Alternatívne výpočtové modely

Motivácia

existencia veľkej triedy prakticky neriešiteľných (*ale rozhodnuteľných*) problémov, ktoré potrebujeme prakticky riešiť!

Idea

využiť principiálne iné spôsoby počítania

- paralelné počítanie
- súbežnosť
- kvantové počítanie
- molekulárne počítanie
- náhodnosť

pomôže to ???

Alternatívne výpočtové modely

Princíp paralelizmu

Praktický príklad

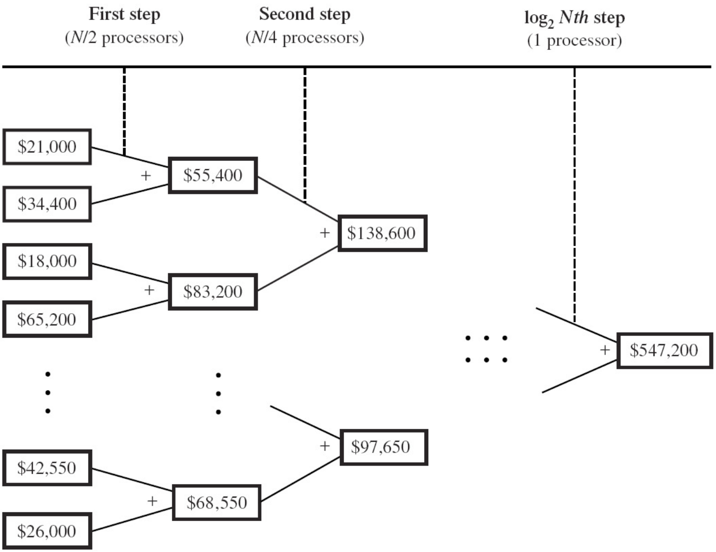
- Varianta A veža o základe 1 m $\times$ 10 m a výšky 1 m; 1 murár vs 10 murárov
- Varianta B veža o základe 1 m $\times$ 1 m a výšky 10 m; 1 murár vs 10 murárov

Jednoduchý program

- Varianta A $X \leftarrow 3;\ Y \leftarrow 4$
- Varianta B $X \leftarrow 3;\ Y \leftarrow X$

paralelizovateľné problémy vs vnútorne sekvenčné problémy

Paralelné sčitovanie



Paralelné sčitovanie

pre sčítanie 1 000 čísel potrebujeme

v 1. kroku 500 procesorov

v 2. kroku 250 procesorov

v 3. kroku 125 procesorov

... ...

pri vhodne zvolenej dátovej štruktúre a organizácii komunikácie mezdi procesormi stačí na realizáciu celého výpočtu práve 500 procesorov

pre sčítanie N čísel potrebujeme $N/2$ procesorov a počet (paralelných) výpočtových krokov je $\mathcal{O}(\log N)$

Paralelizmus - počet procesorov

- počet procesorov potrebných k realizácii paralelného výpočtu ako funkcia veľkosti vstupnej inštancie
  ($N/2$ *pre sčítanie* N *čísel*)
- je to realistické?

indikátor, aké veľké vstupy môžeme riešiť s počtom procesorov, ktoré máme k dispozícii
(*viz analógia s časovou a priestorovou zložitosťou*)

ak počet procesorov, ktoré máme k dispozícii, je menší, môžeme kombinovať paralelný a sekvenčný prístup

Paralelené triedenie

sekvenčné triedenie zoznamu L spájaním (*mergesort*)

procedúra **sort**-L
(1) ak L má len 1 prvok, je utriedený
(2) inak
    (2.1) rozdeľ L na dve polovičky L_1 a L_2
    (2.2) **sort**-L_1
    (2.3) **sort**-L_2
    (2.4) spoj dva utriedené zoznamy do jedného utriedeného zoznamu

počet vykonaných porovnaní je $\mathcal{O}(N \log N)$

Paralelné triedenie

paralelné triedenie zoznamu L spájaním

procedúra **parallel-sort**-L

(1) ak L má len 1 prvok, je utriedený

(2) inak

- (2.1) rozdeľ L na dve polovičky L_1 a L_2
- (2.2) súbežne volaj **parallel-sort**-L_1 a **parallel-sort**-L_2
- (2.3) spoj dva utriedené zoznamy do jedného utriedeného zoznamu

počet (paralelných) porovnaní je (*predpokladáme, že N je mocninou 2*)

