6.6 Die Klassen NC^i, AC^i und NC als Modelle für effiziente parallele Berechnungen

Die Tiefe eines Schaltkreises C ist die Länge eines längsten gerichteten Weges in C.
Offensichtlicherweise kann für einen SK C der Größe s und Tiefe t der Wert $C(x)$ bei Eingabe eines Wortes $x \in \{0,1\}^n$ von s parallel arbeitenden Prozessoren in t Berechnungsschritten ermittelt werden.

Definition 6.26 (NC^i und NC)

(a) Sei $i \in \mathbb{N}$. Die Klasse

NC^i

besteht aus allen Sprachen $L \subseteq \{0,1\}^*$, die von einer SK-Familie $(C_n)_{n\in\mathbb{N}}$ der Größe $poly(n)$ und der Tiefe $O((\log n)^i)$ berechnet werden können.
(D.h.: $(C_n)_{n\in\mathbb{N}}$ berechnet L, und es gibt Konstanten $c, d \geq 1$ s.d. f.a. $n \in \mathbb{N}$ gilt: $|C_n| \leq n^c$ und die Tiefe von C_n ist $\leq d \cdot (\log n)^i$).

(b) $NC := \bigcup_{i\in\mathbb{N}} NC^i$ ist die Klasse aller Sprachen, die von einer SK-Familie polynomieller Größe und polylogarithmischer Tiefe berechnet werden können.

Die Bezeichnung "NC" steht für "Nick's class", benannt nach dem Komplexitätstheoretiker Nick Pippenger (diese Bezeichnung wurde von Stephen Cook geprägt). Klar: $NC \subseteq P/poly$.

Bemerkung 6.27

Für jedes sinnvolle Modell von Parallelrechnern lässt sich eine Variante der folgenden Aussage beweisen:

$L \in$ logspace-uniformes NC

$\Leftrightarrow$ L kann unter Verwendung von poly(n) vielen Prozessoren in Zeit polylog(n) entschieden werden.

Ähnlich wie wir die Klasse P als Modell für "effizient lösbare Probleme" sehen, betrachten wir die Klasse NC als Modell für "effizient parallelisierbare Probleme". Aus Satz 6.24 folgt: logspace-uniformes $NC \subseteq P$.

Beispiel 6.28

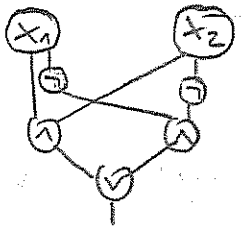
Für die Sprache

$PARITY := \{ x \in \{0,1\}^* : $ die Anzahl der 1en in x ist gerade $\}$

gilt: $PARITY \in NC^1$

Denn: Sei $(C_n)_{n \in \mathbb{N}}$ die SK-Familie mit

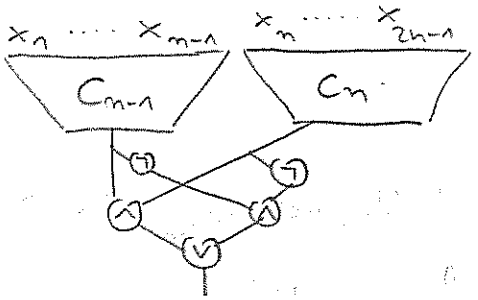
C_1: 

C_2: 

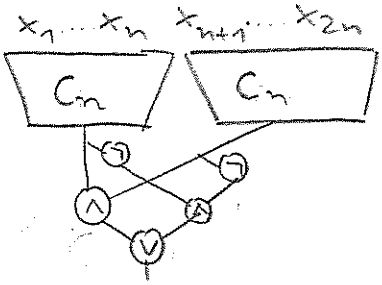
und

f.a. $n \geq 2$:

C_{2n-1}:

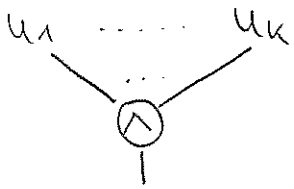


C_{2n}:



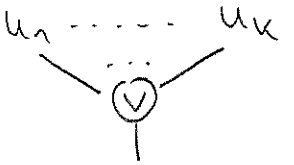
Dann gilt: $(C_n)_{n \in \mathbb{N}}$ ist eine SK-Familie der Größe $O(n)$ und Tiefe $O(\log n)$, die PARITY berechnet. □ Bsp 6.28

Manchmal betrachtet man auch Schaltkreise, bei denen ∧- und ∨-Gatter von unbeschränktem fan-in erlaubt sind. Ein ∧-Gatter der Form



erhält genau dann den Wert 1, wenn $u_1 = \cdots = u_k = 1$ ist.

