Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

# Nedeterminizem



## Intuitivna interpretacija nedeterminizma

- Ugibanje:
  - vsakič, ko imamo več možnost avtomat »ugane« pravo možnost
  - če je beseda v jeziku, potem takšno zaporedje »ugibanj« obstaja – takemu zaporedju pravimo **certifikat/dokazilo**.
  - če pa besede ni v jeziku, potem nobeno ugibanje ne pripelje do končnega stanja



## Nedeterministični RAM

Deluje enako kot RAM:

- dostop do poljubne lokacije v enem koraku
- ukazi enaki kot pri RAM

Dodatni ukaz:

- `izračunaj dokazilo`, s pomočjo katerega lahko preverimo članstvo besede v jeziku
- računanje dokazila traja toliko, kot je veliko dokazilo

# Eulerjev obhod

V mestu Königsberg je bilo mesto povezano s sedmimi mostovi.

Ali obstaja pot, ki se začne v nekem delu mesta (***vozlišče***), prehodi vsak most (***povezavo***) natanko enkrat in konča v istem delu mesta (***vozlišču***)?

*Leonhard Euler*

vir: *Wikipedia*

***Jezik** vseh grafov, v katerih obstaja Eulerjev obhod.*



# Hamiltonov obhod

Ali obstaja pot, ki se začne v nekem delu mesta (***vozlišču***), obišče vsak del mesta (***vozlišče***) natanko enkrat in se vrne v izhodišče?

*William Rowan Hamilton*

vir: *Wikipedia*

***Jezik** vseh grafov, v katerih obstaja Hamiltonov obhod.*



# Eulerjev in Hamiltonov obhod

***Eulerjev obhod:***

- če je graf povezan in
- če imajo vsa vozlišča v grafu sodo število sosedov.

- *Zakaj?*
- ***Zgornja pogoja lahko preverimo v številu korakov sorazmernem številu povezav.***

# Eulerjev in Hamiltonov obhod

***Hamiltonov obhod:***

- uporabimo NRAM:
  - nedeterministično dobimo dokazilo (zaporedje vozlišč, kot naj jih obiščemo)
  - se sprehodimo skozi vozlišča, da preverimo, če je dokazilo pravilno

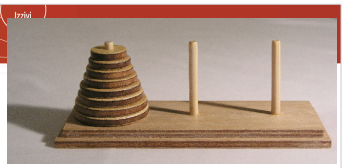
- ***Zgornji opravilo lahko izvedemo v številu korakov sorazmernem številu vozlišč.***

Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

# Problemi s praktičnimi rešitvami



# Hanojski stolpi



Hanojske stolpe imenujemo tudi bramanski stolpi. Sestoji iz $n$=64 vedno manjših zlatih diskov z luknjami v sredi. Diski so nameščeni na palčki in bramani jih morajo prestaviti po posebnem pravilu na drugo palčko. Ko bodo prestavili zadnji disk, pravi zgodba, bo konec sveta.

**Pravilo:** na palčki morajo biti diski urejeni vedno tako, da so večji diski pod manjšimi.

# Rekurzivna rešitev

Prestavi (*odTu*, *sem*, *pomožna*, *n*)
- če je *n* = 0: ne naredimo nič
- če je *n* = 1: prestavimo disk z začetne na končno palčko
- če je *n* > 1:
  - prestavimo (umaknemo) *n-1* disk *odTu* na *pomožno* palčko, pri čemer lahko palčko *sem* uporabimo za pomožno palčko
  - prestavimo najbolj spodnji (*n*-ti) disk *odTu sem*
  - prestavimo še umaknjenih *n*-1 diskov s *pomožne* palčke *sem*, pri čemer lahko *odTu* palčko uporabimo za pomožno palčko



# Rekurzivna rešitev

Število premikov P(n) =

Prestavi (*odTu*, *sem*, *pomožna*, *n*)
 if n > 0
  Prestavi (*odTu*, *pomožna*, *sem*, *n-1*) P(n-1) +
  PrestaviDisk(*odTu*, *sem*) 1 +
  Prestavi (*pomožna*, *sem*, *odTu*, *n-1*) P(n-1) =

2 P(n-1) + 1



# Rekurzivna rešitev

P(n) = 1 + 2 P(n-1) + 1 = 1 + 2(1 + 2P(n-2)) = 1 + 2 + 4P(n-2) =

$= 2^0 + 2^1 + \ldots + 2^n P(n-n) = 2^n - 1$

Pa je to veliko?

# Rekurzivna rešitev

Imamo:
- ročni premik: 10 sek
- robotski premik: 5 sek
- miselni premik: 1 msek

Nekaj časov:
- pojave modernega človeka: 50.000 let
- prvega človečnjaka: 200.000 let
- pojave življenja na zemlji: 3,5 milijarde let
- nastanka zemlje: 4,5 milijarde let
- prapoka: 13,8 milijarde let

Glej preglednico.

