

Sedem základných NP-úplných problémov

SAT	Vstup:	Booleovská formula f
	Problém:	Je f splniteľná?
3-SAT	Vstup:	Booleovská formula f vo forme: $(a_{1,1} \vee a_{1,2} \vee a_{1,3}) \wedge \dots \wedge (a_{n,1} \vee a_{n,2} \vee a_{n,3})$
	Problém:	Je f splniteľná?
VC	Vstup:	Graf $G = (V, E)$; číslo K
	Problém:	Existuje množina vrcholov V' veľkosti $\leq K$ taká, že pre ľubovoľnú hranu $e = (u, v) \in E$, $u \in V'$ alebo $v \in V'$?
HAM	Vstup:	Graf $G = (V, E)$
	Problém:	Existuje v grafe Hamiltonovská kružnica?

Sedem základných NP-úplných problémov (pokrač.)

TSP-D	Vstup:	Ohodnotený graf $G = (V, E)$; číslo K
	Problém:	Existuje obchôdzka dĺžky $\leq K$?
CLIQUE	Vstup:	Graf $G = (V, E)$; číslo K
	Problém:	Obsahuje G úplný podgraf o veľkosti $\geq K$ vrcholov?
SUBSET-SUM	Vstup:	n čísel $s_1, s_2, \dots, s_n$; cieľ t
	Problém:	Existuje podmnožina čísel $s_1, \dots, s_n$ so súčtom presne t ?

Teória vypočítateľnosti

Študujeme problémy, pre ktoré vieme **dokázať**, že neexistuje žiaden algoritmus, ktorý by ich riešil.

Čo je algoritmus?

Turingov stroj (TS) – model výpočtov (Allan Turing, cca 1930), videli ste na UTI

Churchova-Turingova téza: Ľubovoľný proces, ktorý prirodzene môžeme volať efektívna procedúra (alebo algoritmus) je zapísateľný ako TS.

Poznámka: Toto nie je matematická veta. Prečo?

Churchova-Turingova téza

Argumenty pre:

1. Veľa iných výpočtových modelov ekvivalentných to TS.
2. Trieda funkcií, ktorú vedie TS vypočítať je invariantná vzhľadom k rôznym modifikáciám definície TS.
3. Nepoznáme žiadnu efektívnu procedúru, ktorá by sa nedala zapísať ako TS.

Poznámka: Churchova-Turingova téza NEHOVORÍ, že TS vie vypočítať všetko **rovnako rýchlo** ako iné výpočtové modely.

RAM model výpočtov

- **Pamäť:** pole registrov $R_1, R_2, \dots$
v každom registri ľubovoľne veľké celé číslo
- **Program:** fixná postupnosť inštrukcií, očíslované riadky

Inštrukcie:

- ℓ : INC op pričítaj jednotku
- ℓ : DEC op odčítaj jednotku
- ℓ : IFZERO op ak je operand nula, choď na $\ell + 1$
inak choď na $\ell + 2$
- ℓ : GOTO op choď na riadok

Operandy:

i celé číslo i (konštanta)

R_i hodnota registra R_i

$@R_i$ hodnota registra R_{R_i}

- **Vstup a výstup:** Vstup je v R_1 , po skončení RAMu výstup v R_2

RAMy sú dostatočne silné na to, aby sme v nich simulovali TS

TS dokážu simulovať RAMy

$\Rightarrow$ (Churchova téza) **RAMy sú aspoň také silné, ako ľubovoľný iný výpočtový model.**

Príklad: RAM pre funkciu $f(n) = 2^n$

1: INC R2	// R2:=1	
2: IFZERO R1	// while	
	// R1<>0	
3: GOTO 17		
4: IFZERO R2	// R3:=R2;	
	// R2:=0;	
5: GOTO 9		
6: INC R3		
7: DEC R2		
8: GOTO 4		
		9: IFZERO R3
		// R2:=2*R3;
		// R3:=0;
		10: GOTO 15
		11: INC R2
		12: INC R2
		13: DEC R3
		14: GOTO 9
		15: DEC R1
		// R1:=R1-1;
		16: GOTO 2

- Toto je posledný program, ktorý sme napísali ako RAM :)
- Namiesto toho budeme písať pseudokódy a budeme sa spoliehať na Churchovu tézu, t.j. že sa dajú prepísať do RAMu.

Odbočka: Všetko je prirodzené číslo

Zatiaľ RAMy vedia pracovať len s číslami. Čo keď chceme pracovať s inými objektami?