Ein ∨-Gatter der Form



erhält genau dann den Wert 1, wenn mindestens eins der $u_1, \ldots, u_k$ den Wert 1 hat.

Wir sprechen im Folgenden von <u>uSKs</u>, um Schaltkreise zu bezeichnen, die zusätzlich zu den bisher betrachteten Gattern auch ∧-Gatter und ∨-Gatter von <u>unbeschränktem</u> fan-in benützen dürfen.

Definition 6.29 (AC^i)

(a) Sei $i \in \mathbb{N}$. Die Klasse

$$AC^i$$

besteht aus allen Sprachen $L \subseteq \{0,1\}^*$, die von einer uSK-Familie $(C_n)_{n\in\mathbb{N}}$ der Größe $poly(n)$ und der Tiefe $O((\log n)^i)$ berechnet werden können.

(b) $$AC := \bigcup_{i\in\mathbb{N}} AC^i$$

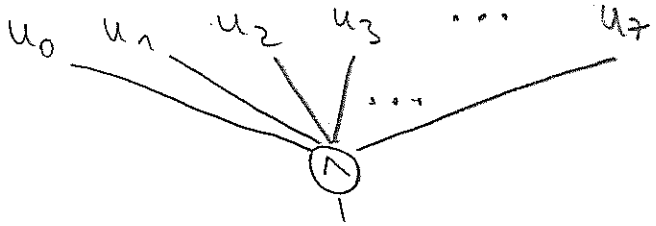
(Der Buchstabe 'A' bei "AC" steht für "alternierend".)

Fakt 6.30

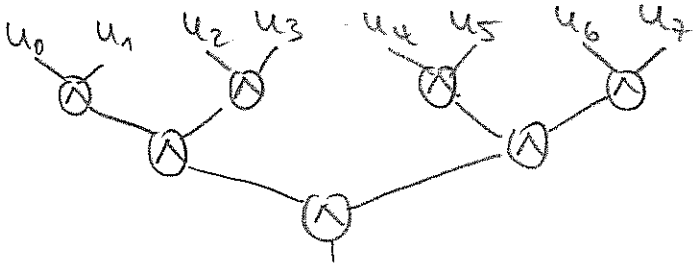
(a) F.a. $i \geq 0$ gilt: $NC^i \subseteq AC^i$ (klar per Definition)

(b) F.a. $i \geq 0$ gilt: $AC^i \subseteq NC^{i+1}$, denn:

Ein $\wedge$-Gatter vom fan-in 8



kann durch einen SK der Größe 7 und Tiefe 3 der folgenden Form ersetzt werden:



Allgemein kann ein ∧-Gatter (bzw. ein ∨-Gatter) vom fan-in $\leq s$ durch einen (herkömmlichen) SK der Größe $\leq s$ und Tiefe $\leq \lceil \log s \rceil$ ersetzt werden.

Daher kann ein uSK der Größe s und Tiefe t durch einen SK der Größe $\leq s \cdot \lceil \log s \rceil$ und Tiefe $\leq t \cdot \lceil \log s \rceil$ simuliert werden.

Daraus folgt insbes.: $AC^i \subseteq NC^{i+1}$.

(c) $AC = NC$ (folgt direkt aus (a) und (b)).

Satz 6.31

Für die Sprache PATH $:=$ $\{ \langle G, s, t \rangle :$ G ist ein endlicher gerichteter Graph, in dem es einen Weg von Knoten s zu Knoten t gibt

gilt: PATH $\in$ logspace-uniformes AC^1

Beweis:

Wir konstruieren eine uSK-Familie, die das iterierte Boolesche Produkt der Adjazenzmatrix von G berechnet.

