Számítási modellek

6. előadás

RAM gép

A RAM gép egy speciális **regiszter gép**.

RAM gép

A RAM gép egy speciális **regiszter gép**. A regiszter gépek olyan gépek, ahol

- adott vagy előre meghatározott számú vagy korlátlan számú regiszter (memóriarekesz), amit együtt tekinthetünk a gép memóriájának.

RAM gép

A RAM gép egy speciális **regiszter gép**. A regiszter gépek olyan gépek, ahol

- adott vagy előre meghatározott számú vagy korlátlan számú regiszter (memóriarekesz), amit együtt tekinthetünk a gép memóriájának.
- a regiszterek tartalma egy korlátlan méretű természetes szám lehet (úgy is gondolhatunk rá, hogy a regiszterekben zsetonok vannak) [mi: kényelmi szempontból negatív számok is],

RAM gép

A RAM gép egy speciális **regiszter gép**. A regiszter gépek olyan gépek, ahol

- adott vagy előre meghatározott számú vagy korlátlan számú regiszter (memóriarekesz), amit együtt tekinthetünk a gép memóriájának.
- a regiszterek tartalma egy korlátlan méretű természetes szám lehet (úgy is gondolhatunk rá, hogy a regiszterekben zsetonok vannak) [mi: kényelmi szempontból negatív számok is],
- a gép egy limitált utasításkészlettel rendelkezik, ami tipikusan tartalmaz minimum 1-2 aritmetikai és egy kontrol (feltételes ugrás) utasítást,

RAM gép

A RAM gép egy speciális **regiszter gép**. A regiszter gépek olyan gépek, ahol

- adott vagy előre meghatározott számú vagy korlátlan számú regiszter (memóriarekesz), amit együtt tekinthetünk a gép memóriájának.
- a regiszterek tartalma egy korlátlan méretű természetes szám lehet (úgy is gondolhatunk rá, hogy a regiszterekben zsetonok vannak) [mi: kényelmi szempontból negatív számok is],
- a gép egy limitált utasításkészlettel rendelkezik, ami tipikusan tartalmaz minimum 1-2 aritmetikai és egy kontrol (feltételes ugrás) utasítást,
- a címkézett utasítások listája lehet külön programtárban vagy valamely regiszterben tárolva,

RAM gép

A RAM gép egy speciális **regiszter gép**. A regiszter gépek olyan gépek, ahol

- adott vagy előre meghatározott számú vagy korlátlan számú regiszter (memóriarekesz), amit együtt tekinthetünk a gép memóriájának.
- a regiszterek tartalma egy korlátlan méretű természetes szám lehet (úgy is gondolhatunk rá, hogy a regiszterekben zsetonok vannak) [mi: kényelmi szempontból negatív számok is],
- a gép egy limitált utasításkészlettel rendelkezik, ami tipikusan tartalmaz minimum 1-2 aritmetikai és egy kontrol (feltételes ugrás) utasítást,
- a címkézett utasítások listája lehet külön programtárban vagy valamely regiszterben tárolva,
- az utasításkészlet egyes modelleknél tartalmazhat indirekt címzésű utasítást is,

RAM gép

A RAM gép egy speciális **regiszter gép**. A regiszter gépek olyan gépek, ahol

- adott vagy előre meghatározott számú vagy korlátlan számú regiszter (memóriarekesz), amit együtt tekinthetünk a gép memóriájának.
- a regiszterek tartalma egy korlátlan méretű természetes szám lehet (úgy is gondolhatunk rá, hogy a regiszterekben zsetonok vannak) [mi: kényelmi szempontból negatív számok is],
- a gép egy limitált utasításkészlettel rendelkezik, ami tipikusan tartalmaz minimum 1-2 aritmetikai és egy kontrol (feltételes ugrás) utasítást,
- a címkézett utasítások listája lehet külön programtárban vagy valamely regiszterben tárolva,
- az utasításkészlet egyes modelleknél tartalmazhat indirekt címzésű utasítást is,
- az aritmetikai utasításokat vagy minden vagy csak egy speciális regiszter tudja elvégezni.

RAM gép (informális kép)

Példa:

	$x[-1]$	$x[0]$	$x[1]$	$x[2]$	$x[3]$	$x[4]$	$x[5]$	$x[6]$	$x[7]$
Cím	-1	0	1	2	3	4	5	6	7
Érték		5	12	-15	33	3	-2		

Memória

0	$x[5] := x[5] + 1;$
1	IF $x[5] \leqslant 0$ THEN GOTO 0;
2	$x[x[4]] := x[1];$
3	IF $x[2] \leqslant 0$ THEN GOTO 2;
4	
5	
6	
7	

Programtár

RAM gép (informális kép)

- A RAM gép működése a **programtárjának** és a **memóriájának** tartalmától függ.

RAM gép (informális kép)

- A RAM gép működése a **programtárjának** és a **memóriájának** tartalmától függ.
- A memória végtelen sok regiszterből (memóriarekeszből) áll, melyek az egész számokkal ($\mathbb{Z}$ elemivel) vannak címezve.

RAM gép (informális kép)

- A RAM gép működése a **programtárjának** és a **memóriájának** tartalmától függ.
- A memória végtelen sok regiszterből (memóriarekeszből) áll, melyek az egész számokkal ($\mathbb{Z}$ elemivel) vannak címezve.
- Minden $i \in \mathbb{Z}$ esetén az i-edik memóriarekesz egy $x[i] \in \mathbb{Z}$ egész számot tartalmaz. A memóriarekeszek közül mindig csak véges soknak nem 0 a tartalma. Az egyes memóriarekeszek tartalma a működés során változhat.

RAM gép (informális kép)

- A RAM gép működése a **programtárjának** és a **memóriájának** tartalmától függ.
- A memória végtelen sok regiszterből (memóriarekeszből) áll, melyek az egész számokkal ($\mathbb{Z}$ elemivel) vannak címezve.
- Minden $i \in \mathbb{Z}$ esetén az i-edik memóriarekesz egy $x[i] \in \mathbb{Z}$ egész számot tartalmaz. A memóriarekeszek közül mindig csak véges soknak nem 0 a tartalma. Az egyes memóriarekeszek tartalma a működés során változhat.
- A programtár potenciálisan végtelen sok, $0, 1, 2, \ldots$ címkékkel ellátott rekeszből áll; ebbe egy adott gépi kódszerű programnyelven egy véges hosszúságú programot írhatunk (rekeszenként 1 utasítással). A RAM gép programja a gép működése során állandó.

RAM gép (informális kép)

- A RAM gép működése a **programtárjának** és a **memóriájának** tartalmától függ.
- A memória végtelen sok regiszterből (memóriarekeszből) áll, melyek az egész számokkal ($\mathbb{Z}$ elemivel) vannak címezve.
- Minden $i \in \mathbb{Z}$ esetén az i-edik memóriarekesz egy $x[i] \in \mathbb{Z}$ egész számot tartalmaz. A memóriarekeszek közül mindig csak véges soknak nem 0 a tartalma. Az egyes memóriarekeszek tartalma a működés során változhat.
- A programtár potenciálisan végtelen sok, $0, 1, 2, \ldots$ címkékkel ellátott rekeszből áll; ebbe egy adott gépi kódszerű programnyelven egy véges hosszúságú programot írhatunk (rekeszenként 1 utasítással). A RAM gép programja a gép működése során állandó.
- Két fontos különbség a **számláló gépekhez** képest (amik a legegyszerűbb regiszter gépek): a korlátlan számban rendelkezésre álló memóriarekesz és az indirekt címzésű utasítások használata.

A RAM gép

Definíció

Egy $M = \langle x, P\rangle$ **RAM gép** két komponensből áll: az $\langle x[i]\rangle_{i\in\mathbb{Z}}$ memóriarekeszekből és a P programból. $P = \langle(0, \omega_0), \ldots (n, \omega_n)\rangle$ egy véges sorozat ($n \in \mathbb{N}$), ahol minden $1 \leqslant k \leqslant n$ esetén az ω_k az alábbiak valamelyike

- $x[i] := 0;$ $\qquad (i \in \mathbb{Z})$

A RAM gép

Definíció

Egy $M = \langle x, P\rangle$ **RAM gép** két komponensből áll: az $\langle x[i]\rangle_{i\in\mathbb{Z}}$ memóriarekeszekből és a P programból. $P = \langle(0, \omega_0), \dots (n, \omega_n)\rangle$ egy véges sorozat ($n \in \mathbb{N}$), ahol minden $1 \leqslant k \leqslant n$ esetén az ω_k az alábbiak valamelyike

- $x[i] := 0;$ $\qquad (i \in \mathbb{Z})$
- $x[i] := x[i] + 1;$ $\qquad (i \in \mathbb{Z})$

A RAM gép

Definíció

Egy $M = \langle x, P\rangle$ **RAM gép** két komponensből áll: az $\langle x[i]\rangle_{i\in\mathbb{Z}}$ memóriarekeszekből és a P programból. $P = \langle(0, \omega_0), \ldots (n, \omega_n)\rangle$ egy véges sorozat ($n \in \mathbb{N}$), ahol minden $1 \leqslant k \leqslant n$ esetén az ω_k az alábbiak valamelyike

- $x[i] := 0;$ $(i \in \mathbb{Z})$
- $x[i] := x[i] + 1;$ $(i \in \mathbb{Z})$
- $x[i] := x[i] - 1;$ $(i \in \mathbb{Z})$

A RAM gép

Definíció

Egy $M = \langle x, P \rangle$ **RAM gép** két komponensből áll: az $\langle x[i] \rangle_{i \in \mathbb{Z}}$ memóriarekeszekből és a P programból. $P = \langle (0, \omega_0), \ldots (n, \omega_n) \rangle$ egy véges sorozat ($n \in \mathbb{N}$), ahol minden $1 \leqslant k \leqslant n$ esetén az ω_k az alábbiak valamelyike

- $x[i] := 0;$ $(i \in \mathbb{Z})$
- $x[i] := x[i] + 1;$ $(i \in \mathbb{Z})$
- $x[i] := x[i] - 1;$ $(i \in \mathbb{Z})$
- $x[i] := x[i] + x[j];$ $(i, j \in \mathbb{Z})$

A RAM gép

Definíció

Egy $M = \langle x, P \rangle$ **RAM gép** két komponensből áll: az $\langle x[i] \rangle_{i \in \mathbb{Z}}$ memóriarekeszekből és a P programból. $P = \langle (0, \omega_0), \ldots (n, \omega_n) \rangle$ egy véges sorozat ($n \in \mathbb{N}$), ahol minden $1 \leqslant k \leqslant n$ esetén az ω_k az alábbiak valamelyike

- $x[i] := 0;$ $\quad (i \in \mathbb{Z})$
- $x[i] := x[i] + 1;$ $\quad (i \in \mathbb{Z})$
- $x[i] := x[i] - 1;$ $\quad (i \in \mathbb{Z})$
- $x[i] := x[i] + x[j];$ $\quad (i, j \in \mathbb{Z})$
- $x[i] := x[i] - x[j];$ $\quad (i, j \in \mathbb{Z})$

A RAM gép

Definíció

Egy $M = \langle x, P \rangle$ **RAM gép** két komponensből áll: az $\langle x[i] \rangle_{i \in \mathbb{Z}}$ memóriarekeszekből és a P programból. $P = \langle (0, \omega_0), \dots (n, \omega_n) \rangle$ egy véges sorozat ($n \in \mathbb{N}$), ahol minden $1 \leqslant k \leqslant n$ esetén az ω_k az alábbiak valamelyike

- $x[i] := 0;$ $(i \in \mathbb{Z})$
- $x[i] := x[i] + 1;$ $(i \in \mathbb{Z})$
- $x[i] := x[i] - 1;$ $(i \in \mathbb{Z})$
- $x[i] := x[i] + x[j];$ $(i, j \in \mathbb{Z})$
- $x[i] := x[i] - x[j];$ $(i, j \in \mathbb{Z})$
- $x[x[i]] := x[j];$ $(i, j \in \mathbb{Z})$