Zoznam je prirodzené číslo

Zoznam $(u_1, u_2, \dots, u_n)$ môžeme reprezentovať ako prirodzené číslo:

$$2^{u_1} \cdot 3^{u_2} \cdot \dots \cdot p_i^{u_i} \cdot \dots \cdot p_n^{u_n}$$

kde p_i je i -te prvočíslo prime numbers.

Písmeno je prirodzené číslo ... použi ASCII

Reťazec je prirodzené číslo ... zoznam písmen

RAM program je prirodzené číslo ... jednoducho reťazec

Keďže všetko je prirodzené číslo, definujme ľubovoľný problém ako $f : \mathbb{N} \rightarrow \mathbb{N}$.

(Dodefinujeme $f(x) = 0$ ak x nereprezentuje platný vstup pre problém.)

Definition: Úplná funkcia $f : \mathbb{N} \rightarrow \mathbb{N}$ is **rekurzívna**/vypočítateľná akk existuje RAM, ktorý f vypočíta.

Vypočítateľnosť: Osnova

- Čo je algoritmus? Churchova-Turingova téza.
- Model výpočtov: RAM.
- Odbočka: Všetko je prirodzené číslo.
- Nevypočítateľné problémy: Problém zastavenia.
- Turingove redukcie alebo
“Ako dokázať, že môj problém nie je vypočítateľný?”
- Užitočné vypočítateľné problémy: Univerzálny RAM.
- Príklady, príklady, príklady. . .

Problém zastavenia

Problém: Daný je RAM program P a vstup x .
Zastaví sa P na vstupe x ?

$$\text{HALT}(P, x) = \begin{cases} 1, & \text{ak sa } P \text{ zastaví na } x, \\ 0, & \text{inak.} \end{cases}$$

Príklad 1:

```
trivial_function(x):  
  while x<>1 do x:=x-2
```

Príklad 2:

```
mystery_function(x):  
  while x<>1 do  
    if (x is even) then x:=x/2  
    else x:=3*x+1;
```

Veta: Neexistuje RAM, ktorý vie vypočítať funkciu HALT.

Dôkaz: Sporom.

- Predpokladajme, že existuje RAM, ktorý počíta HALT.
- Vytvorme RAM podľa nasledujúceho pseudokódu:

NOTHALT(P) :

 if HALT(P,P)=1 then loop forever;
 else return 1;

- Čo sa stane, keď spustíme

NOTHALT(NOTHALT)?

- **Predpokladajme, že $\text{NOTHALT}(\text{NOTHALT})$ zastaví.**
 - Z definície HALT : $\text{HALT}(\text{NOTHALT}, \text{NOTHALT}) = 1$
 - Z pseudokódu NOTHALT :
 $\text{NOTHALT}(\text{NOTHALT})$ sa zacyklí
 - **Ale to vedie k sporu s tým, že sa $\text{NOTHALT}(\text{NOTHALT})$ zastaví!**
- **Predpokladajme, že sa $\text{NOTHALT}(\text{NOTHALT})$ zacyklí.**
 - Z definície HALT : $\text{HALT}(\text{NOTHALT}, \text{NOTHALT}) = 0$
 - Z pseudokódu NOTHALT :
 $\text{NOTHALT}(\text{NOTHALT})$ zastaví
 - **Ale to vedie k sporu s tým, že sa $\text{NOTHALT}(\text{NOTHALT})$ zacyklí!**

Predpoklad, že existuje RAM program pre HALT nás dovedie k tomu, že dokážeme aj tvrdenie aj jeho negáciu $\Rightarrow$ predpoklad je nesprávny!

Diagonalizácia

	0	1	2	3	4	...
0	X	X		X		...
1	X	X	X	X	X	...
2						...
3		X	X	X		...
4		X	X	X		...
...	...	...	...	...	...	...

NOTHALT			X		X	...
---------	--	--	---	--	---	-----

$H(i, j)$ – X, ak sa program i zastaví na vstupe j

Môže sa NOTHALT vyskytnúť v tabuľke H ?

- NIE! Pre ľubovoľný program i sa NOTHALT od neho líši na vstupe i .
- Riadky v tabuľke H reprezentujú všetky RAM programy.
- $\Rightarrow$ neexistuje RAM program pre NOTHALT
 $\Rightarrow$ neexistuje RAM program pre HALT

Podobné dôkazy v matematike:

- **Cantorova veta:**
“Reálnych čísel je viac ako prirodzených čísel”
“Množina reálnych čísel nie je spočítateľná”
- **Gödelova veta o neúplnosti:**
“Každý formálny matematický systém, ktorý zahŕňa aritmetiku je buď nekonzistentný alebo obsahuje tvrdenia, ktoré sa v ňom nedajú dokázať.”

