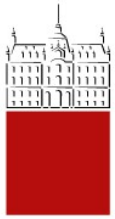
Univerza v Ljubljani
Fakulteta za računalništvo
in informatiko



Od teorije k praksi

Turingovi stroji, modeli, praktični algoritmi in kaj lahko izračunamo

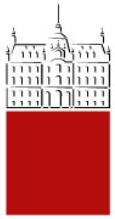
22. sučec 2014

Andrej Brodnik

Pregled dneva

- Sklop 1: Turingovi stroji (8:30 – ...)
- Sklop 2: Računalnik in TS (...)
- Sklop 3: Nedeterminizem (… – 12:00)
- Sklop 4: Problemi s praktičnimi rešitvami (12:15 – ...)
- Sklop dodatek: (Ne)moč TS (… – 13:30)
- Izdelava priprav (13:30 – 14:15)

Univerza *v Ljubljani*
Fakulteta *za računalništvo*
in informatiko



Turingovi stroji

22. sušec 2014

Alan Turing

1912 – 1954

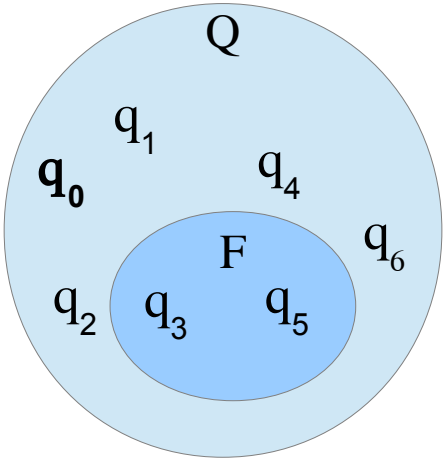
http://en.wikipedia.org/wiki/Alan_Turing

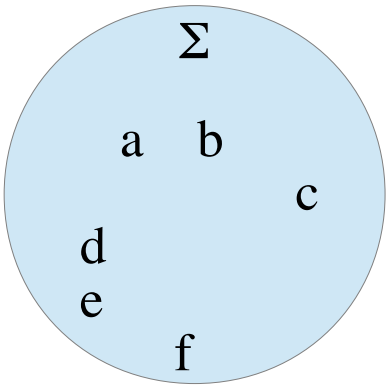
ACM podeljuje Turingovo nagrado: http://amturing.acm.org/

- *2013: Leslie Lamport*

Definicija DKA

- Peterka $\langle Q, \Sigma, \delta, q_0, F \rangle$
  - Q – končna množica stanj,
    - npr. $Q = \{ q_0, q_1, q_2, \ldots, q_n \}$
  - Σ – vhodna abeceda (tudi končna)
  - δ – funkcija prehodov
  - q_0 – začetno stanje, $q_0 \in Q$
  - F – množica končnih stanj, $F \subseteq Q$





Primer

Jezik: $0^n = 0^n$

- Zakaj KA ne razpozna jezika?
- *Intuitivno:* Končni avtomat je končen, se pravi je „velik“ *V*, kar posledično pomeni, da lahko pomni količine do največ *f(V)* – karkoli že je funkcija *f()*. V našem jeziku pa je n poljubno velik in tudi večji od *f(V)*. Slednje pomeni, da, ko KA prebere prvih *n* ničel, si preprosto ne more zapomniti, koliko ničel je prebral (če je *n* prevelik), da bi preveril, če je po enačaju prav tako *n* ničel.
- *Rešitev*: KA potrebuje dodatni pomnilnik in za slednjega bomo uporabili kar sam trak.

Razpoznava $0^n = 0^n$

- Postopek za razpoznavo bomo opisali z načrtovanjem od vrha navzdol (*top-down*)
- Pri tem bomo uporabili tehnike kot so zakrivanje (enkaspulacija, *encapsulation*), posplošitev (abstrakcija, *abstraction*) in razdelitev (modularizacija, *modularization*)

Razpoznava $0^n = 0^n$

- Razpoznavo naredimo v dveh zaporednih fazah (***načrtovanje algoritma***):
  1. ***Preverjanje pravilnosti ustroja***: najprej so same ničle, ki jim sledi enačaj ter ponovno same ničle
  2. ***Preverjanje pravilnosti velikosti***: število ničel na začetku in na koncu je enako
- Problem smo razdelili na povsem ločena dela, ki sta opisana posplošeno. Opis pove kaj dela posamezen del in ne kako.
- *Manjka*: kakšno je stanje na začetku in kakšno na koncu.

