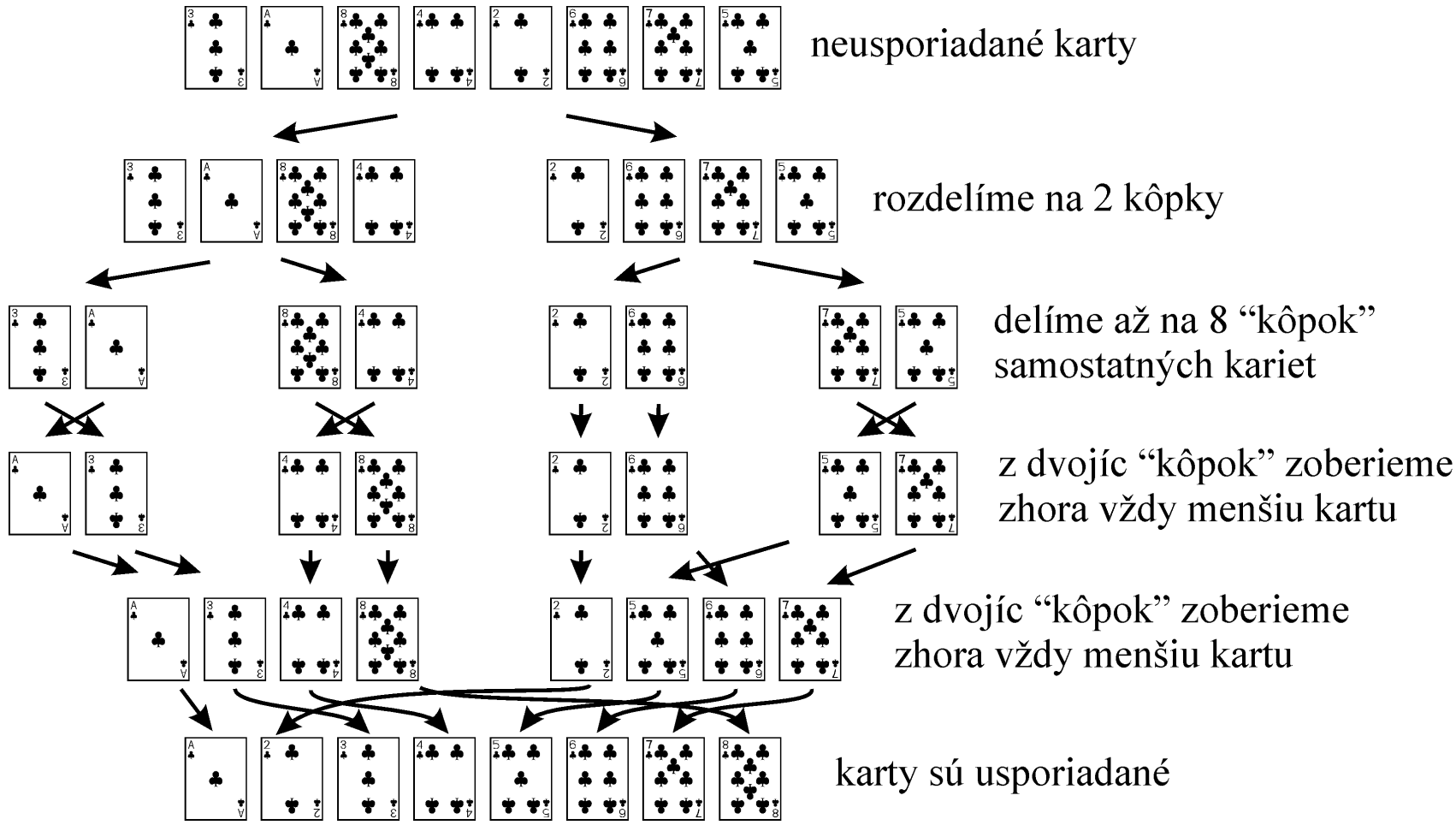
Niektoré všeobecné algoritmy
Vyhľadávanie:
sekvenčné v telefónnom zozname meno k číslu
binárne v telefónnom zozname číslo k menu, zoznam si vždy rozdelíme napol, počet potrebných porovnaní $\log_2$(počet mien v zoznamu).

Keď máme vyhľadať 100 ľudí v zozname občanov Slovenska o 5000000 menách, a porovnanie trvá povedzme 0,000001 s:
sekvenčné vyhľadávanie päť minút
binárne vyhľadávanie 0,002s (ale zoznam musí byť utriedený).

Triedenie

priamym výberom najpomalšie ale najjednoduchšie na pochopenie

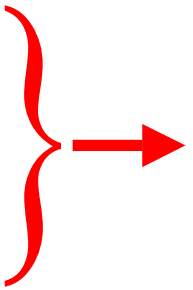
Napr. karty v priebehu usporiadania delíme na dve kôpky, jednu už utriedenú a druhú ešte neutriedenú. Na začiatku samozrejme máme iba neutriedenú kôpku. V každom kroku výpočtu vždy zoberieme kartu s najmenšou hodnotou z neutriedenej kôpky a dáme ju na utriedenú kôpku, dokiaľ nie sú všetky karty utriedené. Najmenšiu kartu z neutriedenej kôpky vyberáme tak, že zoberieme prvú kartu zhora a postupne ju porovnávame zo zvyšnými kartami. Keď je porovnávaná karta z kôpky menšia, vymeníme karty a pokračujeme v porovnávaní kariet až do konca kôpky. Postupne tak pre n kariet vykonávame n-1 +n-2 ...+1=n(n-1)/2 porovnávaní.



triedenie zlučovaním podľa zásady „**rozdeľ a panuj**“, počet porovnaní $n \log n$. Pre 5000000 mien priamym výberom pol roka, zlučovaním 0,5 min

Počítačový program je konkrétna reprezentácia algoritmu v nejakom programovacom jazyku.

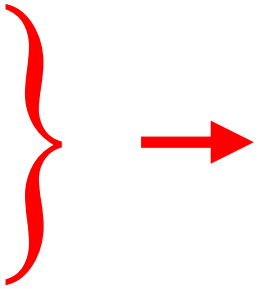
a. Správne výsledky
b. Efektívne (minimum času, niekedy aj pamäti)
   efektivita dosiahnutá použitím najvhodnejšej štruktúry a algoritmu, nie obratným kódovaním alebo asemblerom



Analýza algoritmov a dátových štruktúr

c. Uživateľsky príjemné → **Psychológia, kognitívna veda**

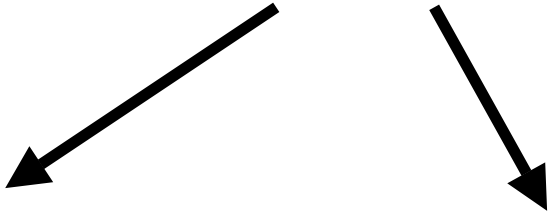
d. Ľahké na čítanie a pochopenie
e. Ľahko sa v nich hľadajú chyby
f. Ľahko sa modifikujú
g. Robustné
h. Rozširiteľné



Softvérové inženierstvo - tvorba veľkých softvérových systémov

i. Spoľahlivé ⇒ modulárne a štrukturované namiesto "špagetového" kódu

Je programovanie umenie?



Áno

vymýšľanie principiálne nových algoritmov – asi 0,001% práce programátorov, a to ešte len tých najlepších

Nie, je to predikovateľná činnosť, kde sa dá plánovať časový harmonogram a robiť podľa predpisov a vzorov (softvérové inžinierstvo)

Understanding Computer Technology