- v 1. kroku N postupností dĺžky 1; $N/2$ procesorov;
  spojenie dvoch postupností $=$ 1 porovnanie
- v 2. kroku $N/2$ postupností dĺžky 2; $N/4$ procesorov;
  spojenie dvoch postupností $=$ 3 porovnania
- v 3. kroku $N/4$ postupností dĺžky 4; $N/8$ procesorov;
  spojenie dvoch postupností $=$ 7 porovnaní

spolu $1 + 3 + 7 + 15 + \cdots + (N-1) \leq 2N$ porovnaní

Paralelizmus - čas × priestor

konvencia: v kontexte paralelných výpočtov sa pod priestorovou zložitosťou rozumie počet procesorov potrebných k realizácii výpočtu

časová a priestorová zložitosť paralelných výpočtov sú spolu tesne zviazané

zníženie jednej obvykle znamená zvýšenie druhej a naopak

Paralelizmus - čas × priestor

	Name	Size (no. of processors)	Time (worst case)	Product (time × size)
SEQUENTIAL	**Bubblesort**	1	$O(N^2)$	$O(N^2)$
	Mergesort	1	$O(N \times \log N)$	$O(N \times \log N)$ (optimal)
PARALLEL	**Parallelized mergesort**	$O(N)$	$O(N)$	$O(N^2)$
	Odd-even sorting network	$O(N \times (\log N)^2)$	$O((\log N)^2)$	$O(N \times (\log N)^4)$
	“Optimal” sorting network	$O(N)$	$O(\log N)$	$O(N \times \log N)$ (optimal)

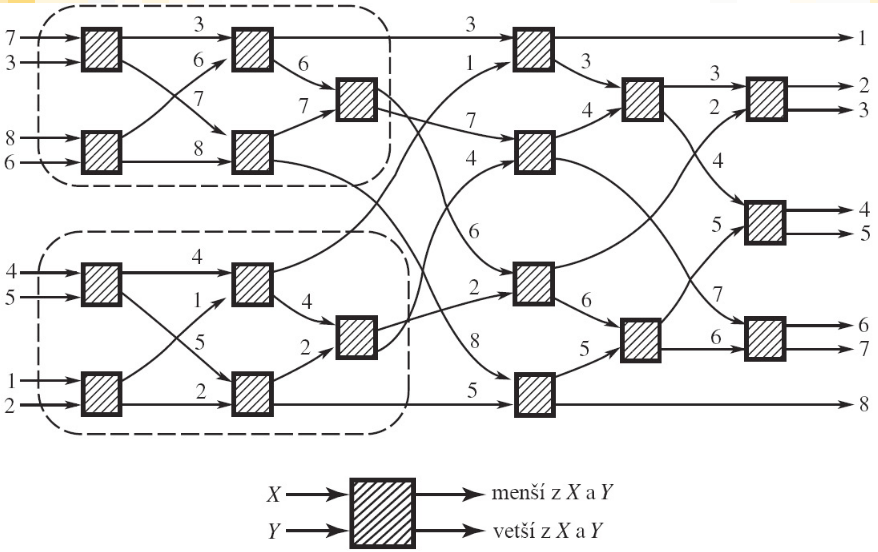
Paralelné siete

Spôsob komunikácie medzi procesormi

zdieľaná pamäť súčasný zápis resp. čítanie z pamäťového miesta ???
v prípade súčasného zápisu nutnosť stratégie riešenia konfliktov

sieť s fixovaným prepojením každý procesor je prepojený (môže komunikovať) len s ohraničením počtom *susediacich* procesorov;
často ako špecializovaný hardware

Triediaca sieť



Môže paralelizmus riešiť neriešiteľné?

Fakt

Každý paralelný algoritmus sa dá transformovať na sekvenčný algoritmus.

(*každý paralelný krok nahradíme postupnosťou sekvenčných krokov; každý sekvenčný krok vykoná prácu jedného procesoru*)

Dôsledok

Neexistuej paralelný algoritmus pre nerozhodnuteľný problém.

(*CT hypotéza sa vzťahuje aj na paralelné výpočtové modely*)

Paralelizmu a prakticky neriešiteľné problémy

- existujú problémy, ktoré sú sekvenčne prakticky neriešiteľná a pritom sú prakticky riešiteľné paralelnými algoritmami?
- špecifikácia pojmu „efektívneho“ paralelného algoritmu ???