A RAM gép

Definíció

Egy $M = \langle x, P\rangle$ **RAM gép** két komponensből áll: az $\langle x[i]\rangle_{i\in\mathbb{Z}}$ memóriarekeszekből és a P programból. $P = \langle(0, \omega_0), \ldots (n, \omega_n)\rangle$ egy véges sorozat ($n \in \mathbb{N}$), ahol minden $1 \leqslant k \leqslant n$ esetén az ω_k az alábbiak valamelyike

- $x[i] := 0;$ $(i \in \mathbb{Z})$
- $x[i] := x[i] + 1;$ $(i \in \mathbb{Z})$
- $x[i] := x[i] - 1;$ $(i \in \mathbb{Z})$
- $x[i] := x[i] + x[j];$ $(i, j \in \mathbb{Z})$
- $x[i] := x[i] - x[j];$ $(i, j \in \mathbb{Z})$
- $x[x[i]] := x[j];$ $(i, j \in \mathbb{Z})$
- $x[i] := x[x[j]];$ $(i, j \in \mathbb{Z})$

A RAM gép

Definíció

Egy $M = \langle x, P \rangle$ **RAM gép** két komponensből áll: az $\langle x[i] \rangle_{i \in \mathbb{Z}}$ memóriarekeszekből és a P programból. $P = \langle (0, \omega_0), \ldots (n, \omega_n) \rangle$ egy véges sorozat ($n \in \mathbb{N}$), ahol minden $1 \leqslant k \leqslant n$ esetén az ω_k az alábbiak valamelyike

- $x[i] := 0;$ $\quad (i \in \mathbb{Z})$
- $x[i] := x[i] + 1;$ $\quad (i \in \mathbb{Z})$
- $x[i] := x[i] - 1;$ $\quad (i \in \mathbb{Z})$
- $x[i] := x[i] + x[j];$ $\quad (i, j \in \mathbb{Z})$
- $x[i] := x[i] - x[j];$ $\quad (i, j \in \mathbb{Z})$
- $x[x[i]] := x[j];$ $\quad (i, j \in \mathbb{Z})$
- $x[i] := x[x[j]];$ $\quad (i, j \in \mathbb{Z})$
- IF $x[i] \leqslant 0$ THEN GOTO p; $\quad (i \in \mathbb{Z},\ 0 \leqslant p \leqslant n)$

RAM gép bemenete

Mivel az első komponens, a memória szerkezete minden RAM gépre azonos ezért a RAM gépet azonosíthatjuk a P programjával.

RAM gép bemenete

Mivel az első komponens, a memória szerkezete minden RAM gépre azonos ezért a RAM gépet azonosíthatjuk a P programjával.

Definíció

Egy **RAM-gép bemenete** egy természetes számokból álló véges sorozat, amelynek a hosszát az $x[0]$ memóriarekeszbe, elemeit pedig rendre az $x[1], x[2], \ldots x[x[0]]$ memóriarekeszekbe írjuk be. A többi memóriarekesz tartalma 0.

RAM gép bemenete

Mivel az első komponens, a memória szerkezete minden RAM gépre azonos ezért a RAM gépet azonosíthatjuk a P programjával.

Definíció

Egy **RAM-gép bemenete** egy természetes számokból álló véges sorozat, amelynek a hosszát az $x[0]$ memóriarekeszbe, elemeit pedig rendre az $x[1], x[2], \dots x[x[0]]$ memóriarekeszekbe írjuk be. A többi memóriarekesz tartalma 0.

Megjegyzés: A bemenet hosszát azért tároljuk el $x[0]$-ban, hogy lehessen 0 is egy bemenethez tartozó rekesz tartalma.

RAM gép bemenete

Mivel az első komponens, a memória szerkezete minden RAM gépre azonos ezért a RAM gépet azonosíthatjuk a P programjával.

Definíció

Egy **RAM-gép bemenete** egy természetes számokból álló véges sorozat, amelynek a hosszát az $x[0]$ memóriarekeszbe, elemeit pedig rendre az $x[1], x[2], \ldots x[x[0]]$ memóriarekeszekbe írjuk be. A többi memóriarekesz tartalma 0.

Megjegyzés: A bemenet hosszát azért tároljuk el $x[0]$-ban, hogy lehessen 0 is egy bemenethez tartozó rekesz tartalma.

Teljes indukcióval könnyen látható, hogy a program végrehajtása során a nem-0 tartalmú memóriarekeszek száma mindig véges.

RAM gép bemenete

Mivel az első komponens, a memória szerkezete minden RAM gépre azonos ezért a RAM gépet azonosíthatjuk a P programjával.

Definíció

Egy **RAM-gép bemenete** egy természetes számokból álló véges sorozat, amelynek a hosszát az $x[0]$ memóriarekeszbe, elemeit pedig rendre az $x[1], x[2], \ldots x[x[0]]$ memóriarekeszekbe írjuk be. A többi memóriarekesz tartalma 0.

Megjegyzés: A bemenet hosszát azért tároljuk el $x[0]$-ban, hogy lehessen 0 is egy bemenethez tartozó rekesz tartalma.

Teljes indukcióval könnyen látható, hogy a program végrehajtása során a nem-0 tartalmú memóriarekeszek száma mindig véges.

Észrevétel: Az értékadások egész szám(ok)ból aritmetikai műveletekkel egész értéket adnak vissza, így (mivel a rekeszek tartalma kezdetben egész) a működés során $x[i] \in \mathbb{Z}$ mindvégig fennáll ($i \in \mathbb{Z}$).

RAM gép működése és kimenete

Definíció

- A RAM gép programja a feltételes ugrás kivételével szekvenciálisan hajtódik végre, $(i-1, \omega_{i-1})$-t (i, ω_i) követi $(1 \leqslant i \leqslant n)$.

RAM gép működése és kimenete

Definíció

- A RAM gép programja a feltételes ugrás kivételével szekvenciálisan hajtódik végre, $(i-1, \omega_{i-1})$-t (i, ω_i) követi $(1 \leqslant i \leqslant n)$.
- Ha ω_k IF $x[i] \leqslant 0$ THEN GOTO p; $(i \in \mathbb{Z},\ 0 \leqslant p \leqslant n)$ típusú utasítás, akkor a feltétel teljesülése esetén a program (p, ω_p)-vel, egyébként pedig $(k+1, \omega_{k+1})$-el folytatódik.

RAM gép működése és kimenete

Definíció

- A RAM gép programja a feltételes ugrás kivételével szekvenciálisan hajtódik végre, $(i-1, \omega_{i-1})$-t (i, ω_i) követi $(1 \leqslant i \leqslant n)$.
- Ha ω_k IF $x[i] \leqslant 0$ THEN GOTO p; $(i \in \mathbb{Z},\ 0 \leqslant p \leqslant n)$ típusú utasítás, akkor a feltétel teljesülése esetén a program (p, ω_p)-vel, egyébként pedig $(k+1, \omega_{k+1})$-el folytatódik.
- Egy RAM gép program működése **végetér**, ha egy olyan programsorhoz ér, amiben nincs utasítás.

RAM gép működése és kimenete

Definíció

- A RAM gép programja a feltételes ugrás kivételével szekvenciálisan hajtódik végre, $(i-1, \omega_{i-1})$-t (i, ω_i) követi $(1 \leqslant i \leqslant n)$.
- Ha ω_k IF $x[i] \leqslant 0$ THEN GOTO p; $(i \in \mathbb{Z},\ 0 \leqslant p \leqslant n)$ típusú utasítás, akkor a feltétel teljesülése esetén a program (p, ω_p)-vel, egyébként pedig $(k+1, \omega_{k+1})$-el folytatódik.
- Egy RAM gép program működése **végetér**, ha egy olyan programsorhoz ér, amiben nincs utasítás.
- **A RAM gép kimenete** alatt a program megállásakor az $\langle x[i] \rangle_{i \in \mathbb{Z}}$ rekeszek tartalmát értjük.

RAM gép működése és kimenete

Definíció

- A RAM gép programja a feltételes ugrás kivételével szekvenciálisan hajtódik végre, $(i-1, \omega_{i-1})$-t (i, ω_i) követi $(1 \leqslant i \leqslant n)$.
- Ha ω_k IF $x[i] \leqslant 0$ THEN GOTO p; $(i \in \mathbb{Z},\ 0 \leqslant p \leqslant n)$ típusú utasítás, akkor a feltétel teljesülése esetén a program (p, ω_p)-vel, egyébként pedig $(k+1, \omega_{k+1})$-el folytatódik.
- Egy RAM gép program működése **végetér**, ha egy olyan programsorhoz ér, amiben nincs utasítás.
- **A RAM gép kimenete** alatt a program megállásakor az $\langle x[i] \rangle_{i \in \mathbb{Z}}$ rekeszek tartalmát értjük.

A fentiek alapján a kimenet mindig végesen reprezentálható.

RAM gép működése és kimenete

Definíció

- A RAM gép programja a feltételes ugrás kivételével szekvenciálisan hajtódik végre, $(i-1, \omega_{i-1})$-t (i, ω_i) követi $(1 \leqslant i \leqslant n)$.
- Ha ω_k IF $x[i] \leqslant 0$ THEN GOTO p; $(i \in \mathbb{Z},\ 0 \leqslant p \leqslant n)$ típusú utasítás, akkor a feltétel teljesülése esetén a program (p, ω_p)-vel, egyébként pedig $(k+1, \omega_{k+1})$-el folytatódik.
- Egy RAM gép program működése **végetér**, ha egy olyan programsorhoz ér, amiben nincs utasítás.
- **A RAM gép kimenete** alatt a program megállásakor az $\langle x[i] \rangle_{i \in \mathbb{Z}}$ rekeszek tartalmát értjük.

A fentiek alapján a kimenet mindig végesen reprezentálható.

Alternatív kimenet: kijelölhetünk bizonyos rekesz(eke)t kimeneti rekesszé.

RAM gép programjának futási ideje

A végrehajtott utasítások száma, nem a legjobb mérőszáma annak, hogy mennyit dolgozik a RAM gép.

Két nagyon nagy természetes szám összeadása ne legyen egységnyi művelet.

RAM gép programjának futási ideje

A végrehajtott utasítások száma, nem a legjobb mérőszáma annak, hogy mennyit dolgozik a RAM gép.

Két nagyon nagy természetes szám összeadása ne legyen egységnyi művelet.

Definíció

A **logaritmikus költségű RAM gép modellben** a egyes utasítások végrehajtásának költsége a benne szereplő természetes számok (rekeszcímek és tartalmak) kettes számrendszerbeli jegyeinek száma. A P RAM gép program **futási ideje** egy I inputra a P által I-re végrehajtott utasítások költségeinek összege.

RAM gép programjának futási ideje

A végrehajtott utasítások száma, nem a legjobb mérőszáma annak, hogy mennyit dolgozik a RAM gép.

Két nagyon nagy természetes szám összeadása ne legyen egységnyi művelet.

Definíció

A **logaritmikus költségű RAM gép modellben** a egyes utasítások végrehajtásának költsége a benne szereplő természetes számok (rekeszcímek és tartalmak) kettes számrendszerbeli jegyeinek száma. A P RAM gép program **futási ideje** egy I inputra a P által I-re végrehajtott utasítások költségeinek összege.

Néha e helyett két paraméterrel jellemezzük a futási időt, pl.

„a gép legfeljebb $O(f(n))$ lépést végez kettes számrendszerben legfeljebb $O(g(n))$ jegyű számokon".

RAM gép programjának futási ideje

A végrehajtott utasítások száma, nem a legjobb mérőszáma annak, hogy mennyit dolgozik a RAM gép.

Két nagyon nagy természetes szám összeadása ne legyen egységnyi művelet.

Definíció

A **logaritmikus költségű RAM gép modellben** a egyes utasítások végrehajtásának költsége a benne szereplő természetes számok (rekeszcímek és tartalmak) kettes számrendszerbeli jegyeinek száma. A P RAM gép program **futási ideje** egy I inputra a P által I-re végrehajtott utasítások költségeinek összege.

