Univerzálny RAM

Univerzálny RAM (alebo SIM) je RAM program, ktorý dostane na vstupe program P a číslo x a:

- ak sa P zacyklí na x, SIM(P, x) sa zacyklí
- ak P zastaví na x and vráti y, SIM(P, x) zastaví a vráti y

Ako spraviť univerzálny RAM.

- RAM program je v zásade jednoduchý asemblér
- Napíšeme simulátor v nejakom "rozumnom" jazyku
- Z Churchovej tézy—existuje RAM implementujúci to isté

Možné modifikácie univerzálneho RAM.

- simuluj iba prvých t krokov (SIM(P, x, t))
- môže odpovedať rôzne otázky o stave simulovaného RAMu po t krokoch

Univerzálne RAMy sú jednoduché

Očakávali by sme, že takéto simulátory sú zložité programy.

Nie je to pravda: ľudia dokonca súťažia, kto napíše "jednoduchší" simulátor (menší počet riadkov kódu, menší počet registrov, ...)

Ako implementovať SIM s malým počtom registrov?

- Všetky registre simulovaného stroja uložíme v jedinom registri:

$$2^{R_1} \cdot 3^{R_2} \cdot \ldots \cdot p_i^{R_i} \cdot \ldots$$

- Zvyšok simulácie bude potrebovať iba malý počet registrov — **povedzme** < 1000

Ďalší nevypočítateľný problém: Koľko pamäte program používa?

$$\text{IS\_BIG}_{10000}(P, x) = \begin{cases} 1, & \text{ak } P \text{ použije} > 10000 \text{ registrov na vstupe } x \\ 0, & \text{inak} \end{cases}$$

Tvrdenie: Funkcia IS_BIG$_{10000}$ nie je vypočítateľná.

Dôkaz: Redukciou z HALT
(t.j. chceme ukázať $\mathrm{HALT} \leq^T$ IS_BIG$_{10000}$)

- **Chceme:** RAM program pre HALT
  s IS_BIG$_{10000}$ ako procedúrou.

```
HALT(P,x):
  Q:=encoding of the program
       ‘‘Q(x): SIM(P,x); increment R1,...,R10001;’’
  return IS_BIG_10000(Q,x);
```

```
HALT(P,x):
  Q:=encoding of the program
       ``Q(x): SIM(P,x); increment R1,...,R10001;''
  return IS_BIG_10000(Q,x);
```

- **Predpokladajme P zastaví na vstupe x.**
  Potom SIM(P, x) tiež zastaví a teda
  program Q zastaví a použije ≥ 10001 registrov.
  $\Rightarrow$ IS_BIG$_{10000}(Q, x) = 1$
- **Predpokladajme P nezastaví na vstupe x.**
  Potom sa SIM(P, x) zacyklí a použije len malý počet registov
  $\Rightarrow$ IS_BIG$_{10000}(Q, x) = 0$

• Teda $\mathrm{HALT} \leq^T$ IS_BIG$_{10000}$ a keďže HALT nie je vypočítateľná, IS_BIG$_{10000}$ takisto nie je vypočítateľná.

Férovejší prístup k otázke "Koľko pamäte program používa"?

Def: Program P **pretečie** na vstupe x ak použije > 10000 registrov alebo **použije hodnotu** > 10000 **v jednom z registrov**.

$$\text{IS\_BIG}_{10000,10000}(P, x) = \begin{cases} 1, & \text{ak } P \text{ pretečie na } x \\ 0, & \text{inak} \end{cases}$$

Dobrá správa: Táto funkcia je vypočítateľná!

Myšlienka: Ak by sme vedeli, že sa P vždy zastaví, tak by sme ho mohli simulovať a počas behu kontrolovať pretečenie.

Tvrdenie: Ak program P s k riadkami beží na vstupe x bez pretečenia

$> k.10000^{10001}$

krokov, tak:

- sa zacyklí a
- nikdy nepretečie.

Dôkaz:

- Stav RAMu je jednoznačne daný:
  - obsahom všetkých nenulových registrov
  - riadkom, ktorý sa práve vykonáva
- Ak vieme stav S_i RAMu v čase i, potom stav S_{i+1} v čase $i+1$ vieme jednoznačne určiť.

- **Teda:** Ak sa rovnaký stav RAMu vyskytne v dvoch rôznych časoch $i < j$, potom sa postupnosť stavov:

$$S_i, S_{i+1}, S_{i+2}, \ldots, S_{j-1}$$

bude donekonečna opakovať a program sa zacyklí.