Paralelizmu a prakticky neriešiteľné problémy

- existujú problémy, ktoré sú sekvenčne prakticky neriešiteľná a pritom sú prakticky riešiteľné paralelnými algoritmami?
- špecifikácia pojmu „efektívneho“ paralelného algoritmu ???
- pozorovanie: pre problém z triedy NP máme nedeterministický algoritmus polynomiálnej časovej zložitosti;
  ak nedeterministický výber nahradíme paralelizmom, tak okamžite dostávama polynomiálne časovo ohraničený paralelný algoritmus pre tento problém
- je to prijateľné riešenie?

Paralelizmu a prakticky neriešiteľné problémy

- existujú problémy, ktoré sú sekvenčne prakticky neriešiteľná a pritom sú prakticky riešiteľné paralelnými algoritmami?
- špecifikácia pojmu „efektívneho“ paralelného algoritmu ???
- pozorovanie: pre problém z triedy NP máme nedeterministický algoritmus polynomiálnej časovej zložitosti;
  ak nedeterministický výber nahradíme paralelizmom, tak okamžite dostávama polynomiálne časovo ohraničený paralelný algoritmus pre tento problém
- je to prijateľné riešenie?
  - exponenciálny počet procesorov

Paralelizmu a prakticky neriešiteľné problémy

- existujú problémy, ktoré sú sekvenčne prakticky neriešiteľná a pritom sú prakticky riešiteľné paralelnými algoritmami?
- špecifikácia pojmu „efektívneho“ paralelného algoritmu ???
- pozorovanie: pre problém z triedy NP máme nedeterministický algoritmus polynomiálnej časovej zložitosti;
  ak nedeterministický výber nahradíme paralelizmom, tak okamžite dostávama polynomiálne časovo ohraničený paralelný algoritmus pre tento problém
- je to prijateľné riešenie?
  - exponenciálny počet procesorov
  - čo ak prakticky neriešiteľný problém nepatrí do NP?

Paralelizmu a prakticky neriešiteľné problémy

- existujú problémy, ktoré sú sekvenčne prakticky neriešiteľná a pritom sú prakticky riešiteľné paralelnými algoritmami?
- špecifikácia pojmu „efektívneho“ paralelného algoritmu ???
- pozorovanie: pre problém z triedy NP máme nedeterministický algoritmus polynomiálnej časovej zložitosti;
  ak nedeterministický výber nahradíme paralelizmom, tak okamžite dostávama polynomiálne časovo ohraničený paralelný algoritmus pre tento problém
- je to prijateľné riešenie?
  - exponenciálny počet procesorov
  - čo ak prakticky neriešiteľný problém nepatrí do NP?
  - otázka praktickej implementácie paralelného algoritmu, ktorý má síce polynomiálnu zložitosť, ale potrebuje exponenciálny počet procesorov (*napr. otázka komunikácie*)

Paralelná výpočtová téza

Časť A

Všetky „rozumné“ paralelné výpočtové modely sú polynomiálne ekvivalentné.

Časť B

Paralelný čas je polynomiálne ekvivalentý sekvenčnému času.

$$\textit{Sekvenčný}\text{-PSPACE} = \textit{Paralelný}\text{-PTIME}$$

NC - Nickova trieda

- polynomiálne časovo ohraničené paralelné algoritmy nemôžeme považovať za prakticky použiteľné
- zmyslom zavedenia paralelizmu je výrazne redukovať výpočtový čas

sublineárne algoritmy

Trieda NC

Trieda problémov riešiteľných paralelnými algoritmami s polylogaritmickou časovou zložitosťou a polynomiálnym počtom procesorov.

Trieda NC je robustná

Vzťah sekvenčných a paralelných zložitostných tried

$$\mathsf{NC} \subseteq \mathsf{P} \subseteq \mathsf{NP} \subseteq \mathsf{PSPACE}$$

Otvorené problémy: o žiadnej z inklúzií nie je známe, či je ostrá, alebo predstavuje rovnosť

Predpoklady

1. existujú problémy, ktoré sú prakticky (sekvenčne) riešiteľné, ale nie sú riešiteľné veľmi rýchlo paralelne s použitím rozumne veľkého hardwaru
2. existujú problémy, ktoré sa dajú riešiť (sekvenčne) v polynomiálnom čase s využitím nedeterminizmu, ale nie bez neho
3. existujú problémy, ktoré sa dajú riešiť v rozmunom (tj. polynomiálnom) sekvenčnom priestore (tj. aj v rozumnom paralelnom čase), ale nie sú riešiteľné v rozumnom sekvenčnom čase bez využitia nedeterminizmu.