Néha e helyett két paraméterrel jellemezzük a futási időt, pl.

„a gép legfeljebb $O(f(n))$ lépést végez kettes számrendszerben legfeljebb $O(g(n))$ jegyű számokon".

Ekkor persze a gép $O(f(n)g(n))$ futási idejű.

Alternatív input fogalom

Feladat: Írjunk programot RAM gépre, mely egy N pozitív egészre meghatározza a legnagyobb olyan m-t melyre $2^m \leqslant N$(azaz $m = \lfloor \log_2 N \rfloor$). A program $O(m)$ lépést tegyen m jegyű számokkal.

Alternatív input fogalom

Feladat: Írjunk programot RAM gépre, mely egy N pozitív egészre meghatározza a legnagyobb olyan m-t melyre $2^m \leqslant N$(azaz $m = \lfloor \log_2 N \rfloor$). A program $O(m)$ lépést tegyen m jegyű számokkal.

Megoldás: A bemenet az $x[0]$ rekeszben van, minden más rekesz tartalma 0, a kimenet az $x[1]$ rekeszben lesz.

Alternatív input fogalom

Feladat: Írjunk programot RAM gépre, mely egy N pozitív egészre meghatározza a legnagyobb olyan m-t melyre $2^m \leqslant N$(azaz $m = \lfloor \log_2 N \rfloor$). A program $O(m)$ lépést tegyen m jegyű számokkal.

Megoldás: A bemenet az $x[0]$ rekeszben van, minden más rekesz tartalma 0, a kimenet az $x[1]$ rekeszben lesz.

0 $x[2] := x[2] + 1$;

1 $x[3] := x[2]$;

2 $x[2] := x[2] + x[3]$;

3 $x[4] := x[0] - x[2]$;

4 $x[4] := x[4] + 1$;

5 IF $x[4] \leqslant 0$ THEN GOTO 8;

6 $x[1] := x[1] + 1$;

7 IF $x[5] \leqslant 0$ THEN GOTO 1;

8

Alternatív input fogalom

Állítás: Kiszámítható $O(\log N)$ lépésben, $O(\log N)$ jegyű számokkal N kettes számrendszerbeli alakja.

Alternatív input fogalom

Állítás: Kiszámítható $O(\log N)$ lépésben, $O(\log N)$ jegyű számokkal N kettes számrendszerbeli alakja.

Ötlet: Az előző feladat alapján állítsuk elő $m = \lfloor \log_2 N \rfloor$-t. Majd $0, \ldots m$-re állítsuk elő a 2-hatványokat $m + 1$ egymást követő rekeszben (2 utasítás kettőhatványonként). Majd N-ből a legnagyobbtól a legkisebb felé haladva kivonva sorra a 2-hatványokat a rekeszek feltölthetők a 2-es számrendszerbeli alak bitjeivel.

Alternatív input fogalom

Állítás: Kiszámítható $O(\log N)$ lépésben, $O(\log N)$ jegyű számokkal N kettes számrendszerbeli alakja.

Ötlet: Az előző feladat alapján állítsuk elő $m = \lfloor \log_2 N \rfloor$-t. Majd $0, \ldots m$-re állítsuk elő a 2-hatványokat $m+1$ egymást követő rekeszben (2 utasítás kettőhatványonként). Majd N-ből a legnagyobbtól a legkisebb felé haladva kivonva sorra a 2-hatványokat a rekeszek feltölthetők a 2-es számrendszerbeli alak bitjeivel.

Alternatív input fogalom: Az input mindig egyetlen, de méretében nem korlátozott N természetes szám az $x[0]$ rekeszben. A fentiek alapján hatékonyan, azaz $O(\log N)$ lépésben, $O(\log N)$ jegyű számokkal átkonvertálható az eredeti alakra.

Turing ekvivalencia

Tétel

Ha egy függvény $f(n)$ időkorlátos TG-pel kiszámítható, akkor konstruálható $O(f(n))$ lépésszámú $O(\log f(n))$ jegyű számokkal dolgozó program RAM gépre ami ugyanazt a függvényt számítja ki.

Turing ekvivalencia

Tétel

Ha egy függvény $f(n)$ időkorlátos TG-pel kiszámítható, akkor konstruálható $O(f(n))$ lépésszámú $O(\log f(n))$ jegyű számokkal dolgozó program RAM gépre ami ugyanazt a függvényt számítja ki.

Bizonyítás: Mivel a TG egy függvényt számít ki, feltehetjük, hogy a TG-nek egyetlen megállási állapota van.

Turing ekvivalencia

Tétel

Ha egy függvény $f(n)$ időkorlátos TG-pel kiszámítható, akkor konstruálható $O(f(n))$ lépésszámú $O(\log f(n))$ jegyű számokkal dolgozó program RAM gépre ami ugyanazt a függvényt számítja ki.

Bizonyítás: Mivel a TG egy függvényt számít ki, feltehetjük, hogy a TG-nek egyetlen megállási állapota van.
Legyen $M = \langle Q, \Sigma, \Gamma, \delta, q_1, q_r \rangle$ TG, ahol $r \geqslant 1$, $Q = \{q_1, \ldots, q_r\}$, $\Gamma = \{0, 1, \ldots |\Gamma| - 1\}$, q_r a megállási, q_1 a kezdőállapot.

Turing ekvivalencia

Tétel

Ha egy függvény $f(n)$ időkorlátos TG-pel kiszámítható, akkor konstruálható $O(f(n))$ lépésszámú $O(\log f(n))$ jegyű számokkal dolgozó program RAM gépre ami ugyanazt a függvényt számítja ki.

Bizonyítás: Mivel a TG egy függvényt számít ki, feltehetjük, hogy a TG-nek egyetlen megállási állapota van.
Legyen $M = \langle Q, \Sigma, \Gamma, \delta, q_1, q_r \rangle$ TG, ahol $r \geqslant 1$, $Q = \{q_1, \ldots, q_r\}$, $\Gamma = \{0, 1, \ldots |\Gamma| - 1\}$, q_r a megállási, q_1 a kezdőállapot.

A Turing gép számolásának szimulálása során a RAM gép $2i$-edik memóriarekeszében ugyanaz a Γ-beli szám fog állni, mint a Turing gép szalagjának i-edik cellájában.

Turing ekvivalencia

Tétel

Ha egy függvény $f(n)$ időkorlátos TG-pel kiszámítható, akkor konstruálható $O(f(n))$ lépésszámú $O(\log f(n))$ jegyű számokkal dolgozó program RAM gépre ami ugyanazt a függvényt számítja ki.

Bizonyítás: Mivel a TG egy függvényt számít ki, feltehetjük, hogy a TG-nek egyetlen megállási állapota van.
Legyen $M = \langle Q, \Sigma, \Gamma, \delta, q_1, q_r \rangle$ TG, ahol $r \geqslant 1$, $Q = \{q_1, \ldots, q_r\}$, $\Gamma = \{0, 1, \ldots |\Gamma| - 1\}$, q_r a megállási, q_1 a kezdőállapot.

A Turing gép számolásának szimulálása során a RAM gép $2i$-edik memóriarekeszében ugyanaz a Γ-beli szám fog állni, mint a Turing gép szalagjának i-edik cellájában. Feltehető, hogy az input M nulladik cellájánál kezdődik, így egy n hosszú input első betűje a RAM gép $x[0]$ rekeszében, utolsó betűje $x[2n-2]$-ben található.

Turing ekvivalencia

Tétel

Ha egy függvény $f(n)$ időkorlátos TG-pel kiszámítható, akkor konstruálható $O(f(n))$ lépésszámú $O(\log f(n))$ jegyű számokkal dolgozó program RAM gépre ami ugyanazt a függvényt számítja ki.

Bizonyítás: Mivel a TG egy függvényt számít ki, feltehetjük, hogy a TG-nek egyetlen megállási állapota van.
Legyen $M = \langle Q, \Sigma, \Gamma, \delta, q_1, q_r \rangle$ TG, ahol $r \geqslant 1$, $Q = \{q_1, \ldots, q_r\}$, $\Gamma = \{0, 1, \ldots |\Gamma| - 1\}$, q_r a megállási, q_1 a kezdőállapot.

A Turing gép számolásának szimulálása során a RAM gép $2i$-edik memóriarekeszében ugyanaz a Γ-beli szám fog állni, mint a Turing gép szalagjának i-edik cellájában. Feltehető, hogy az input M nulladik cellájánál kezdődik, így egy n hosszú input első betűje a RAM gép $x[0]$ rekeszében, utolsó betűje $x[2n-2]$-ben található.

A kis, páratlan indexű rekeszeket használjuk minden másra. $x[1]$ tartalma a működés során mindig a fej helyzetének megfelelő cella sorszáma.

Turing ekvivalencia

Programunk $P_i\,(1 \leqslant i \leqslant r)$ és $Q_{i,j}\,(1 \leqslant i \leqslant r-1, 0 \leqslant j \leqslant |\Gamma|-1)$ részekből fog állni.

Turing ekvivalencia

Programunk $P_i\,(1 \leqslant i \leqslant r)$ és $Q_{i,j}\,(1 \leqslant i \leqslant r-1, 0 \leqslant j \leqslant |\Gamma|-1)$ részekből fog állni. A P_i programrész $1 \leqslant i \leqslant r-1$ akkor kerül meghívásra, ha a Turing gép vezérlőegysége egy átmenet után az i-edik állapotába kerül, ekkor a RAM program a szalagon olvasott betű (j) függvényében elágazik a $Q_{i,j}$ programrészekre.

Turing ekvivalencia

Programunk $P_i\,(1 \leqslant i \leqslant r)$ és $Q_{i,j}\,(1 \leqslant i \leqslant r-1, 0 \leqslant j \leqslant |\Gamma|-1)$ részekből fog állni. A P_i programrész $1 \leqslant i \leqslant r-1$ akkor kerül meghívásra, ha a Turing gép vezérlőegysége egy átmenet után az i-edik állapotába kerül, ekkor a RAM program a szalagon olvasott betű (j) függvényében elágazik a $Q_{i,j}$ programrészekre.

A P_i programrész $(1 \leqslant i \leqslant r-1)$:

$x[3] := x[x[1]]$;
IF $x[3] \leqslant 0$ THEN GOTO [$Q_{i,0}$ címe];
$x[3] := x[3] - 1$;
IF $x[3] \leqslant 0$ THEN GOTO [$Q_{i,1}$ címe];
$\vdots$
$x[3] := x[3] - 1$;
IF $x[3] \leqslant 0$ THEN GOTO [$Q_{i,|\Gamma|-1}$ címe];

A P_r programrész álljon egyetlen üres programsorból.

Turing ekvivalencia

Programunk $P_i\,(1 \leqslant i \leqslant r)$ és $Q_{i,j}\,(1 \leqslant i \leqslant r-1, 0 \leqslant j \leqslant |\Gamma|-1)$ részekből fog állni. A P_i programrész $1 \leqslant i \leqslant r-1$ akkor kerül meghívásra, ha a Turing gép vezérlőegysége egy átmenet után az i-edik állapotába kerül, ekkor a RAM program a szalagon olvasott betű (j) függvényében elágazik a $Q_{i,j}$ programrészekre.

A P_i programrész $(1 \leqslant i \leqslant r-1)$:

$x[3] := x[x[1]]$;
IF $x[3] \leqslant 0$ THEN GOTO [$Q_{i,0}$ címe];
$x[3] := x[3] - 1$;
IF $x[3] \leqslant 0$ THEN GOTO [$Q_{i,1}$ címe];
$\vdots$
$x[3] := x[3] - 1$;
IF $x[3] \leqslant 0$ THEN GOTO [$Q_{i,|\Gamma|-1}$ címe];

A P_r programrész álljon egyetlen üres programsorból.