Raziskovanje Ustvarjanje novih svetov Izzivi FRI Študij
Univerza v Ljubljani Fakulteta za računalništvo in informatiko

# Rešitve, ki so praktične

- Praktična rešitev mora biti izračunljiva v praktičnem času
  - odvisno od problema a od nekaj minut do nekaj dni
  - Rešitev Hanojskih stolpov brez dvoma ni praktična
- Problemi s praktično rešitvijo (*tractable problems*) imajo rešitev, ki izvede za nek *n* **polinomsko** število korakov

$$2n,\ n^2,\ n \log n,\ n^3+n^2,\ \ldots$$

Raziskovanje Ustvarjanje novih svetov Izzivi FRI Študij
Univerza v Ljubljani Fakulteta za računalništvo in informatiko

# Problemi z (ne)deterministično praktično rešitvijo

- Problemi z deterministično polinomsko rešitvijo: **DP** ali na kratko **P**
  - primer: Eulerjev obhod
- Problemi z nedeterministično polinomsko rešitvijo: **NP**
  - primer: Hamiltonov obhod
- Vsak problem, ki je **P** je tudi **NP**.
- Kaj pa obratno?
  - pri KA je veljalo tudi obratno

# P in NP

- Za problem Hamiltonovega obhoda poznamo edino deterministično rešitev, ki tvori vse možne permutacije vozlišč (*n!*) in preveri vsako od njih.
  - število korakov *n!* pomeni, da za problem ne poznamo praktične deterministične rešitve
- Poznamo seznam najtežjih problemov, ki so v NP in če bi rešili kateregakoli med njimi v deterministično polinomskem (praktičnem) času, bi rešili v takšnem času vse probleme v NP – ***NP-polni problemi***.

Raziskovanje Ustvarjanje novih svetov Izzivi FRI Študij

# P in NP

- Poudarki:
  - Obstajajo problemi, ki zahtevajo eksponentno število korakov za rešitev.
  - Praktično rešljivi problemi imajo rešitev v polinomskem številu korakov glede na velikost vhodnega podatka.
  - Obstajajo deterministično (**P**) in nedeterministično (**NP**) polinomski (praktično rešljivi) problemi.
  - Vsak **P** problem je tudi **NP** problem.
  - **Ne vemo** pa, če je **NP** problem tudi **P** problem.

Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

# (Ne)moč Turingovih strojev

# Problem

Peter Zmeda uči programiranje in kot domačo nalogo daje običajno pisanje takšnih in drugačnih programov. Domače naloge mora seveda popraviti.

***Da bi pospešil popravljanje domačih nalog, se je odločil napisati program, ki bo preveril ali je oddani program pravilen.***



# Kako izgledajo programi

*izvorna koda:*

```
#include <stdio.h>
int main(int argc, char* argv[]) {
  printf ("Dober dan!\n");
}
```

*shranjeno kot (83 zlogov):*

```
00000000: 2369 6e63 6c75 6465 203c 7374 6469 6f2e
00000010: 683e 0a69 6e74 206d 6169 6e28 696e 7420
00000020: 6172 6763 2c20 6368 6172 2a20 6172 6776
00000030: 5b5d 2920 7b0a 2020 7072 696e 7466 2028
00000040: 2244 6f62 6572 2064 616e 215c 6e22 293b
00000050: 0a7d 0a
```



# Vsak program je le številka

- naš program je številka manjša od $256^{83} = 2^{664}$
- vsak program je samo številka in obratno ter
- vsak podatek je številka in obratno
- vendar:
  - nekatere številke ne predstavljajo pravilnega programa; v tem primeru naj bo njihov jezik prazen
  - nekateri programi imajo dve enaki številki; nič hudega, njuna jezika sta enaka

```
#include <stdio.h>
int main(int argc, char* argv[]) {
  printf ("Dober dan!\n");
}
```

```
#include <stdio.h>
int main(int argc,
         char* argv[]) {
  printf ("Dober dan!\n");
}
```

# Čudni jezik $L_d$

- ker je vsaka številka x lahko hkrati podatek $x_p$ (=x) in program $x_T$ (=x), se lahko vedno vprašamo ali je $x_p$ v jeziku programa $L(x_T)$: $x_p \in L(x_T)$
- podobno lahko definiramo jezik tistih opisov programov, za katere velja:

$$L_d = \{x_T : x_T \notin L(x_T)\}$$

- kako izgleda program $P_d$ za razpoznavo $L_d$?



$L_d = \{x_T : x_T \notin L(x_T)\}$

# Neizračunljivi problem $L_d$

Recimo, da obstaja program $P_d$, ki razpozna $L_d$ => potem je tudi $P_d$ neka številka.

***Ključno vprašanje:*** Ali je $P_d \in L(P_d)$?

- Recimo, da $P_d \in L(P_d)$ =>
  - $P_d \notin L_d$, toda po predpostavki) $L_d = L(P_d)$ in zato $P_d \in L(P_d)$ **protislovje**
- Potem pa recimo $P_d \notin L(P_d)$ =>
  - $P_d \in L_d$, toda ker $L_d = L(P_d)$ zato $P_d \notin L(P_d)$ **tudi protislovje**
- **Torej $P_d$ ne obstaja in zato problem članstva v jeziku $L_d$ ni izračunljiv!**



# Gödlov izrek

„Ne obstaja konsistenten sistem aksiomov in izrekov, ki bi zaobjel vso matematiko.“
*(vir: Wikipedia)*

Gödlovo nagrado podeljuje ACM za dosežke v teoretičnem računalništvu.

Kurt Gödel, 1906-1978, Wikipedia

# (Ne)moč Turingovih strojev

- Poudarki:
  - Obstajajo problemi, za katere ne obstajajo rešitve.
  - Praktičen primer problema je Petrov problem.