- **Koľko existuje "nepretečených" stavov RAMu?**

$$M = k \cdot 10000^{10001}$$

- Takže po $M + 1$ krokoch, ak RAM nepretiekol alebo sa nezastavil, **tak stav** $S_{M+1} = S_i$ **pre niektoré** $i \leq M$**.** (Dirichletov princíp)

- $\Rightarrow$ **program sa bude cykliť donekonečna a nikdy nepretečie.**

$\mathsf{IS\_BIG}_{10000,10000}$ je vypočítateľná!

```
IS_BIG_10000,10000(P,x):
  k := number of lines of P;
  SIM(P,x,k.10000^10001+1);

  if P did overflow during simulation
     return 1;
  else
     return 0;
```

Vypočítateľnosť: Zhrnutie

- Churchova-Turingova téza: všetko je Turingov stroj (...alebo RAM)
- Niektoré problémy (napr. HALT) nie sú vypočítateľné
- Nevypočítateľnosť môžeme dokazovať pomocou Turingových redukcií
- Niekedy malá zmena môže znamenať rozdiel medzi vypočítateľnou a nevypočítateľnou funkciou.
- Univerzálne RAMy nám môžu pomôcť pre dokazovaní, že funkcia je vypočítateľná aj pri dokazovaní, že funkcia nie je vypočítateľná.

“Vzor” dôkazu správnosti greedy algoritmu

Lema: Predpokladajme, že greedy algoritmus vráti riešenie G. Potom existuje optimálne riešenie, ktoré sa s riešením G zhoduje na prvých k voľbách.

Dôkaz: Matematickou indukciou podľa k.

Báza indukcie. Pre $k = 0$ – ľubovoľné optimálne riešenie.

Indukčný krok. (Prepokladajme, že sme neurobili chybu pri prvých k voľbách, potom aj $(k + 1)$-vá voľba je OK.)

- Predpokladajme, že existuje optimálne riešenie OPT, ktoré sa zhoduje s G na prvých k voľbách.
- Vyrobíme riešenie OPT':
  - OPT' má rovnakú hodnotu ako OPT (a preto je tiež optimálne)
  - OPT' súhlasí s G na jednej ďalšej $(k + 1)$-vej voľbe.

Dynamické programovanie

1. **Určíme podproblém.**
   - aké sú rozmery matice, ktorú budeme vypĺňať?
   - aký je presný význam každého políčka matice?
   - kde v matici nájdeme riešenie pôvodnej úlohy?
2. **Vyriešime podproblém za pomoci iných podproblémov.**
   Ako vypočítame jedno políčko matice z iných políčiek matice?
3. **Bázové podproblémy.** Ktoré políčka nemožno vypočítať pomocou vzťahov z predchádzajúceho kroku? Aké hodnoty by mali obsahovať?
4. **Vyberieme poradie vypĺňania.** V akom poradí musíme maticu vypĺňať tak, aby sme v každom kroku mali vypočítané všetky políčka, ktoré potrebujeme na výpočet daného políčka?

Master theorem

Nech $T(n) = aT(n/b) + f(n)$, $T(1) = \Theta(1)$. Nech $k = \log_b a$. Potom:

1. Ak $\boxed{f(n) \in O(n^{k-\varepsilon})}$ pre niektoré $\varepsilon > 0$, potom $\boxed{T(n) \in \Theta(n^k)}$.
2. Ak $\boxed{f(n) \in \Theta(n^k)}$, potom $\boxed{T(n) \in \Theta(f(n) \log n)}$.
3. Ak $\boxed{f(n) \in \Omega(n^{k+\varepsilon})}$ pre niektoré $\varepsilon > 0$ a platí podmienka regularity, potom $\boxed{T(n) \in \Theta(f(n))}$.

Podmienka regularity: Existuje $c < 1$ také, že pre všetky dostatočne veľké n platí $af(n/b) \leq cf(n)$.

Poznámka: Veta platí aj v prípade rozumných usporiadaní dolných a horných celých častí - viď napr. CLRS2 4.4.2.