Turing ekvivalencia

$Q_{i,j}$ programrész akkor kerül meghívásra, ha az i-edik állapotban j-t olvas a fej. Ekkor a TG átírja az $x[1]$-edik rekeszt $\delta(q_i, j) = (q_{\alpha(i,j)}, \beta(i,j), \gamma(i,j))$ szerint $\beta(i,j)$-re, módosítja $x[1]$-et $\gamma(i,j)$ szerint, és az új, $q_{\alpha(i,j)}$ állapotának megfelelő $P_{\alpha(i,j)}$ programrészre ugrik.

Turing ekvivalencia

$Q_{i,j}$ programrész akkor kerül meghívásra, ha az i-edik állapotban j-t olvas a fej. Ekkor a TG átírja az $x[1]$-edik rekeszt $\delta(q_i, j) = (q_{\alpha(i,j)}, \beta(i,j), \gamma(i,j))$ szerint $\beta(i,j)$-re, módosítja $x[1]$-et $\gamma(i,j)$ szerint, és az új, $q_{\alpha(i,j)}$ állapotának megfelelő $P_{\alpha(i,j)}$ programrészre ugrik. A $Q_{i,j}$ programrész:

$$
\begin{array}{ll}
x[3] := 0; & \\
\left.\begin{array}{l} x[3] := x[3] + 1; \\ \vdots \\ x[3] := x[3] + 1; \end{array}\right\} & \beta(i,j) \text{ -szer} \\
x[x[1]] := x[3]; & \\
x[1] := x[1] + 1; & \text{ha } \gamma(i,j) = R \\
x[1] := x[1] + 1; & \text{ha } \gamma(i,j) = R \\
x[1] := x[1] - 1; & \text{ha } \gamma(i,j) = L \\
x[1] := x[1] - 1; & \text{ha } \gamma(i,j) = L \\
x[3] := 0; & \\
\text{IF } x[3] \leqslant 0 \text{ THEN GOTO } [P_{\alpha(i,j)} \text{ címe}]; &
\end{array}
$$

Turing ekvivalencia

A főprogram:

$$\boxed{\begin{array}{l} x[1] := 0; \\ P_1 \\ \vdots \\ P_r \\ Q_{0,0} \\ \vdots \\ Q_{r-1,|\Gamma|-1} \end{array}}$$

Ezzel a Turing gép szimulálását befejeztük.

A futási időre vonatkozó állítás abból következik, hogy a Turing gép minden lépésének szimulálása legfeljebb $3|\Gamma| + 6 = O(1)$ RAM gép utasítással történik, így a RAM gép $O(f(n))$ utasítást hajt végre. $f(n)$ lépésben a Turing gép legfeljebb egy $-f(n)$ és $f(n)$ közötti sorszámú cellába írhat bármit is. Így a program egyes lépéseiben $O(\log f(n))$ hosszúságú számokkal dolgozunk. $\square$

Turing ekvivalencia

Tétel

Minden RAM gépre írt programhoz van olyan Turing gép, amely minden bemenetre ugyanazt a kimenetet számítja ki, mint a RAM gép. Ha a RAM-gép futási ideje $f(n)$, akkor a Turing gép $O(f(n)^2)$ időkorlátos.

Turing ekvivalencia

Tétel

Minden RAM gépre írt programhoz van olyan Turing gép, amely minden bemenetre ugyanazt a kimenetet számítja ki, mint a RAM gép. Ha a RAM-gép futási ideje $f(n)$, akkor a Turing gép $O(f(n)^2)$ időkorlátos.

Bizonyítás: A RAM-gép számolását egy négyszalagos Turing géppel fogjuk szimulálni.

Turing ekvivalencia

Tétel

Minden RAM gépre írt programhoz van olyan Turing gép, amely minden bemenetre ugyanazt a kimenetet számítja ki, mint a RAM gép. Ha a RAM-gép futási ideje $f(n)$, akkor a Turing gép $O(f(n)^2)$ időkorlátos.

Bizonyítás: A RAM-gép számolását egy négyszalagos Turing géppel fogjuk szimulálni.

Alapötlet: A Turing gép egyik szalagjára írjuk fel sorban az $x[i]$ memóriarekeszek tartalmát (kettes számrendszerben, ha negatív, előjellel ellátva, $\#$ jelekkel elválasztva), minden rekesz tartalmát sorra feltüntetve.

Turing ekvivalencia

Gond: Ha a RAM-gép a 2^x sorszámú rekeszbe ír, akkor ez a logaritmikus költség modell szerint x költséget jelent.

Turing ekvivalencia

Gond: Ha a RAM-gép a 2^x sorszámú rekeszbe ír, akkor ez a logaritmikus költség modell szerint x költséget jelent. Ilyenkor a Turing gépnek a 2^x-edik (vagy akár még nagyobb sorszámú) cellájába kell írnia, és ha a fej aktuális helyzete távol esik ettől a cellától, akkor az akár $\Omega(2^x)$ lépést (és így költséget) is jelenthet a Turing gép számára.

Turing ekvivalencia

Gond: Ha a RAM-gép a 2^x sorszámú rekeszbe ír, akkor ez a logaritmikus költség modell szerint x költséget jelent. Ilyenkor a Turing gépnek a 2^x-edik (vagy akár még nagyobb sorszámú) cellájába kell írnia, és ha a fej aktuális helyzete távol esik ettől a cellától, akkor az akár $\Omega(2^x)$ lépést (és így költséget) is jelenthet a Turing gép számára.

Ilyen költséges reprezentációt nem engedhetünk meg magunknak, mert ez összességében akár a futási idő exponenciális növekedését eredményezheti.

Turing ekvivalencia

Gond: Ha a RAM-gép a 2^x sorszámú rekeszbe ír, akkor ez a logaritmikus költség modell szerint x költséget jelent. Ilyenkor a Turing gépnek a 2^x-edik (vagy akár még nagyobb sorszámú) cellájába kell írnia, és ha a fej aktuális helyzete távol esik ettől a cellától, akkor az akár $\Omega(2^x)$ lépést (és így költséget) is jelenthet a Turing gép számára.

Ilyen költséges reprezentációt nem engedhetünk meg magunknak, mert ez összességében akár a futási idő exponenciális növekedését eredményezheti.

Vegyük azonban észre, hogy túl nagy indexű rekeszek használata $f(n)$ összköltség esetén csak úgy lehetséges, ha a kisebb indexű rekeszek túlnyomó többségének a tartalma 0 marad az egész számítás folyamán.

Turing ekvivalencia

Javítás: Csak azoknak a rekeszeknek a tartalmát tároljuk a Turing gép szalagján, melyekbe ténylegesen ír a RAM-gép. Ekkor azt is fel kell tüntetni, hogy mi a szóban forgó rekesz sorszáma.

Turing ekvivalencia

Javítás: Csak azoknak a rekeszeknek a tartalmát tároljuk a Turing gép szalagján, melyekbe ténylegesen ír a RAM-gép. Ekkor azt is fel kell tüntetni, hogy mi a szóban forgó rekesz sorszáma.

Azt tesszük tehát, hogy valahányszor a RAM-gép egy $x[z]$ rekeszbe egy y számot ír, a Turing gép ezt úgy szimulálja, hogy az első szalagja végére a $\#\#y\#z$ jelsorozatot írja, y-t és z-t kettes számrendszerben. (Korábbi tartalmat sosem ír felül, csak hozzáadja ezt az információt a szalag tartalmához!)

Turing ekvivalencia

Javítás: Csak azoknak a rekeszeknek a tartalmát tároljuk a Turing gép szalagján, melyekbe ténylegesen ír a RAM-gép. Ekkor azt is fel kell tüntetni, hogy mi a szóban forgó rekesz sorszáma.

Azt tesszük tehát, hogy valahányszor a RAM-gép egy $x[z]$ rekeszbe egy y számot ír, a Turing gép ezt úgy szimulálja, hogy az első szalagja végére a $\#\#y\#z$ jelsorozatot írja, y-t és z-t kettes számrendszerben. (Korábbi tartalmat sosem ír felül, csak hozzáadja ezt az információt a szalag tartalmához!)

$\#\#y\#z$ hossza így 3-mal hosszabb, mint a logaritmikus költség modellben a RAM gép értékadásának a költsége. Mivel y és z leírásához legalább 1 bit kell, így legfeljebb 5/2-szerese a hossznövekmény a szimulált RAM gép utasítás logaritmikus költségének.

Turing ekvivalencia

Ha a RAM-gép egy $x[z]$ rekesz tartalmát olvassa ki, akkor a Turing gép első szalagján a fej hátulról indulva megkeresi az első $\#\#u\#z$ alakú részszót; ez az u érték a rekesz tartalma, hiszen ez volt utoljára a z-edik rekeszbe írva.

Turing ekvivalencia

Ha a RAM-gép egy $x[z]$ rekesz tartalmát olvassa ki, akkor a Turing gép első szalagján a fej hátulról indulva megkeresi az első $\#\#u\#z$ alakú részszót; ez az u érték a rekesz tartalma, hiszen ez volt utoljára a z-edik rekeszbe írva.

Ha ilyen alakú részszót nem talál, akkor $x[z]$-t 0-nak tekinthetjük.

Turing ekvivalencia

Ha a RAM-gép egy $x[z]$ rekesz tartalmát olvassa ki, akkor a Turing gép első szalagján a fej hátulról indulva megkeresi az első $\#\#u\#z$ alakú részszót; ez az u érték a rekesz tartalma, hiszen ez volt utoljára a z-edik rekeszbe írva.

Ha ilyen alakú részszót nem talál, akkor $x[z]$-t 0-nak tekinthetjük.

A RAM-gép programnyelvének minden egyes utasítását könnyű szimulálni egy-egy alkalmas Turing géppel, amely csak a másik három szalagot használja.

Turing ekvivalencia

Ha a RAM-gép egy $x[z]$ rekesz tartalmát olvassa ki, akkor a Turing gép első szalagján a fej hátulról indulva megkeresi az első $\#\#u\#z$ alakú részszót; ez az u érték a rekesz tartalma, hiszen ez volt utoljára a z-edik rekeszbe írva.

Ha ilyen alakú részszót nem talál, akkor $x[z]$-t 0-nak tekinthetjük.

A RAM-gép programnyelvének minden egyes utasítását könnyű szimulálni egy-egy alkalmas Turing géppel, amely csak a másik három szalagot használja.

A Turing gép egy olyan „szupergép” lesz, amelyben minden programsornak megfelel egy rész-Turing gép, és az ehhez tartozó állapotok egy halmaza.

Turing ekvivalencia

Ha a RAM-gép egy $x[z]$ rekesz tartalmát olvassa ki, akkor a Turing gép első szalagján a fej hátulról indulva megkeresi az első $\#\#u\#z$ alakú részszót; ez az u érték a rekesz tartalma, hiszen ez volt utoljára a z-edik rekeszbe írva.

Ha ilyen alakú részszót nem talál, akkor $x[z]$-t 0-nak tekinthetjük.

A RAM-gép programnyelvének minden egyes utasítását könnyű szimulálni egy-egy alkalmas Turing géppel, amely csak a másik három szalagot használja.

A Turing gép egy olyan „szupergép" lesz, amelyben minden programsornak megfelel egy rész-Turing gép, és az ehhez tartozó állapotok egy halmaza.

Ez a rész-Turing gép az illető utasítást végrehajtja, az első szalag fejét a végére (utolsó nemüres cellájára) állítja, a többi szalagot üresen adja vissza.

Turing ekvivalencia

Minden ilyen Turing gép végállapota azonosítva van a következő sornak megfelelő Turing gép kezdőállapotával. (A feltételes ugrás esetén, ha $x[i] \leqslant 0$ teljesül, a p sornak megfelelő Turing gép kezdőállapotába megy át a „szupergép”.)

Turing ekvivalencia

Minden ilyen Turing gép végállapota azonosítva van a következő sornak megfelelő Turing gép kezdőállapotával. (A feltételes ugrás esetén, ha $x[i] \leqslant 0$ teljesül, a p sornak megfelelő Turing gép kezdőállapotába megy át a „szupergép".)

A 0-dik programsornak megfelelő Turing gép kezdőállapota lesz a szupergép kezdőállapota is.