Vzťah sekvenčných a paralelných zložitostných tried

$$\mathsf{NC} \subseteq \mathsf{P} \subseteq \mathsf{NP} \subseteq \mathsf{PSPACE}$$

Príklady problémov

NC	sčítať N čísel
	rozhodnúť, či v grafe existuje cesta z vrcholu s do vrcholu
P \ NC	rozhodnúť, či c je najväčším spoločným deliteľom čísel a,
NP \ P	problém Hamiltonovského cyklu; splniteľnosť logickej form
PSPACE \ PN	slovný futbal

Alternatívne výpočtové modely

Súbežnosť

situácie, keď paralelizmus nevyužívame k tomu, aby sme zefektívnili výpočty, ale keď paralelizmus vzniká v reálnych aplikáciach

reaktívne a zapúzdrené systémy

Úloha

navrhnúť komunikačné protokoly pre komunikujúce objekty tak, aby spĺňali požadované vlastnosti

Špecifikum

úlohou systémov nie je nájsť konkrétne riešenie, ale počítať (byť funkčný) „donekonečna“

Problém hotelovej sprchy

- na poschodí je len jedna sprcha, na veľmi studenej chodbe
- každý hosť sa chce osprchovať, ale nemôže čakať na voľnú sprchu na chodbe
- ak by z času na čas kontroloval, či je sprcha voľná, môže nastať situácia, že sa nikdy neosprchuje

možné riešenie

- tabuľa pri vstupe do sprchy
- hosť pri odchode zo sprchy zmaže z tabule číslo svojej izby a napíše na ňu číslo nasledujúcej izby (v nejakom fixovanom poradí)
- každý hosť z času na čas kontroluje, či je na tabuli napísané číslo jeho izby a ak áno, osprchuje sa

je to dobré riešenie?

(*prázdna izba, práve jedno sprchovanie v cykle, . . .*)

Distribuované súbežné systémy

- (súbežné) komponenty systému sú fyzicky oddelené
- komponenty vzájomne komunikujú
- typicky od systémov nepožadujeme vstupno - výstupné chovanie, ale kontinuálnu (neprerušenú, neukončenú) funkcionalitu
- dôležité je chovanie systému v čase
- okrem požiadavkov na jednotlivé komponenty, kladieme na systém globálne obmedzujúce podmienky
  - zdieľanie spločných zdrojov
  - prevencia uviaznutia (vzájomné čakanie, *deadlock*)
  - prevencia starnutia procesov (*starvation*)

(algoritmické) protokoly

Problém vzájomného vylúčenia

zobecnenie problému hotelovej sprchy

Problém vzájomného vylúčenia: je daných N procesov, každý proces opakovane (v nekonečnom cykle) vykonáva

- súkromné aktivity (proces ich môže vykonávať nezávislé na ostatných procesoch) a
- kritické aktivity (žiadne dva procesy nemôžu súčasne vykonávať svoje kritické aktivity)

Protokol pre problém vzájomného vylúčenia

pre dva procesy P_1 a P_2

protokol využíva 3 premenné

- Z globálna premenná (*všetky procesy môžu meniť jej hodnotu*); nadobúda hodnoty 1 a 2
- X_1 lokálna premenná procesu P_1; nadobúda hodnoty *yes* a *no*
- X_2 lokálna premenná procesu P_2; nadobúda hodnoty *yes* a *no*

počiatočná hodnota premenných X_1 a X_2 je *no*, počiatočná hodnota Z je buď 1 alebo 2