Faza 1: Preverjanje pravilnosti ustroja

- V treh korakih/stanjih (***načrtovanje algoritma***):
  - ali so na začetku same ničle
  - ali sledi enačaj
  - ali sledijo same ničle
- Na koncu vhodne besede potrebujemo poseben znak B (*blank*), da vemo, kdaj je besede konec.
- Ponovno: zakrivanje , posplošitev in razdelitev.

Vmesna faza – 1

- Na začetku nismo povedali, kje je tračna glava na začetku pred vsako fazo:
  1. ***Preverjanje pravilnosti ustroja***
  2. ***Preverjanje pravilnosti velikosti***
- Med fazama se vrnemo na začetek besede, da lahko druga faza pravilno deluje
  - pogoj (predpogoj) druge faze
- Tudi na začetku vhodne besede potrebujemo poseben znak B (*blank*).

Faza 2: Preverjanje pravilnosti velikosti

- Ideja (***načrtovanje algoritma***):
  - v enem prehodu bomo prepisali po eno 0 na začetku in na koncu z B
  - po prehodu se vrnemo na začetek
- Ponovno: zakrivanje, posplošitev in razdelitev.

Faza 2: Preverjanje pravilnosti velikosti

- En prehod:
  - ničla na začetku obstaja in jo prepišemo
    - če je še na koncu, jo prepišemo
    - če je ni na koncu, beseda ni v jeziku
  - ničle na začetku ni (je samo enačaj):
    - če je še na koncu ničla, beseda ni v jeziku
    - če je ni, beseda je v jeziku

Od KA do Turingovega stroja

- Turingov stroj je takšen, kot KA, samo da:
  - lahko pišemo nazaj na trak – pišemo črke iz abecede Γ (v našem primeru Γ = Σ U {B}
  - imamo poseben znak B, s katerim je napolnjen trak levo in desno od vhodne besede (v neskončnost)
  - prehodna funkcija δ slika iz črke in stanja v novo stanje, določi, kaj se izpiše na trak ter v katero smer se premakne glava: $\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times [L \vee D]$
- Sedmerka 〈 Q, Σ, δ, q0, F, Γ, B 〉

Turingov stroj

- V tridesetih so bili računalniki (*computers*) ljudje – običajno ženske, ker so natančnejše
- Opazil je, da uporabljajo beležnice in nekaj napišejo na list v beležnici ter nato na naslednjega ter se ponovno vrnejo na prejšnjega in tako naprej

ljudsko izročilo

Turingov stroj

- Definiral je ***a-machine*** (*automatic machine*): *On Computable Numbers, with an application to the Entscheidungsproblem*, 1936
  - *Entscheidungsproblem*: odločitveni problem, ali je beseda *w* v jeziku *L*

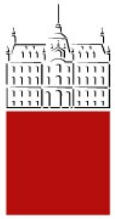
Turingov stroj

- Zakaj uporabljati?
  - zelo preprost princip (**model stroja**), kar pomeni, da lahko matematično kaj povemo o tem, kaj lahko z njim sploh izračunamo
  - preprosto „štejemo“ čas – kolikokrat smo uporabili δ: kolikokrat v našem primeru?
- Zakaj ne uporabljati?
  - resnični računalniki sploh ne izgledajo kot TS – pa res ne?