Turing ekvivalencia

Minden ilyen Turing gép végállapota azonosítva van a következő sornak megfelelő Turing gép kezdőállapotával. (A feltételes ugrás esetén, ha $x[i] \leqslant 0$ teljesül, a p sornak megfelelő Turing gép kezdőállapotába megy át a „szupergép”.)

A 0-dik programsornak megfelelő Turing gép kezdőállapota lesz a szupergép kezdőállapota is.

Ezenkívül lesz még egy végállapot; ez felel meg minden üres programsornak.

Turing ekvivalencia

Minden ilyen Turing gép végállapota azonosítva van a következő sornak megfelelő Turing gép kezdőállapotával. (A feltételes ugrás esetén, ha $x[i] \leqslant 0$ teljesül, a p sornak megfelelő Turing gép kezdőállapotába megy át a „szupergép”.)

A 0-dik programsornak megfelelő Turing gép kezdőállapota lesz a szupergép kezdőállapota is.

Ezenkívül lesz még egy végállapot; ez felel meg minden üres programsornak.

Belátható, hogy az így megkonstruált Turing gép lépésről lépésre szimulálja a RAM-gép működését.

Turing ekvivalencia

A legtöbb programsort a Turing gép a benne szereplő számok számjegyeinek számával, vagyis éppen a RAM-gépen erre fordított idejével arányos lépésszámban hajtja végre.

Turing ekvivalencia

A legtöbb programsort a Turing gép a benne szereplő számok számjegyeinek számával, vagyis éppen a RAM-gépen erre fordított idejével arányos lépésszámban hajtja végre.

Kivétel egy $x[i]$ érték kiolvasása, amelyhez esetleg (egy lépésnél legfeljebb kétszer) végig kell keresni az egész szalagot.

Turing ekvivalencia

A legtöbb programsort a Turing gép a benne szereplő számok számjegyeinek számával, vagyis éppen a RAM-gépen erre fordított idejével arányos lépésszámban hajtja végre.

Kivétel egy $x[i]$ érték kiolvasása, amelyhez esetleg (egy lépésnél legfeljebb kétszer) végig kell keresni az egész szalagot.

Mivel az első szalag tartalmának a hossza egy K költségű RAM gép utasítás esetén $O(K)$-val nő lépésenként, ezért $f(n)$ lépés alatt $O(f(n))$-re duzzadhat. Így a TG futási ideje $f(n)O(f(n)) = O(f(n)^2)$. $\square$

PRAM gép

PRAM gép: párhuzamos RAM gép – párhuzamos számítást végez. Ez $p \geqslant 1$ azonos RAM-gépből (processzorból) áll.

PRAM gép

PRAM gép: párhuzamos RAM gép – párhuzamos számítást végez. Ez $p \geqslant 1$ azonos RAM-gépből (processzorból) áll.

A gépek programtára közös, és van egy közös memóriaterületük is, amely mondjuk az $x[i]$ rekeszekből áll (ahol i az egész számokon fut végig).

PRAM gép

PRAM gép: párhuzamos RAM gép – párhuzamos számítást végez. Ez $p \geqslant 1$ azonos RAM-gépből (processzorból) áll.

A gépek programtára közös, és van egy közös memóriaterületük is, amely mondjuk az $x[i]$ rekeszekből áll (ahol i az egész számokon fut végig).

Feltehető, hogy minden processzornak van még végtelen sok $u[i]$ ($i \in \mathbb{Z}$) saját memóriarekesze.

PRAM gép

PRAM gép: párhuzamos RAM gép – párhuzamos számítást végez. Ez $p \geqslant 1$ azonos RAM-gépből (processzorból) áll.

A gépek programtára közös, és van egy közös memóriaterületük is, amely mondjuk az $x[i]$ rekeszekből áll (ahol i az egész számokon fut végig).

Feltehető, hogy minden processzornak van még végtelen sok $u[i]$ ($i \in \mathbb{Z}$) saját memóriarekesze.

Induláskor minden processzor $u[0]$ rekeszében a processzor saját sorszáma áll.

PRAM gép

PRAM gép: párhuzamos RAM gép – párhuzamos számítást végez. Ez $p \geqslant 1$ azonos RAM-gépből (processzorból) áll.

A gépek programtára közös, és van egy közös memóriaterületük is, amely mondjuk az $x[i]$ rekeszekből áll (ahol i az egész számokon fut végig).

Feltehető, hogy minden processzornak van még végtelen sok $u[i]$ $(i \in \mathbb{Z})$ saját memóriarekesze.

Induláskor minden processzor $u[0]$ rekeszében a processzor saját sorszáma áll.

A processzor írhat és olvashat a saját $u[i]$ rekeszeibe, ill. rekeszeiből és a közös $x[i]$ memóriarekeszekbe, ill. memóriarekeszekből.

PRAM gép

A bemenetet az $x[0], x[1], \ldots, x[n]$ rekeszekbe írjuk, ahol $x[0] = n$ a bemenet hossza.

PRAM gép

A bemenetet az $x[0], x[1], \ldots, x[n]$ rekeszekbe írjuk, ahol $x[0] = n$ a bemenet hossza.

A bemenet és a (közös) program mellé meg kell adni azt is, hogy hány processzorral dolgozunk; ezt írhatjuk pl. az $x[-1]$ rekeszbe.

PRAM gép

A bemenetet az $x[0], x[1], \dots, x[n]$ rekeszekbe írjuk, ahol $x[0] = n$ a bemenet hossza.

A bemenet és a (közös) program mellé meg kell adni azt is, hogy hány processzorral dolgozunk; ezt írhatjuk pl. az $x[-1]$ rekeszbe.

A processzorok párhuzamosan, de ütemre hajtják végre a programot.

PRAM gép

A bemenetet az $x[0], x[1], \dots, x[n]$ rekeszekbe írjuk, ahol $x[0] = n$ a bemenet hossza.

A bemenet és a (közös) program mellé meg kell adni azt is, hogy hány processzorral dolgozunk; ezt írhatjuk pl. az $x[-1]$ rekeszbe.

A processzorok párhuzamosan, de ütemre hajtják végre a programot.

További utasítások az alapmodell utasításain felül:

- $u[i] := x[u[j]];$ $\qquad (i, j \in \mathbb{Z})$

PRAM gép

A bemenetet az $x[0], x[1], \ldots, x[n]$ rekeszekbe írjuk, ahol $x[0] = n$ a bemenet hossza.

A bemenet és a (közös) program mellé meg kell adni azt is, hogy hány processzorral dolgozunk; ezt írhatjuk pl. az $x[-1]$ rekeszbe.

A processzorok párhuzamosan, de ütemre hajtják végre a programot.

További utasítások az alapmodell utasításain felül:

- $u[i] := x[u[j]];$ $\quad (i, j \in \mathbb{Z})$
- $x[u[i]] := u[j];$ $\quad (i, j \in \mathbb{Z})$

PRAM gép

A bemenetet az $x[0], x[1], \ldots, x[n]$ rekeszekbe írjuk, ahol $x[0] = n$ a bemenet hossza.

A bemenet és a (közös) program mellé meg kell adni azt is, hogy hány processzorral dolgozunk; ezt írhatjuk pl. az $x[-1]$ rekeszbe.

A processzorok párhuzamosan, de ütemre hajtják végre a programot.

További utasítások az alapmodell utasításain felül:

- $u[i] := x[u[j]];$ $\quad (i, j \in \mathbb{Z})$
- $x[u[i]] := u[j];$ $\quad (i, j \in \mathbb{Z})$
- $u[i] := u[i] \cdot u[j];$ $\quad (i, j \in \mathbb{Z})$

PRAM gép

A bemenetet az $x[0], x[1], \ldots, x[n]$ rekeszekbe írjuk, ahol $x[0] = n$ a bemenet hossza.

A bemenet és a (közös) program mellé meg kell adni azt is, hogy hány processzorral dolgozunk; ezt írhatjuk pl. az $x[-1]$ rekeszbe.

A processzorok párhuzamosan, de ütemre hajtják végre a programot.

További utasítások az alapmodell utasításain felül:

- $u[i] := x[u[j]];$ $\quad (i, j \in \mathbb{Z})$
- $x[u[i]] := u[j];$ $\quad (i, j \in \mathbb{Z})$
- $u[i] := u[i] \cdot u[j];$ $\quad (i, j \in \mathbb{Z})$
- $u[i] := u[i] : u[j];$ $\quad (i, j \in \mathbb{Z})$

PRAM gép

A bemenetet az $x[0], x[1], \ldots, x[n]$ rekeszekbe írjuk, ahol $x[0] = n$ a bemenet hossza.

A bemenet és a (közös) program mellé meg kell adni azt is, hogy hány processzorral dolgozunk; ezt írhatjuk pl. az $x[-1]$ rekeszbe.

A processzorok párhuzamosan, de ütemre hajtják végre a programot.

További utasítások az alapmodell utasításain felül:

- $u[i] := x[u[j]];$ $\quad (i, j \in \mathbb{Z})$
- $x[u[i]] := u[j];$ $\quad (i, j \in \mathbb{Z})$
- $u[i] := u[i] \cdot u[j];$ $\quad (i, j \in \mathbb{Z})$
- $u[i] := u[i] : u[j];$ $\quad (i, j \in \mathbb{Z})$

PRAM gép

A bemenetet az $x[0], x[1], \ldots, x[n]$ rekeszekbe írjuk, ahol $x[0] = n$ a bemenet hossza.

A bemenet és a (közös) program mellé meg kell adni azt is, hogy hány processzorral dolgozunk; ezt írhatjuk pl. az $x[-1]$ rekeszbe.

A processzorok párhuzamosan, de ütemre hajtják végre a programot.

További utasítások az alapmodell utasításain felül:

- $u[i] := x[u[j]];$ $\qquad (i, j \in \mathbb{Z})$
- $x[u[i]] := u[j];$ $\qquad (i, j \in \mathbb{Z})$
- $u[i] := u[i] \cdot u[j];$ $\qquad (i, j \in \mathbb{Z})$
- $u[i] := u[i] : u[j];$ $\qquad (i, j \in \mathbb{Z})$

Az utóbbi a maradékos osztás, $a : b$ eredménye a legnagyobb c természetes szám, amelyre $|a| \geqslant |b|c$.

PRAM gép

A szorzást és maradékos osztást azért célszerű hozzávenni, hogy egy processzor a saját sorszámából mindig ki tudja 1 lépésben számolni, hogy melyik $x[f(x[-1], x[0], u[0])]$ inputcellát kell előszörre olvasnia, legalábbis elég egyszerű f függvény esetén.

Példa: $f(x[-1], x[0], u[0]) = (u[0] : x[0]) (\mathrm{mod}\ x[-1])$.

PRAM gép

Logaritmikus költségfüggvényt használunk: pl. egy k egész számnak az $x[t]$ memóriarekeszbe való beírásának vagy kiolvasásának a költsége $\log|k| + \log|t|$.

PRAM gép

Logaritmikus költségfüggvényt használunk: pl. egy k egész számnak az $x[t]$ memóriarekeszbe való beírásának vagy kiolvasásának a költsége $\log|k| + \log|t|$.

A szorzásnál és maradékos osztásnál ehhez még hozzáadjuk a két operandus bitszámának szorzatát.

PRAM gép

Logaritmikus költségfüggvényt használunk: pl. egy k egész számnak az $x[t]$ memóriarekeszbe való beírásának vagy kiolvasásának a költsége $\log|k| + \log|t|$.

A szorzásnál és maradékos osztásnál ehhez még hozzáadjuk a két operandus bitszámának szorzatát.

A következő ütem akkor kezdődik, amikor az előző műveletet minden processzor végrehajtotta.

PRAM gép

Logaritmikus költségfüggvényt használunk: pl. egy k egész számnak az $x[t]$ memóriarekeszbe való beírásának vagy kiolvasásának a költsége $\log|k| + \log|t|$.

A szorzásnál és maradékos osztásnál ehhez még hozzáadjuk a két operandus bitszámának szorzatát.