Protokol pre problém vzájomného vylúčenia (pre dva procesy)

protokol pre proces P_1

(1) opakuj v nekonečnom cykle
- (1.1) vykonaj súkromné aktivity
- (1.2) $X_1 \leftarrow yes$
- (1.3) $Z \leftarrow 1$
- (1.4) čakaj kým buď
  X_2 nenadobudne hodnotu *no* alebo
  Z nenadobudne hodnotu 2
- (1.5) vykonaj kritické aktivity
- (1.6) $X_1 \leftarrow no$

protokol pre proces P_2

(1) opakuj v nekonečnom cykle
- (1.1) vykonaj súkromné aktivity
- (1.2) $X_2 \leftarrow yes$
- (1.3) $Z \leftarrow 2$
- (1.4) čakaj kým buď
  X_1 nenadobudne hodnotu *no* alebo
  Z nenadobudne hodnotu 1
- (1.5) vykonaj kritické aktivity
- (1.6) $X_2 \leftarrow no$

korektnosť protokolu	vzájomné vylúčenie	*OK*
	starnutie procesu	*OK*
	uviaznutie	*OK*

Protokol pre problém vzájomného vylúčenia (pre N procesov)

protokol pre I-ty proces P_I

(1) opakuj v nekonečnom cykle

(1.1) vykonaj súkromné aktivity

(1.2) pre každé J od 1 do $N-1$ urob

(1.2.1) $X[I] \leftarrow J$

(1.2.2) $Z[I] \leftarrow I$

(1.2.3) čakaj kým buď

$X[K] < J$ pre nejaké $K \neq I$ alebo

$Z[J] \neq I$

(1.5) vykonaj kritické aktivity

(1.6) $X[I] \leftarrow 0$

Vlastnosti distribuovaných súbežných systémov

distribuované súbežné systémy sa odlišujú od sekvenčných systémov

špecifikácia problému nemôžeme použiť špecifikáciu problému prostredníctvom množiny vstupných inštancií a fuknčnej závislosti medzi vstupom a požadovaným výstupom

korektnosť nepostačuje dôkaz konečnosti výpočtu a správnosti vstupno-výstupného vzťahu

efektívnosť nepostačuje vyjadrenie efektivity cez časovú (*počet krokov od začiatku do ukončenia*) a priestorovú zložitosť

aké vlastnosti požadujeme od súbežných procesov?

Vlastnosti živosti a bezpečnosti

Vlastnosti, ktoré požadujeme od protokolov pre súbežné procesy (najčastejšie) spadajú do dvoch kategórií: bezpečnosť a živosť.

bezpečnosť nikdy nenastanú „špatné“ udalosti resp. vždy sa stávajú len „dobré“ udalosti

živosť určité „dobré“ udalosti sa nakoniec stanú

aby sme ukázali, že protokol porušuje vlastnosť bezpečnosti, stačí ukázať konkrétnu postupnosť akcií, ktoré vedú k zakázanej udalosti

aby sme ukázali, že protokol porušuje vlastnosť živosti, musíme ukázať existenciu nekonečnej postupnosti akcií, ktoré nevedú k požadovanej dobrej udalosti

Overovanie vlastností živosti a bezpečnosti

testovanie a simulácia môžu odhaliť chybu, ale nemôžu dokázať platnosť požadovanej vlastnosti;
techniky sú jednoduché

formálna verifikácia matematická metóda, ktorá *dokáže* platnosť požadovanej vlastnosti;
metóda overovania modelov (*model checking*)
náročné na implementáciu a samotné overenie vlastnosti;
pre niektoré systémy je problém nerozhodnuteľný

Temporálna logika

jazyk pre popis vlastností súbežných systémov

formule logiky sa vyjadrujú o pravdivosti základných tvrdení v čase (*v priebehu výpočtu*)

základné konštrukcie: globálna platnosť (globally, G) a platnosť v budúcnosti (future, F)

príklady - protokol vzájomného vylúčenia pre dva procesy

$$P_1\text{-je-v-}(1.4) \Longrightarrow \mathsf{F}\ (P_1\text{-je-v-}(1.5))$$

$$\mathsf{G}\ (\neg[P_1\text{-je-v-}(1.5) \wedge P_2\text{-je-v-}(1.5)\ ])$$

Alternatívne výpočtové modely

Kvantové počítanie

- založené na princípoch kvantovej mechaniky
- teoretický model: základnou jednotkou reprezentácie informácie je qubit, ktorý (zjednodušene) môže nadobúdať ľuboboľnú hodnotu z intervalu $[0, 1]$
- kvantové algoritmy
- otázka konštrukcie kvantového počítača

Alternatívne výpočtové modely

Molekulárne (DNA) počítanie

- DNA == reťazce nad abecedou A, C, T, G
- vývoj organizmu == manipulácia s reťazcacmi: kopírovanie, rozdeľovanie, spájanie
- 1994: experiment, v ktorom pomocou manipulácie s molekulami bola vyriešená inštancia problému Hamiltonovského cyklu veľkosti 7; experiment predstavoval niekoľko dní laboratórnej práce
- pozitívum: veľká miera paralelizmu
- negatívum: veľký objem biologického materiálu potrebného k riešeniu rozsiahlejších inštancií
- budúcnosť: ???