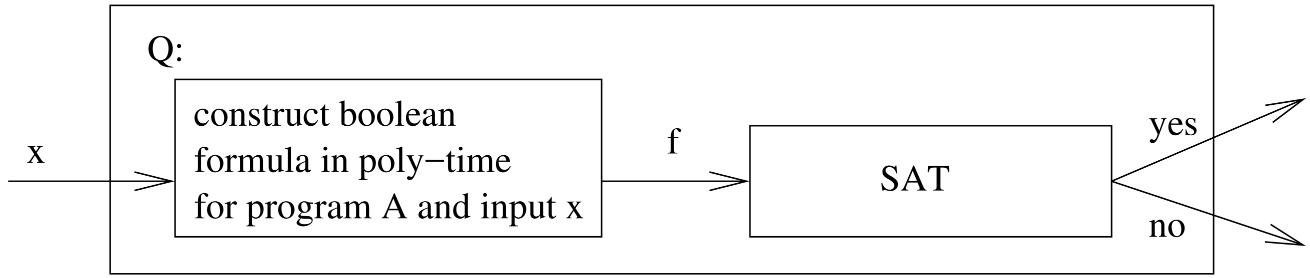
Nedeterministický algoritmus pre riešenie TSP-D

```
function TSP-D
  visited[i]:=false for all vertices;
  last_visited:=1; visited[1]:=true;
  length:=0;
  repeat n-1 times
    choose next_visited between 1 and n;
    if visited[next_visited] then reject;
    //we cannot visit a single vertex twice
    visited[next_visited]:=true;
    length:=length+w(last_visited,next_visited);
    last_visited:=next_visited;
  length:=length+w(last_visited,1);
  if length<=B then accept;
               else reject;
```

SAT je NP-ťaždký: zhrnutie

Vyššieuvedeným postup skonštruujeme pre daný algoritmus A a vstup x formulu f:

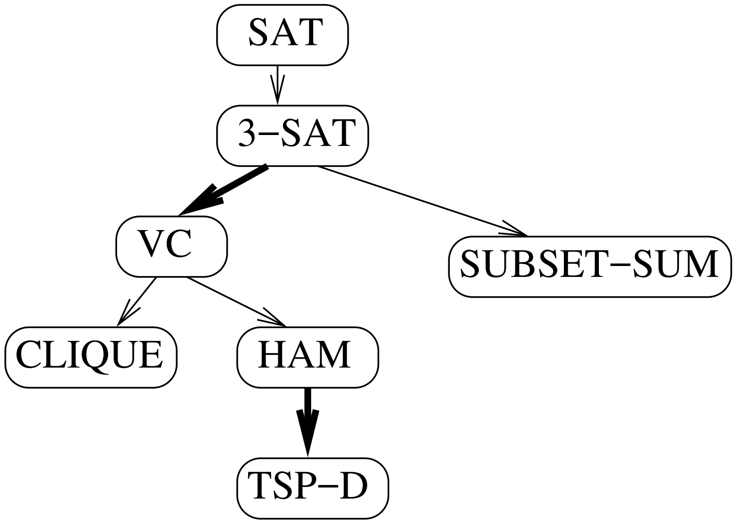
- Postup možno zrealizovať v polynomiálnom čase v závislosti od n.
- Výsledná formula má polynomiálnu veľkosť v závislosti od n.
- f je splniteľná $\iff$ A akceptuje x



$\Rightarrow$ **Ukázali sme: $Q \leq_p \mathrm{SAT}$ pre ľubovoľné $Q \in NP$**

Ako dokázať, že problém Q je NP-ťažký?

1. Vyberme si problém N o ktorom už vieme, že je NP-úplný
2. Ukážeme $N \leq_P Q$:
   - Navrhneme polynomiálny algoritmus, ktorý prerobí vstup x pre problém N na vstup $f(x)$ pre problém Q.
   - Dokážeme: Ak je x pozitívny vstup pre N, potom
     $f(x)$ je pozitívny vstup pre Q
   - Dokážeme: Ak je x negatívny vstup pre N, potom
     $f(x)$ je negatívny vstup pre Q
     —ALEBO—
     Ak $f(x)$ je pozitívny vstup pre Q, potom
     x je pozitívny vstup pre N
3. Keďže N je NP-úplný, Q musí byť NP-ťažký.



A $\longrightarrow$ B means reduction A to B

means reduction shown already

Diagonalizácia

	0	1	2	3	4	...
0	$\boxed{X}$	X		X		...
1	X	$\boxed{X}$	X	X	X	...
2			$\boxed{}$			...
3		X	X	$\boxed{X}$		...
4		X	X	X	$\boxed{}$	...
...	...	...	...	...	...	...
NOTHALT			X		X	...

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

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.

Univerzálny RAM

Univerzálny RAM (alebo SIM) je RAM program, ktorý dostane na vstupe program P a číslo x and:

- ak sa P zacyklí na x, SIM(P, x) sa zacyklí
- ak P zastaví na x and vráti y, SIM(P, x) zastaví a vráti y

Univerzálny RAM je rekurzívny.

- RAM program je v zásade jednoduchý asemblér
- Napíšeme simulátor v nejakom "rozumnom" jazyku
- Z Churchovej tézy—existuje RAM implementujúci to isté

Možné modifikácie univerzálneho RAM.

- simuluj iba prvých t krokov (SIM(P, x, t))
- môže odpovedať rôzne otázky o stave simulovaného RAMu po t krokoch