A következő ütem akkor kezdődik, amikor az előző műveletet minden processzor végrehajtotta.

Egy ütem idejét tehát a legtöbb időt igénybevevő processzor határozza meg.

PRAM gép

Logaritmikus költségfüggvényt használunk: pl. egy k egész számnak az $x[t]$ memóriarekeszbe való beírásának vagy kiolvasásának a költsége $\log|k| + \log|t|$.

A szorzásnál és maradékos osztásnál ehhez még hozzáadjuk a két operandus bitszámának szorzatát.

A következő ütem akkor kezdődik, amikor az előző műveletet minden processzor végrehajtotta.

Egy ütem idejét tehát a legtöbb időt igénybevevő processzor határozza meg.

A gép akkor áll meg, ha minden processzor olyan programsorhoz ér, amelyben nincsen utasítás.

PRAM gép

Logaritmikus költségfüggvényt használunk: pl. egy k egész számnak az $x[t]$ memóriarekeszbe való beírásának vagy kiolvasásának a költsége $\log|k| + \log|t|$.

A szorzásnál és maradékos osztásnál ehhez még hozzáadjuk a két operandus bitszámának szorzatát.

A következő ütem akkor kezdődik, amikor az előző műveletet minden processzor végrehajtotta.

Egy ütem idejét tehát a legtöbb időt igénybevevő processzor határozza meg.

A gép akkor áll meg, ha minden processzor olyan programsorhoz ér, amelyben nincsen utasítás.

A kimenet az $x[i]$ rekeszek tartalma.

PRAM gép konfliktuskezelés

Konvenciók az I/O konfliktusok kezelésére:

PRAM gép konfliktuskezelés

Konvenciók az I/O konfliktusok kezelésére:

EREW (Exclusive Read Exclusive Write) – Teljesen konfliktusmentes modell

PRAM gép konfliktuskezelés

Konvenciók az I/O konfliktusok kezelésére:

EREW (Exclusive Read Exclusive Write) – Teljesen konfliktusmentes modell

Nem szabad két processzornak ugyanabból a rekeszből olvasni vagy ugyanabba a rekeszbe írni. A programozónak kell gondoskodnia arról, hogy ez ne következzen be; ha mégis megtörténne, a gép programhibát jelez.

PRAM gép konfliktuskezelés

Konvenciók az I/O konfliktusok kezelésére:

EREW (Exclusive Read Exclusive Write) – Teljesen konfliktusmentes modell

Nem szabad két processzornak ugyanabból a rekeszből olvasni vagy ugyanabba a rekeszbe írni. A programozónak kell gondoskodnia arról, hogy ez ne következzen be; ha mégis megtörténne, a gép programhibát jelez.

CREW (Concurrent Read Exclusive Write) – Félig konfliktusmentes modell

PRAM gép konfliktuskezelés

Konvenciók az I/O konfliktusok kezelésére:

EREW (Exclusive Read Exclusive Write) – Teljesen konfliktusmentes modell

Nem szabad két processzornak ugyanabból a rekeszből olvasni vagy ugyanabba a rekeszbe írni. A programozónak kell gondoskodnia arról, hogy ez ne következzen be; ha mégis megtörténne, a gép programhibát jelez.

CREW (Concurrent Read Exclusive Write) – Félig konfliktusmentes modell

Talán legtermészetesebb az a modell, melyben megengedjük, hogy több processzor is ugyanazt a rekeszt olvassa, de ha ugyanoda akarnak írni, az már programhiba.

PRAM gép konfliktuskezelés

CRCW (Concurrent Read Concurrent Write) –
Konfliktus-korlátozó modell

PRAM gép konfliktuskezelés

CRCW (Concurrent Read Concurrent Write) – Konfliktus-korlátozó modell

Szabad több processzornak ugyanabból a rekeszből olvasni és ugyanabba a rekeszbe írni is, de csak akkor, ha ugyanazt akarják beírni. A gép akkor áll le programhibával, ha két processzor ugyanabba a rekeszbe más-más számot akar beírni.

PRAM gép konfliktuskezelés

CRCW (Concurrent Read Concurrent Write) – Konfliktus-korlátozó modell

Szabad több processzornak ugyanabból a rekeszből olvasni és ugyanabba a rekeszbe írni is, de csak akkor, ha ugyanazt akarják beírni. A gép akkor áll le programhibával, ha két processzor ugyanabba a rekeszbe más-más számot akar beírni.

P-CRCW (Priority Concurrent Read Concurrent Write) – Elsőbbségi modell

PRAM gép konfliktuskezelés

CRCW (Concurrent Read Concurrent Write) – Konfliktus-korlátozó modell

Szabad több processzornak ugyanabból a rekeszből olvasni és ugyanabba a rekeszbe írni is, de csak akkor, ha ugyanazt akarják beírni. A gép akkor áll le programhibával, ha két processzor ugyanabba a rekeszbe más-más számot akar beírni.

P-CRCW (Priority Concurrent Read Concurrent Write) – Elsőbbségi modell

Szabad több processzornak ugyanabból a rekeszből olvasni és ugyanabba a rekeszbe írni. Ha többen akarnak ugyanoda írni, a legkisebb sorszámúnak sikerül.

PRAM gép

A PRAM-gépnél a processzorok számát nem csak azért szokás megadni, mert ettől függ a számítás, hanem azért is, mert ez – az idő és tár mellett – a számítás igen fontos bonyolultsági mértéke.

PRAM gép

A PRAM-gépnél a processzorok számát nem csak azért szokás megadni, mert ettől függ a számítás, hanem azért is, mert ez – az idő és tár mellett – a számítás igen fontos bonyolultsági mértéke.

Példa: Ha ezt nem korlátozzuk,akkor igen nehéz feladatokat is nagyon röviden meg tudunk oldani, pl. egy gráf 3 színnel való színezhetőségét el tudjuk dönteni úgy, hogy az alaphalmaz minden színezéséhez és a gráf minden éléhez rendelünk egy processzort (n csúcsú, e élű gráf esetén $e3^n \leqslant n^2 3^n$ darab processzor). Adott színezéshez tartozó élprocesszorok megnézik, hogy a színezésben az adott él két végpontja különböző színű-e.

PRAM gép

A PRAM-gépnél a processzorok számát nem csak azért szokás megadni, mert ettől függ a számítás, hanem azért is, mert ez – az idő és tár mellett – a számítás igen fontos bonyolultsági mértéke.

Példa: Ha ezt nem korlátozzuk,akkor igen nehéz feladatokat is nagyon röviden meg tudunk oldani, pl. egy gráf 3 színnel való színezhetőségét el tudjuk dönteni úgy, hogy az alaphalmaz minden színezéséhez és a gráf minden éléhez rendelünk egy processzort (n csúcsú, e élű gráf esetén $e3^n \leqslant n^2 3^n$ darab processzor). Adott színezéshez tartozó élprocesszorok megnézik, hogy a színezésben az adott él két végpontja különböző színű-e.

Az eredmények összesítése a CRCW modellben $O(1)$ lépésben megoldható.

PRAM gép

A PRAM-gépnél a processzorok számát nem csak azért szokás megadni, mert ettől függ a számítás, hanem azért is, mert ez – az idő és tár mellett – a számítás igen fontos bonyolultsági mértéke.

Példa: Ha ezt nem korlátozzuk,akkor igen nehéz feladatokat is nagyon röviden meg tudunk oldani, pl. egy gráf 3 színnel való színezhetőségét el tudjuk dönteni úgy, hogy az alaphalmaz minden színezéséhez és a gráf minden éléhez rendelünk egy processzort (n csúcsú, e élű gráf esetén $e3^n \leqslant n^2 3^n$ darab processzor). Adott színezéshez tartozó élprocesszorok megnézik, hogy a színezésben az adott él két végpontja különböző színű-e.

Az eredmények összesítése a CRCW modellben $O(1)$ lépésben megoldható. (Az élprocesszorok a színezésüknek „jelentenek'' egy közös memóriacímen, majd a színezések egyetlen egy közös memóriacímen „jelentik'', hogy jó-színezések-e.)

PRAM gép

A PRAM-gépnél a processzorok számát nem csak azért szokás megadni, mert ettől függ a számítás, hanem azért is, mert ez – az idő és tár mellett – a számítás igen fontos bonyolultsági mértéke.

Példa: Ha ezt nem korlátozzuk,akkor igen nehéz feladatokat is nagyon röviden meg tudunk oldani, pl. egy gráf 3 színnel való színezhetőségét el tudjuk dönteni úgy, hogy az alaphalmaz minden színezéséhez és a gráf minden éléhez rendelünk egy processzort (n csúcsú, e élű gráf esetén $e3^n \leqslant n^2 3^n$ darab processzor). Adott színezéshez tartozó élprocesszorok megnézik, hogy a színezésben az adott él két végpontja különböző színű-e.

Az eredmények összesítése a CRCW modellben $O(1)$ lépésben megoldható. (Az élprocesszorok a színezésüknek „jelentenek” egy közös memóriacímen, majd a színezések egyetlen egy közös memóriacímen „jelentik”, hogy jó-színezések-e.)

Az EREW modellben az összesítés színezésenként $O(\log n)$ lépésben, mindösszesen $O(n \log n)$ lépésben elvégezhető.

PRAM gép – Brent tétele

Feltehetjük, hogy egy párhuzamos számítási modell processzorai, amennyiben egy adott lépésben aktívak, 1 egységnyi munkát végeznek, míg ha inaktívak, akkor 0 egységnyit.

PRAM gép – Brent tétele

Feltehetjük, hogy egy párhuzamos számítási modell processzorai, amennyiben egy adott lépésben aktívak, 1 egységnyi munkát végeznek, míg ha inaktívak, akkor 0 egységnyit.

Definíció

A P (akármelyik PRAM-gép modellben írt) program adott bemenetre t lépésben végzett **összmunkája** $w = \sum_{i=1}^{t} w_i$, ahol w_i az i-edik lépésben aktív processzorok száma.

PRAM gép – Brent tétele

Feltehetjük, hogy egy párhuzamos számítási modell processzorai, amennyiben egy adott lépésben aktívak, 1 egységnyi munkát végeznek, míg ha inaktívak, akkor 0 egységnyit.

Definíció

A P (akármelyik PRAM-gép modellben írt) program adott bemenetre t lépésben végzett **összmunkája** $w = \sum_{i=1}^{t} w_i$, ahol w_i az i-edik lépésben aktív processzorok száma.

Brent tétele

Ha egy feladatot egy konkrét inputra a P program (akármelyik PRAM-gép modellben) valahány processzorral t lépésben és w összmunkával megoldja, akkor minden p pozitív egész számra megoldható p processzorral is $(w - t)/p + t$ lépésben (ugyanabban a modellben).

PRAM gép – Brent tétele

Bizonyítás: Tegyük fel, hogy a P program i-edik lépésében w_i darab processzor aktív, ekkor az összmunka $w = \sum_{i=1}^{t} w_i$.