Turingov stroj

- Pri pouku:
  - preprost princip za formalno (natančno) programiranje – jflaps
  - možno poučevanje osnovnih principov zakrivanje, posplošitev in razdelitev
  - preprosto „štetje“ časa (korakov) in iz tega izpeljana zahtevnost (kompleksnost, *complexity*) postopka

Univerza v Ljubljani
Fakulteta za računalništvo
in informatiko



Računalnik in TS

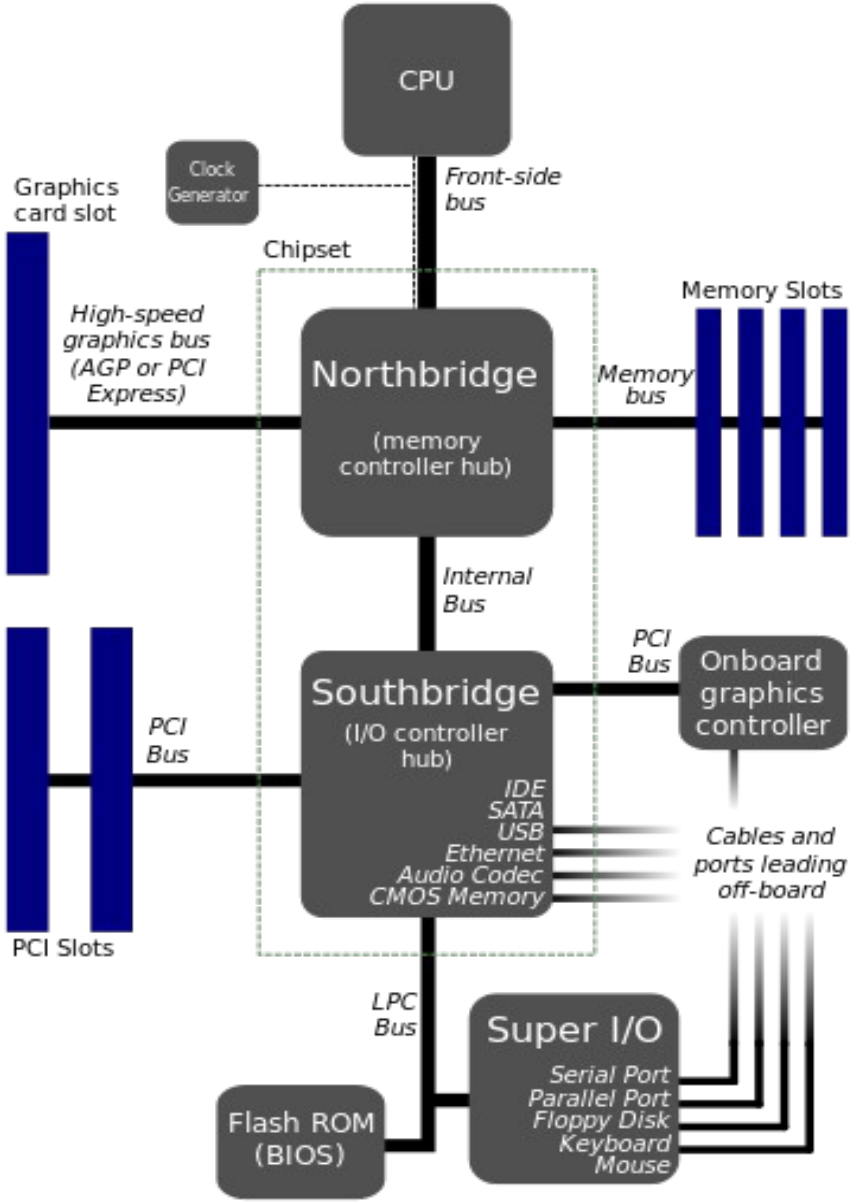
22. sušec 2014

Računalnik

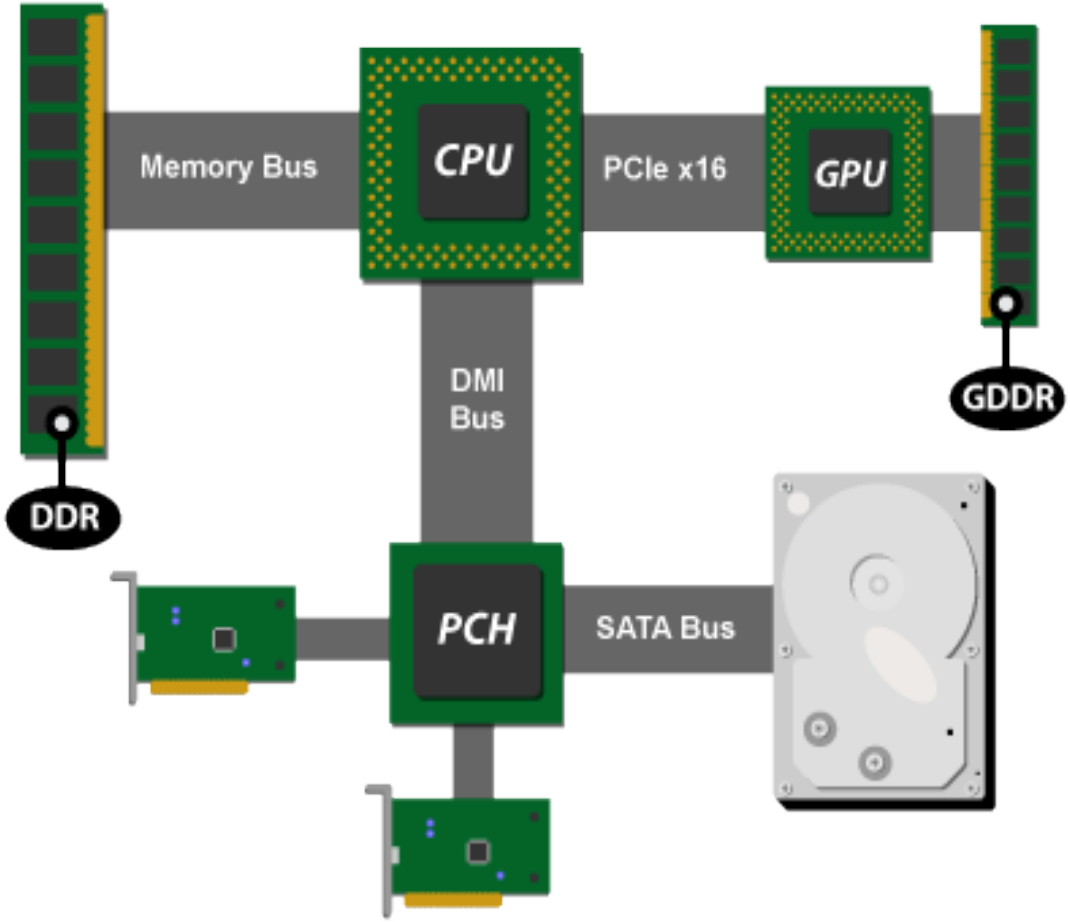
- Von Neumannova arhitektura:
  - prej: procesor, bralni pomnilnik, pisalni pomnilnik
  - sedaj: procesor in en sam pomnilnik ter med njima vodilo
- Pospešitve:
  - neposreden dostop do pomnilnika za V/I naprave (DMA)
  - poseben poudarek na grafičnih karticah in dostopu do pomnilnika (GPU)
  - predpomnilnik, več jeder, ...

Računalnik

- Vir: wikipedia



Računalnik



Jon Stokes: *Intel launches all-new PC architecture with Core i5/i7 CPUs*, ArsTechnica, 2009,
http://arstechnica.com/gadgets/2009/09/intel-launches-all-new-pc-architecture-with-core-i5i7-cpus/

Računalnik

- Kako **modelirati** računalnik?
- **CPE:**
  - **izvaja** ukaz za ukazom, pri čemer jih **bere** iz pomnilnika
  - nabor ukazov je določen v naprej
- **Pomnilnik:**
  - sestoji iz registrov (pomnilniških lokacij), kateri imajo vsak svoj naslov ter lahko iz njih CPE bere ali piše
  - do vsakega registra lahko CPE dostopi v enem koraku
- Takšnemu modelu rečemo **RAM** – *Random Access Machine*, Stroj z naključnim dostopom

RAM in TS

- Kakšen je odnos med njima?
  - sposobnost in hitrost računanja
- CPE je zelo podoben končnemu avtomatu v TS
  - končen skupek za vsak ukaz v RAM
- Pomnilnik je kot trak pri TS
  - samo dostopi na traku so obvezno zaporedni, medtem ko so v RAM lahko naključni

TS ≤ RAM

Trditev: RAM lahko **naredi vse**, kar naredi TS in **v enakem času**.