Alternatívne výpočtové modely

náhodnostný protokol pre večerajúcich filozofov

Porovnávanie reťazcov

Na počítačoch A a B sú uložené databázy x a y; nech x a y sú binárne reťazce dĺžky n. Úlohou je rozhodnúť, či x a y sú zhodné. Zaujíma nás, koľko bitov si musia počítače A a B vymeniť, aby dokázali vyriešiť problém rovnosti.

Dá sa dokázať, že neexistuje deterministický komunikačný protokol, ktorý by riešil problém rovnosti a pritom si A a B vymenili nanajvýš $n-1$ bitov. T.j. protokol, v ktorom A pošle celý reťazec x počítaču B je optimálny.

Porovnávanie reťazcov

Randomizovaný protokol pre problém rovnosti

Vstup $x = x_1x_2 \ldots x_n,\ y = y_1y_2 \ldots y_n$

Krok 1 A vyberie náhodne prvočíslo p z intervalu $[2, n^2]$.

Krok 2 A vypočíta číslo $s = x \mod p$ a pošle čísla s, p počítaču B.

Krok 3 B vypočíta číslo $q = y \mod p$.
Ak $q \neq s$, tak B vráti odpoveď $x \neq y$.
Ak $q = s$, tak B vráti odpoveď $x = y$.

počet bitov, ktoré si počítače pošlú je $2 \cdot \lceil \log_2 n^2 \rceil \leq 4 \cdot \lceil \log_2 n \rceil$
pravdepodobnosť, že protokol vráti nesprávnu odpoveď je $\leq \frac{\ln n^2}{n}$

Ak $n = 10^{16}$, tak zložitosť deterministického protokolu je 10^{16}, zatiaľ čo zložitosť randomizovaného protokolu je 256.
Pravdepodobnosť, že randomizovaný protokol vráti nesprávnu odopoveď je $\leq 0.36892 \cdot 10^{-14}$.

Randomizovaný Quicksort

Rand-Quicksort(A)

Vstup zoznam prvkov A

Krok 1 ak A má jeden prvok, je utriedený
ak A má viac prvkov, tak náhodne vyber prvok x z A

Krok 2 vytvor zoznam $A_<$ obsahujúci prvky z A menšie než x
vytvor zoznam $A_>$ obsahujúci prvky z A väčšie než x

Krok 3 výstup je *Rand-Quicksort*$(A_<)$, x, *Rand-Quicksort*$(A_>)$

očakávaná zložitosť algoritmu je $\mathcal{O}(N \log N)$

Typy náhodnostných algoritmov

Monte Carlo s ohraničenou pravdepodobnosťou je odpoveď nesprávna
príklad: randomizovaný protokol pre problém rovnosti

Las Vegas odpoveď je vždy správna;
cieľ: očakávaná zložitosť Las Vegas algoritmu pre problém je lepšia než zložitosť (deterministického) algorimtu
príklad: randomizovaný Quicksort

Náhodnostné zložitostné triedy

Pravdepodobnostný Turingov stroj

pracuje ako nedeterministický TS s tým rozdielom, že nedeterministický výber kroku výpočtu interpretujeme ako náhodnostnú voľbu

Trieda RP

obsahuje rozhodovacie problémy, pre ktoré existuje polynomiálne časovo ohraničený pravdepodobnostný Turingov stroj s vlastnosťou:
ak odpoveďou pre vstup X je „Nie“, tak s pravdepodobnosťou 1 stroj dá správnu odpoveď
ak odpoveďou je „Áno“, tak stroj s pravdepodobnosťou $\geq 1/2$ dá *yes*.

$$\mathrm{P} \subseteq \mathrm{RP} \subseteq \mathrm{NP}$$

náhodnostné algoritmy nemôžu efektívne riešiť problémy mimo NP; problémy z NP ale dokážu (často) riešiť s väčšou efektivitou