PRAM gép – Brent tétele

Bizonyítás: Tegyük fel, hogy a P program i-edik lépésében w_i darab processzor aktív, ekkor az összmunka $w = \sum_{i=1}^{t} w_i$. Az eredeti algoritmus i-edik lépése során elvégzett feladatot nyilván el lehet végezni p processzorral $\lceil w_i/p \rceil$ lépésben,

PRAM gép – Brent tétele

Bizonyítás: Tegyük fel, hogy a P program i-edik lépésében w_i darab processzor aktív, ekkor az összmunka $w = \sum_{i=1}^{t} w_i$. Az eredeti algoritmus i-edik lépése során elvégzett feladatot nyilván el lehet végezni p processzorral $\lceil w_i/p \rceil$ lépésben, így a szükséges idő $\sum_{i=1}^{t} \lceil w_i/p \rceil \leqslant \sum_{i=1}^{t} (w_i + p - 1)/p \leqslant (w - t)/p + t$. $\square$

PRAM gép – Brent tétele

Bizonyítás: Tegyük fel, hogy a P program i-edik lépésében w_i darab processzor aktív, ekkor az összmunka $w = \sum_{i=1}^{t} w_i$. Az eredeti algoritmus i-edik lépése során elvégzett feladatot nyilván el lehet végezni p processzorral $\lceil w_i/p \rceil$ lépésben, így a szükséges idő $\sum_{i=1}^{t} \lceil w_i/p \rceil \leqslant \sum_{i=1}^{t} (w_i + p - 1)/p \leqslant (w - t)/p + t$. □

Példa: ha van egy feladatra párhuzamos algoritmusunk, mely n^2 processzort használ és $\log_2 n$ lépést tesz, akkor olyan is van, mely

- 32 processzort használ $\leqslant n^2 \log_2 n/32 + \log_2 n = O(n^2 \log n)$ lépést tesz
- $\sqrt{n}$ processzort használ és $\leqslant n^2 \log_2 n/\sqrt{n} + \log_2 n = O(n^{3/2} \log n)$ lépést tesz

PRAM gép – Brent tétele

Bizonyítás: Tegyük fel, hogy a P program i-edik lépésében w_i darab processzor aktív, ekkor az összmunka $w = \sum_{i=1}^{t} w_i$. Az eredeti algoritmus i-edik lépése során elvégzett feladatot nyilván el lehet végezni p processzorral $\lceil w_i/p \rceil$ lépésben, így a szükséges idő $\sum_{i=1}^{t} \lceil w_i/p \rceil \leqslant \sum_{i=1}^{t} (w_i + p - 1)/p \leqslant (w - t)/p + t$. $\square$

Példa: ha van egy feladatra párhuzamos algoritmusunk, mely n^2 processzort használ és $\log_2 n$ lépést tesz, akkor olyan is van, mely

- 32 processzort használ $\leqslant n^2 \log_2 n/32 + \log_2 n = O(n^2 \log n)$ lépést tesz
- $\sqrt{n}$ processzort használ és $\leqslant n^2 \log_2 n/\sqrt{n} + \log_2 n = O(n^{3/2} \log n)$ lépést tesz

Valóban, $t = \log_2 n$ a w összmunkára $w \leqslant n^2 \log_2 n$, alkalmazzuk Brent tételét.

PRAM gép – P-CRCW EREW szimulációja

A P-CRCW számítások hatékonyan szimulálhatók az EREW modellben.

PRAM gép – P-CRCW EREW szimulációja

A P-CRCW számítások hatékonyan szimulálhatók az EREW modellben.

Tétel

Minden P programhoz létezik olyan Q program, hogy ha P egy bemenetből p processzorral t időben kiszámít egy kimenetet az elsőbbségi (P-CRCW) modellben, akkor a Q program $O(p^2)$ processzorral $O(t \log^2 p)$ időben a teljesen konfliktusmentes (EREW) modellben számítja ki ugyanazt.

PRAM gép – P-CRCW EREW szimulációja

A P-CRCW számítások hatékonyan szimulálhatók az EREW modellben.

Tétel

Minden P programhoz létezik olyan Q program, hogy ha P egy bemenetből p processzorral t időben kiszámít egy kimenetet az elsőbbségi (P-CRCW) modellben, akkor a Q program $O(p^2)$ processzorral $O(t \log^2 p)$ időben a teljesen konfliktusmentes (EREW) modellben számítja ki ugyanazt.

Bizonyítás:

A P programot végrehajtó minden processzornak meg fog felelni egy processzor a Q program végrehajtása során. Ezeket főnökprocesszoroknak nevezzük.

PRAM gép – P-CRCW EREW szimulációja

A P-CRCW számítások hatékonyan szimulálhatók az EREW modellben.

Tétel

Minden P programhoz létezik olyan Q program, hogy ha P egy bemenetből p processzorral t időben kiszámít egy kimenetet az elsőbbségi (P-CRCW) modellben, akkor a Q program $O(p^2)$ processzorral $O(t \log^2 p)$ időben a teljesen konfliktusmentes (EREW) modellben számítja ki ugyanazt.

Bizonyítás:

A P programot végrehajtó minden processzornak meg fog felelni egy processzor a Q program végrehajtása során. Ezeket főnökprocesszoroknak nevezzük.

Ezenfelül a Q programban minden főnökprocesszornak p további „segédprocesszora'' is lesz.

PRAM gép – P-CRCW EREW szimulációja

Az a konstrukció alapgondolata, hogy ami a P program futása során egy adott ütem után a z című memóriarekeszben van, az a Q program futása során a megfelelő ütemben a $2pz, 2pz+1, \ldots,$ $2pz+p-1$ című rekeszek mindegyikében meglegyen.

PRAM gép – P-CRCW EREW szimulációja

Az a konstrukció alapgondolata, hogy ami a P program futása során egy adott ütem után a z című memóriarekeszben van, az a Q program futása során a megfelelő ütemben a $2pz, 2pz+1, \ldots,$ $2pz+p-1$ című rekeszek mindegyikében meglegyen.

	17	53	8		
	z	$z{+}1$	$z{+}2$		

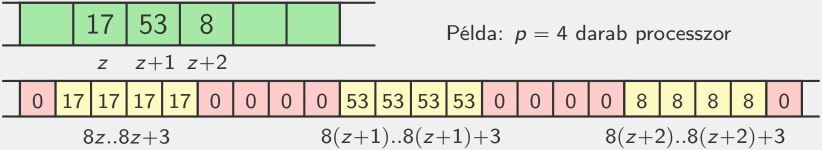
Példa: $p = 4$ darab processzor

0	17	17	17	17	0	0	0	0	53	53	53	53	0	0	0	0	8	8	8	8	0

$8z..8z{+}3$ $\quad$ $8(z{+}1)..8(z{+}1){+}3$ $\quad$ $8(z{+}2)..8(z{+}2){+}3$

PRAM gép – P-CRCW EREW szimulációja

Az a konstrukció alapgondolata, hogy ami a P program futása során egy adott ütem után a z című memóriarekeszben van, az a Q program futása során a megfelelő ütemben a $2pz, 2pz+1, \ldots,$ $2pz+p-1$ című rekeszek mindegyikében meglegyen.

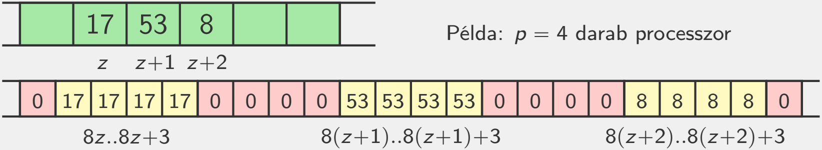


Példa: $p = 4$ darab processzor

Ha a P következő ütemében az i-edik processzor z című rekeszből kell, hogy olvasson, akkor a Q program következő ütemében az ennek megfelelő processzor olvashatja a $2pz + i$ című rekeszt.

PRAM gép – P-CRCW EREW szimulációja

Az a konstrukció alapgondolata, hogy ami a P program futása során egy adott ütem után a z című memóriarekeszben van, az a Q program futása során a megfelelő ütemben a $2pz, 2pz+1, \ldots,$ $2pz+p-1$ című rekeszek mindegyikében meglegyen.

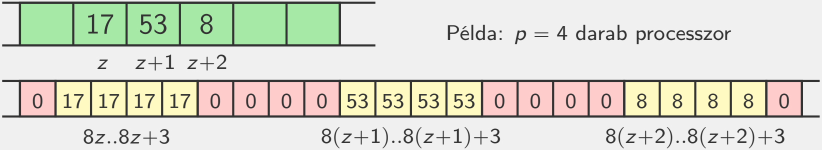


Ha a P következő ütemében az i-edik processzor z című rekeszből kell, hogy olvasson, akkor a Q program következő ütemében az ennek megfelelő processzor olvashatja a $2pz+i$ című rekeszt.

Ha pedig ebben az ütemben (P szerint) a z című rekeszbe kell, hogy írjon, akkor Q szerint a $2pz+i$ rekeszbe írhat.

PRAM gép – P-CRCW EREW szimulációja

Az a konstrukció alapgondolata, hogy ami a P program futása során egy adott ütem után a z című memóriarekeszben van, az a Q program futása során a megfelelő ütemben a $2pz, 2pz + 1, \ldots,$ $2pz + p - 1$ című rekeszek mindegyikében meglegyen.



Példa: $p = 4$ darab processzor

Ha a P következő ütemében az i-edik processzor z című rekeszből kell, hogy olvasson, akkor a Q program következő ütemében az ennek megfelelő processzor olvashatja a $2pz + i$ című rekeszt.

Ha pedig ebben az ütemben (P szerint) a z című rekeszbe kell, hogy írjon, akkor Q szerint a $2pz + i$ rekeszbe írhat.

Ez biztosan elkerüli az összes konfliktust, mert a különböző processzorok modulo p különböző rekeszeket használnak.

PRAM gép – P-CRCW EREW szimulációja

Gondoskodni kell azonban arról, hogy a Q program futása során a $2pz, 2pz+1, \ldots, 2pz+p-1$ rekeszek mindegyikébe az a szám kerüljön, amit az elsőbbségi szabály a P program futásának megfelelő ütemében z-be ír.

PRAM gép – P-CRCW EREW szimulációja

Gondoskodni kell azonban arról, hogy a Q program futása során a $2pz, 2pz + 1, \ldots, 2pz + p - 1$ rekeszek mindegyikébe az a szám kerüljön, amit az elsőbbségi szabály a P program futásának megfelelő ütemében z-be ír.

Ehhez a P futását szimuláló minden ütem után beiktatunk egy $O(\log p)$ lépésből álló fázist, amely ezt megvalósítja.

PRAM gép – P-CRCW EREW szimulációja

Gondoskodni kell azonban arról, hogy a Q program futása során a $2pz, 2pz+1, \ldots, 2pz+p-1$ rekeszek mindegyikébe az a szám kerüljön, amit az elsőbbségi szabály a P program futásának megfelelő ütemében z-be ír.

Ehhez a P futását szimuláló minden ütem után beiktatunk egy $O(\log p)$ lépésből álló fázist, amely ezt megvalósítja.

Először is minden olyan processzor, mely az előző (P-t szimuláló) ütemben írt, mondjuk a $2pz+i$ című rekeszbe, ír egy 1-est a $2pz+p+i$ című rekeszbe.

PRAM gép – P-CRCW EREW szimulációja

Gondoskodni kell azonban arról, hogy a Q program futása során a $2pz, 2pz+1, \ldots, 2pz+p-1$ rekeszek mindegyikébe az a szám kerüljön, amit az elsőbbségi szabály a P program futásának megfelelő ütemében z-be ír.

Ehhez a P futását szimuláló minden ütem után beiktatunk egy $O(\log p)$ lépésből álló fázist, amely ezt megvalósítja.

Először is minden olyan processzor, mely az előző (P-t szimuláló) ütemben írt, mondjuk a $2pz+i$ című rekeszbe, ír egy 1-est a $2pz+p+i$ című rekeszbe.

	17	53	8		

0	17	4	17	-3	0	1	0	1	53	53	6	53	0	0	1	0	8	8	8	8	0

proc2: $x[z] := 4$
proc4: $x[z] := -3$
proc3: $x[z+1] := 6$

PRAM gép – P-CRCW EREW szimulációja

A következő lépésben megnézi, hogy a $2pz + p + i - 1$ rekeszben 1-es áll-e.

PRAM gép – P-CRCW EREW szimulációja

A következő lépésben megnézi, hogy a $2pz + p + i - 1$ rekeszben 1-es áll-e.

Ha igen, elalszik ennek a fázisnak a hátralevő idejére.

PRAM gép – P-CRCW EREW szimulációja

A következő lépésben megnézi, hogy a $2pz + p + i - 1$ rekeszben 1-es áll-e.

Ha igen, elalszik ennek a fázisnak a hátralevő idejére.

Ha nem, akkor ő ír oda 1-est, majd „felébreszti” egy segédjét.