Oris dokaza:

- RAM očitno lahko simulira prehode med stanji pri TS
- trak: (i) shrani v pomnilnik od lokacije 2 naprej; in (ii) ima na lokaciji 1 hrani vrednost zadnje lokacije dostopane v pomnilniku (po njej so samo B)
- vsak korak TS lahko izvede RAM v enem koraku (konstantnem številu korakov)

RAM ≤ TS

Trditev: TS lahko **naredi vse**, kar naredi RAM in **največ v kvadratičnem času**.

Oris dokaza:

- TS ima za vsak ukaz RAM konstantno velik podsklop stanj v svojem KA
- pomnilnik: (i) kodiranje pomnilnika in vsebine; ter (ii) kodiranje dovoljuje naključno branje vrednosti iz pomnilnika z največ kvadratično zakasnitvijo
- vsak korak RAM lahko izvede TS v konstantnem številu korakov, vsak dostop do pomnilnika stane največ kvadratično upočasnitev

RAM ≤ TS

(i) Pomnilnik v RAM je definran kot skupek parov

<naslov>:<vrednost>

ki jih zapišemo na trak tako, da naslov in vrednost zapišemo dvojiško ter ločimo pare s presledki; seveda, na koncu so B.

(ii) Ko program želi dostopati do <naslova>, TS prečeše celoten trak, da najde ustrezen par.

Vprašanje: koliko stane eno česanje traku?

Odgovor: če RAM naredi v celotnem izvajanju programa *k* korakov, lahko tudi samo toliko registrov popiše. Torej eno česanje traku traja največ *k*log k* korakov. Ker mora TS simulirati *k* korakov RAM in vsak (zaradi česanja) traja *k*log k* časa, TS izvede isti program kot RAM v času *k*k*log k*.

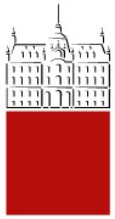
RAM in TS

- Poudarki:
  - Modeliranje stroja.
  - Pojem koraka pri računanju in ocenjevanje časovne zahtevnosti.
  - Uporaba (psevdo)kode je dovolj dober približek za (formalen) zapis postopka, ki omogoča tudi računanje časovne zahtevnosti, dokazovanje pravilnosti in podobno.

RAM in TS

- Pri pouku:
  - oblikovanje modela stroja – iskanje osnovnih lastnosti, ki nas zanimajo (ukazi, čas, ...)
  - simuliranje enega stroja z drugim
  - (pol)formalno dokazovanje, kodiranje, ...
  - štetje korakov pri izvajanju programa in ocenjevanje časovne zahtevnosti

Univerza v Ljubljani
Fakulteta za računalništvo
in informatiko



Nedeterminizem

22. sušec 2014

Formalna definicija NKA

$$M = (Q, \Sigma, \delta, q_{0,} F)$$

Q	Množica stanj
Σ	Abeceda
$\delta : Q \times \Sigma \rightarrow 2^Q$	**Funkcija prehodov, ki slika v poljubno podmnožico Q**
$q_0 \in Q$	Začetno stanje
$F \subseteq Q$	Množica končnih stanj