Ako dokážete, že vaša obľúbená funkcia Q nie je rekurzívna?

Definícia: Funkcia A je **reducibilná (v Turingovom zmysle)** na funkciu B (alebo $A \leq^T B$) ak existuje algoritmus, ktorý vypočíta A tak, že používa B ako procedúru.

Note: Rozdiely medzi $A \leq^T B$ a $A \leq_P B$:

- $\leq^T$ pre všetky problémy, nie len rozhodovacie.
- Žiadne obmedzenia na zložitosť.
- Žiadne obmedzenia na počet volaní funkcie B .

Lema: Ak A nie je rekurzívna (nie je vypočítateľná) a $A \leq^T B$, potom B nie je rekurzívna.

Príklad: **HALT_ALL**

$$\text{HALT_ALL}(P) = \begin{cases} 1, & \text{ak sa } P \text{ zastaví } \mathbf{\text{všetkých vstupoch}}, \\ 0, & \textit{inak}. \end{cases}$$

Tvrdenie: HALT_ALL nie je vypočítateľná.

Dôkaz: Redukciou z HALT

(t.j., chceme dokázať $\text{HALT} \leq^T \text{HALT_ALL}$)

- **Potrebuje:** RAM program pre funkciu HALT používajúci HALT_ALL ako precedúru.

HALT(P, x) :

 Q:=encoding of the program

 ‘‘Q(y): return P(x);’’

 return HALT_ALL(Q);

HALT(P, x):

$Q :=$ encoding of the program

 ‘‘ $Q(y) : \text{return } P(x);$ ’’

 return HALT_ALL(Q);

- **Ukážeme:** Vyššie uvedené je implementácia funkcie HALT.
 - **Predpokladajme, že sa P zastaví na x .**
Program Q zastaví na ľubovoľnom vstupe y teda HALT_ALL(Q) vráti 1.
 - **Predpokladajme, že sa P zacyklí na x .**
Program Q sa zacyklí na ľubovoľnom vstupe y teda HALT_ALL(Q) vráti 0.
- Teda $HALT \leq^T \text{HALT_ALL}$ a HALT_ALL nie je rekurzívna funkcia (resp. HALT_ALL nie je vypočítateľná).

Typický postup dôkazu nevypočítateľnosti

Chceme: Dokázať, že Q nie je rekurzívna funkcia.

- 1 Vyber funkciu P o ktorej už vieme, že nie je rekurzívna.
- 2 Napíš pseudokód pre RAM program, ktorý vypočíta funkciu P používajúc Q ako procedúru.
- 3 Zdôvodni, že pseudokód skutočne počíta P .
- 4 Keďže $P \leq^T Q$ a P nie je rekurzívna, tak Q tiež nie je rekurzívna.

Príklad: EQUIV

Dané sú dva programy (P_1, P_2) , správajú sa rovnako?

$$\text{EQUIV}(P_1, P_2) = \begin{cases} 0, & \text{ak existuje } x \text{ pre ktoré } P_1(x) \neq P_2(x), \\ 1, & \text{inak.} \end{cases}$$

Tvrdenie: EQUIV nie je rekurzívna.

Proof: Redukciou y HALT_ALL

(i.e., chceme ukázať $\text{HALT_ALL} \leq^T \text{EQUIV}$)

- **Want:** RAM pre HALT_ALL používajúc EQUIV ako procedúru.

HALT_ALL(P) :

Q:=encoding of the program ‘‘Q(y): return 0;’’

R:=encoding of the program ‘‘R(x): P(x); return 0;’’

return EQUIV(Q,R);

- **Ukážeme:** Vyššie uvedená skutočne implementuje HALT_ALL.

- **Poznámka:** Program Q sa vždy zastaví a vráti 0

- **Predpokladajme** P **zastaví na všetkých vstupoch.**

Program R zastaví na všetkých vstupoch a vráti 0

$$\Rightarrow \text{EQUIV}(Q, R) = 1$$

- **Predpokladajme** P **nezastaví na niektorom vstupe** x .

Program R sa na x zacyklí ale Q sa na x zastaví

$$\Rightarrow \text{EQUIV}(Q, R) = 0$$

- Teda $\text{HALT_ALL} \leq^T \text{EQUIV}$ a keďže HALT_ALL nie je rekurzívna funkcia, EQUIV takisto nie je rekurzívna funkcia.