PRAM gép – P-CRCW EREW szimulációja

A következő lépésben megnézi, hogy a $2pz + p + i - 1$ rekeszben 1-es áll-e.

Ha igen, elalszik ennek a fázisnak a hátralevő idejére.

Ha nem, akkor ő ír oda 1-est, majd „felébreszti" egy segédjét.

Általában a k-adik lépésben az i-edik processzornak legfeljebb 2^{k-1} segédje lesz ébren (magát is beleértve), és ezek rendre, amennyiben aktívak, a $2pz + p + i - 2^{k-1}, \ldots, 2pz + p + i - (2^k - 1)$ című rekeszeket olvassák le.

PRAM gép – P-CRCW EREW szimulációja

A következő lépésben megnézi, hogy a $2pz + p + i - 1$ rekeszben 1-es áll-e.

Ha igen, elalszik ennek a fázisnak a hátralevő idejére.

Ha nem, akkor ő ír oda 1-est, majd „felébreszti” egy segédjét.

Általában a k-adik lépésben az i-edik processzornak legfeljebb 2^{k-1} segédje lesz ébren (magát is beleértve), és ezek rendre, amennyiben aktívak, a $2pz + p + i - 2^{k-1}, \dots, 2pz + p + i - (2^k - 1)$ című rekeszeket olvassák le.

Amelyik 1-est talál, elalszik.

PRAM gép – P-CRCW EREW szimulációja

A következő lépésben megnézi, hogy a $2pz + p + i - 1$ rekeszben 1-es áll-e.

Ha igen, elalszik ennek a fázisnak a hátralevő idejére.

Ha nem, akkor ő ír oda 1-est, majd „felébreszti” egy segédjét.

Általában a k-adik lépésben az i-edik processzornak legfeljebb 2^{k-1} segédje lesz ébren (magát is beleértve), és ezek rendre, amennyiben aktívak, a $2pz + p + i - 2^{k-1}, \ldots, 2pz + p + i - (2^k - 1)$ című rekeszeket olvassák le.

Amelyik 1-est talál, elalszik.

Amelyik nem talál 1-est, az 1-est ír, felébreszt egy új segédet, azt 2^{k-1} hellyel „balra” küldi, míg maga 2^k hellyel „balra” lép.

PRAM gép – P-CRCW EREW szimulációja

A következő lépésben megnézi, hogy a $2pz + p + i - 1$ rekeszben 1-es áll-e.

Ha igen, elalszik ennek a fázisnak a hátralevő idejére.

Ha nem, akkor ő ír oda 1-est, majd „felébreszti" egy segédjét.

Általában a k-adik lépésben az i-edik processzornak legfeljebb 2^{k-1} segédje lesz ébren (magát is beleértve), és ezek rendre, amennyiben aktívak, a $2pz + p + i - 2^{k-1}, \ldots, 2pz + p + i - (2^k - 1)$ című rekeszeket olvassák le.

Amelyik 1-est talál, elalszik.

Amelyik nem talál 1-est, az 1-est ír, felébreszt egy új segédet, azt 2^{k-1} hellyel „balra" küldi, míg maga 2^k hellyel „balra" lép.

Amelyik segéd kiér a $[2pz + p, 2pz + 2p - 1]$ intervallumból, elalszik; ha egy főnök kiér ebből az intervallumból, akkor már tudja, hogy ő „győzött".

PRAM gép – P-CRCW EREW szimulációja

Könnyű meggondolni, hogy ha a P program megfelelő ütemében több processzor is írni akart a z rekeszbe, akkor az ezeknek megfelelő főnökök és segédeik nem kerülnek konfliktusba, mialatt a $[2pz + p, 2pz + 2p - 1]$ intervallumban mozognak.

PRAM gép – P-CRCW EREW szimulációja

Könnyű meggondolni, hogy ha a P program megfelelő ütemében több processzor is írni akart a z rekeszbe, akkor az ezeknek megfelelő főnökök és segédeik nem kerülnek konfliktusba, mialatt a $[2pz + p, 2pz + 2p - 1]$ intervallumban mozognak.

A k-adik lépésre ugyanis az i-edik processzor és segédei $2pz + p + i$-től lefelé 2^{k-1} egymás utáni helyre 1-est írtak.

PRAM gép – P-CRCW EREW szimulációja

Könnyű meggondolni, hogy ha a P program megfelelő ütemében több processzor is írni akart a z rekeszbe, akkor az ezeknek megfelelő főnökök és segédeik nem kerülnek konfliktusba, mialatt a $[2pz + p, 2pz + 2p - 1]$ intervallumban mozognak.

A k-adik lépésre ugyanis az i-edik processzor és segédei $2pz + p + i$-től lefelé 2^{k-1} egymás utáni helyre 1-est írtak.

Minden tőlük jobbra induló processzor és annak minden segédje ebbe szükségképpen bele fog lépni, és ezért elalszik, mielőtt konfliktusba kerülhetne az i-edik processzor segédeivel.

PRAM gép – P-CRCW EREW szimulációja

Könnyű meggondolni, hogy ha a P program megfelelő ütemében több processzor is írni akart a z rekeszbe, akkor az ezeknek megfelelő főnökök és segédeik nem kerülnek konfliktusba, mialatt a $[2pz+p, 2pz+2p-1]$ intervallumban mozognak.

A k-adik lépésre ugyanis az i-edik processzor és segédei $2pz+p+i$-től lefelé 2^{k-1} egymás utáni helyre 1-est írtak.

Minden tőlük jobbra induló processzor és annak minden segédje ebbe szükségképpen bele fog lépni, és ezért elalszik, mielőtt konfliktusba kerülhetne az i-edik processzor segédeivel.

Ha nem ért még célba főnök és legalább 2 főnök aktív még, akkor a hátrébbról induló épp akkorát lép, mint az előbbről induló által eddig készített 1-esekből álló blokk mérete, mivel ez a blokk teljes egészében előtte van, így nem érhet a következő lépésben célba.

PRAM gép – P-CRCW EREW szimulációja

Könnyű meggondolni, hogy ha a P program megfelelő ütemében több processzor is írni akart a z rekeszbe, akkor az ezeknek megfelelő főnökök és segédeik nem kerülnek konfliktusba, mialatt a $[2pz + p, 2pz + 2p - 1]$ intervallumban mozognak.

A k-adik lépésre ugyanis az i-edik processzor és segédei $2pz + p + i$-től lefelé 2^{k-1} egymás utáni helyre 1-est írtak.

Minden tőlük jobbra induló processzor és annak minden segédje ebbe szükségképpen bele fog lépni, és ezért elalszik, mielőtt konfliktusba kerülhetne az i-edik processzor segédeivel.

Ha nem ért még célba főnök és legalább 2 főnök aktív még, akkor a hátrébbról induló épp akkorát lép, mint az előbbről induló által eddig készített 1-esekből álló blokk mérete, mivel ez a blokk teljes egészében előtte van, így nem érhet a következő lépésben célba.

Ez mutatja azt is, hogy mindig egyetlenegy főnök fog győzni, éspedig a legkisebb sorszámú.

PRAM gép – P-CRCW EREW szimulációja

Tehát $O(\log p)$ lépésben eldől, melyik processzor a „győztes”. Ennek a processzornak még van egy dolga. Amit a $2pz + i$. rekeszbe írt, azt most az egész $[2pz, 2pz + p - 1]$ intervallum minden rekeszébe be kell írnia.

PRAM gép – P-CRCW EREW szimulációja

Tehát $O(\log p)$ lépésben eldől, melyik processzor a „győztes”. Ennek a processzornak még van egy dolga. Amit a $2pz + i$. rekeszbe írt, azt most az egész $[2pz, 2pz + p - 1]$ intervallum minden rekeszébe be kell írnia.

Ezt könnyű megtenni $O(\log p)$ lépésben az előzőhöz nagyon hasonló eljárással: ő maga beírja a kívánt értéket a $2pz$ című rekeszbe, majd felébreszt egy segédet; beírják a kívánt értéket a $2pz + 1$ és $2pz + 2$ rekeszekbe, majd felébresztenek egy-egy segédet stb.

PRAM gép – P-CRCW EREW szimulációja

Tehát $O(\log p)$ lépésben eldől, melyik processzor a „győztes”. Ennek a processzornak még van egy dolga. Amit a $2pz + i$. rekeszbe írt, azt most az egész $[2pz, 2pz + p - 1]$ intervallum minden rekeszébe be kell írnia.

Ezt könnyű megtenni $O(\log p)$ lépésben az előzőhöz nagyon hasonló eljárással: ő maga beírja a kívánt értéket a $2pz$ című rekeszbe, majd felébreszt egy segédet; beírják a kívánt értéket a $2pz + 1$ és $2pz + 2$ rekeszekbe, majd felébresztenek egy-egy segédet stb.

Ha mindannyian kiértek a $[2pz, 2pz + p - 1]$ intervallumból, jöhet a következő szimulálandó P-CRCW lépés.

PRAM gép – P-CRCW EREW szimulációja

Egy m költségű P-CRCW utasítás szimulálásához használt minden egyes EREW utasítás költsége $O(\log p + m)$, mivel a z című rekesz helyett $O(pz)$ méretűeket használunk, így $\log z$ helyett $O(\log p + \log z)$ lesz a címből adódó költség, így a nagyobb című rekeszek használata csak egy $O(\log p)$-s additív taggal növeli a költséget.

PRAM gép – P-CRCW EREW szimulációja

Egy m költségű P-CRCW utasítás szimulálásához használt minden egyes EREW utasítás költsége $O(\log p + m)$, mivel a z című rekesz helyett $O(pz)$ méretűeket használunk, így $\log z$ helyett $O(\log p + \log z)$ lesz a címből adódó költség, így a nagyobb című rekeszek használata csak egy $O(\log p)$-s additív taggal növeli a költséget.

A bizonyítás során egy P-CRCW lépést $O(\log p)$ darab EREW lépéssel szimuláltunk, így összességében az elsőbbségi modell egy m költségű lépésének szimulációja $O((\log p)(\log p + m))$ költségű.

PRAM gép – P-CRCW EREW szimulációja

Egy m költségű P-CRCW utasítás szimulálásához használt minden egyes EREW utasítás költsége $O(\log p + m)$, mivel a z című rekesz helyett $O(pz)$ méretűeket használunk, így $\log z$ helyett $O(\log p + \log z)$ lesz a címből adódó költség, így a nagyobb című rekeszek használata csak egy $O(\log p)$-s additív taggal növeli a költséget.

A bizonyítás során egy P-CRCW lépést $O(\log p)$ darab EREW lépéssel szimuláltunk, így összességében az elsőbbségi modell egy m költségű lépésének szimulációja $O((\log p)(\log p + m))$ költségű.

Mivel minden $p \geqslant 2$-re $(\log p)(\log p + m) \leqslant (\log^2 p)m$, ezért az elsőbbségi modellbeli lépések szimulálása a konfliktusmentes modellben a költséget egy $O(\log^2 p)$-s szorzóval növeli.

PRAM gép – P-CRCW EREW szimulációja

Egy m költségű P-CRCW utasítás szimulálásához használt minden egyes EREW utasítás költsége $O(\log p + m)$, mivel a z című rekesz helyett $O(pz)$ méretűeket használunk, így $\log z$ helyett $O(\log p + \log z)$ lesz a címből adódó költség, így a nagyobb című rekeszek használata csak egy $O(\log p)$-s additív taggal növeli a költséget.

A bizonyítás során egy P-CRCW lépést $O(\log p)$ darab EREW lépéssel szimuláltunk, így összességében az elsőbbségi modell egy m költségű lépésének szimulációja $O((\log p)(\log p + m))$ költségű.

Mivel minden $p \geqslant 2$-re $(\log p)(\log p + m) \leqslant (\log^2 p)m$, ezért az elsőbbségi modellbeli lépések szimulálása a konfliktusmentes modellben a költséget egy $O(\log^2 p)$-s szorzóval növeli.

Mivel ez minden lépésről elmondható, ezért a tétel futási időre vonatkozó állítását beláttuk. $\square$