Nedeterministični Turingov stroj

- Nedeterministični Turingov stroj (NTS) je sedmerka

$$\langle\ Q,\ \Sigma,\ \delta,\ q0,\ F,\ \Gamma,\ B\ \rangle$$

$$\delta : Q \times \Gamma \rightarrow 2^{Q} \times \Gamma \times [L \vee D]$$

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

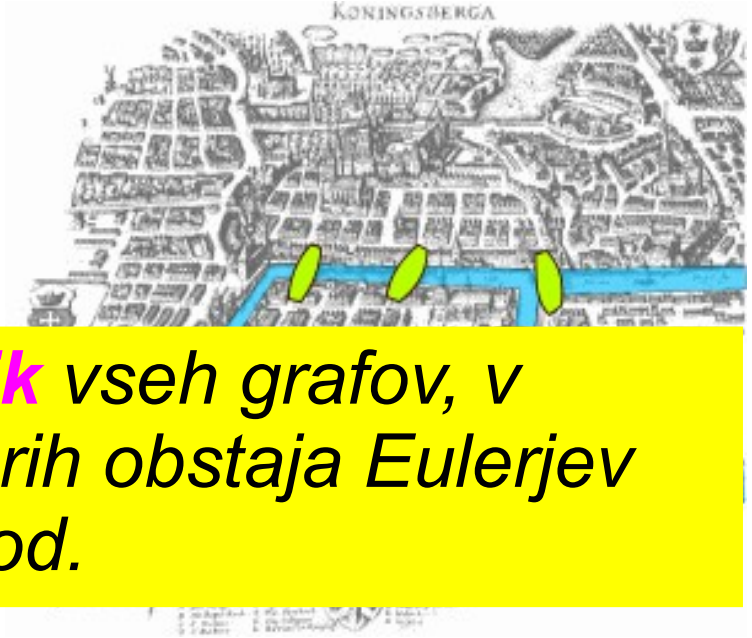
Eulerjev obhod

Leonhard Euler

vir: *Wikipedia*

V mestu Königsberg je bilo mesto povezano s sedmimi mostovi.

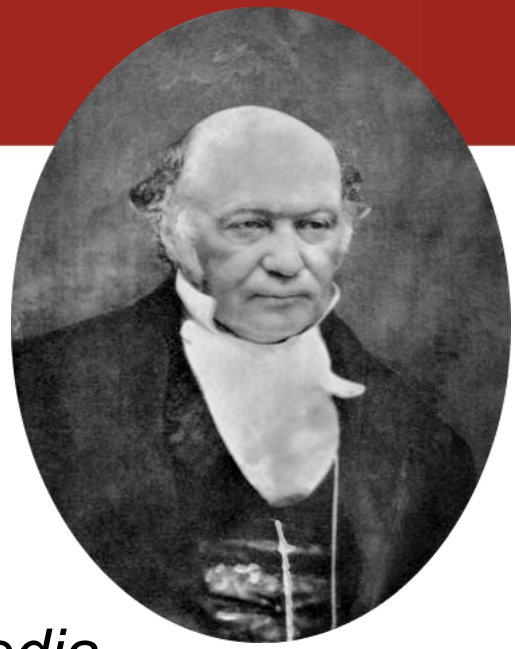
Ali obstaja pot, ki se začne v nekem delu mesta (***vozlišče***), prehodi vsak most (***povezavo***) natanko enkrat in konča v istem delu mesta (***vozlišču***)?



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

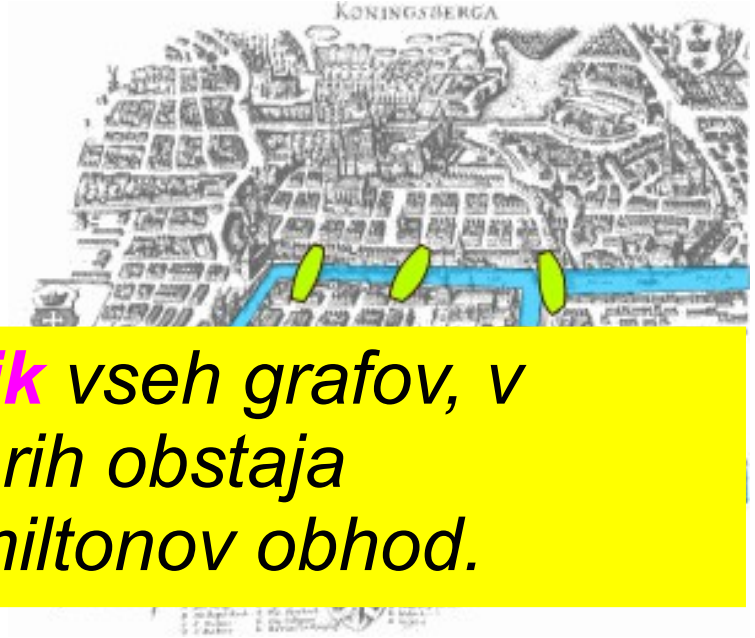
Hamiltonov obhod

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.***

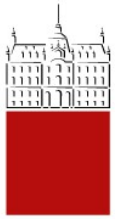
Nedeterministični RAM in TS

- Poudarki:
  - Nedeterminizem nastopa tudi pri TS na podoben način kot pri KA.
  - Uporabljamo pojem dokazila in z njegovo pomočjo preverimo pravilnost odgovora.
  - Dokazila so pri različnih problemih različna.
  - Časovna zahtevnost računanja dokazila je sorazmerna njegovi velikosti.

Nedeterministični RAM in TS

- Pri pouku:
  - oblikovanje dokazila pri različnih problemih
  - ali lahko učinkovito rešimo problem z determinističnim RAM ozroma TS

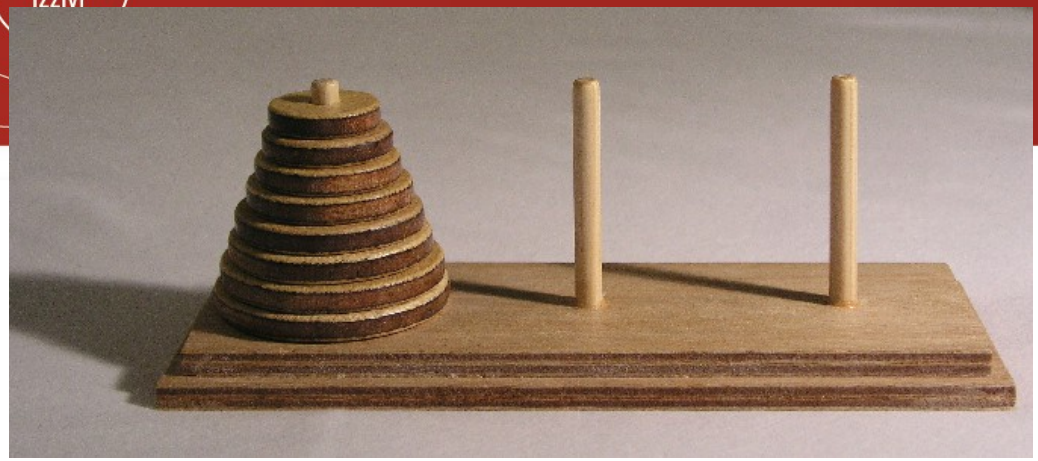
Univerza v Ljubljani
Fakulteta za računalništvo
in informatiko



Problemi s praktičnimi rešitvami

22. sušec 2014

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)
    PrestaviDisk(odTu, sem)
    Prestavi (pomožna, sem, odTu, n-1)
```

P(n-1) +
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.

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

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

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***.

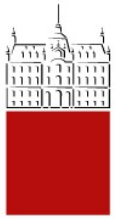
P in NP

- 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.

P in NP

- Pri pouku:
  - pridobivanje občutka kako hitro raste zahtevnost problema pri eksponentnem številu korakov
  - predstavitev dokazil za NP-polne probleme
  - spoznavanje NP-polnih problemov in kje jih srečamo v praksi

Univerza v Ljubljani
Fakulteta za računalništvo
in informatiko



(Ne)moč Turingovih strojev

22. sušec 2014

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.

(Ne)moč Turingovih strojev

- Pri pouku:
  - primer jezika L_d in dokaz, da ne obstaja P_d za L_d
  - učenci razumejo, da obstajajo problemi, ki se jih ne da rešiti

Hvala za vašo pozornost in vedno dobrodošli z vprašanji!

Andrej (Andy) Brodnik

Uroš Čibej

Nataša Kristan

Jurij Mihelič