Details: Übung!

□

Satz 6.32

NL $\subseteq$ logspace-uniformes AC^1

Beweis:

Sei $L \in NL$ und sei M eine $O(\log n)$ platzbeschränkte NDTM, die L entscheidet. F.a. $x \in \{0,1\}^*$ gilt:

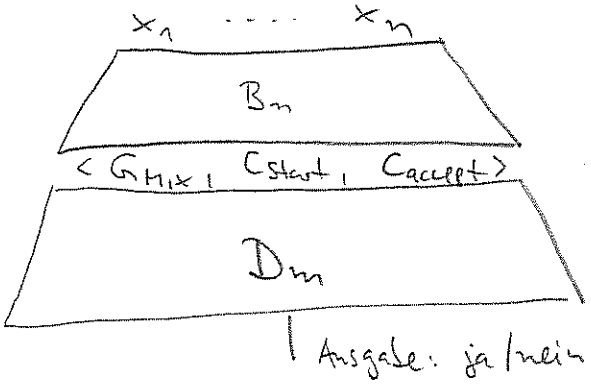
$x \in L \iff \langle G_{M,x}, C_{start}, C_{accept} \rangle \in PATH$,

wobei $G_{M,x}$ der Konfigurationsgraph von M bei Eingabe x ist.

Wir konstruieren eine uSK-Familie $(C_n)_{n \in \mathbb{N}}$, so dass C_n bei Eingabe von $x \in \{0,1\}^n$

- zunächst aus dem Bitstring x den Bitstring $\langle G_{M,x}, C_{start}, C_{accept} \rangle$ erzeugt (durch einen uSK B_n) und
- dann diesen mit dem uSK D_m aus Satz 6.31 verarbeitet, der testet, ob $\langle G_{M,x}, C_{start}, C_{accept} \rangle \in PATH$ ist.

Skizze:



Hier ist D_m der uSK der Größe poly(m) und Tiefe $O(\log m)$ aus Satz 6.31, der testet ob ein Bitstring $\langle G_{M,x}, C_{start}, C_{accept} \rangle$ der Länge m zur Sprache PATH gehört.

Beachte: Hier ist $m \leq n^c$, für eine Konstante c,
da $G_{M,x}$ höchstens $2^{O(\log n)}$ Knoten hat
(denn M ist $O(\log n)$ platzbeschränkt).
Somit hat D_m Größe poly(n) und Tiefe $O(\log n)$.

Daher sind wir fertig, sobald wir einen uSk B_n
der Größe poly(n) und Tiefe $O(\log n)$ konstruiert haben,
der bei Eingabe von $x \in \{0,1\}^n$ einen Bitstring der
Form $\langle G_{M,x}, C_{start}, C_{accept} \rangle$ erzeugt.

Wir repräsentieren jeden Knoten von $G_{M,x}$ durch einen
Bitstring der Länge $d \cdot \log n$, bei dem die ersten $\log n$ Bits
die aktuelle Kopfposition auf dem Eingabeband bezeichnet
und die restlichen Bits die Beschriftung und die
Kopfpositionen auf den Arbeitsbändern beschreiben.
Jeden Bitstring der Länge $d \cdot \log n$ identifizieren wir mit
einer Zahl aus $\{0, 1, \ldots, n^d - 1\}$. D.h.: Zahlen
$i \in \{0, 1, \ldots, n^d - 1\}$ repräsentieren Konfigurationen von M bei
Eingabe x.
Der Eintrag A_{ij} in Zeile i und Spalte j der
Adjazenzmatrix von $G_{M,x}$ hängt nur ab von
i, j, den Überführungsfunktionen δ_0 und δ_1 von M
und dem Eintrag von x an der Kopfposition $p(i)$
des Eingabebandes, die zur Konfiguration i gehört.

D.h: A_{ij} hängt nur von einem einzigen Bit der
Eingabe x ab, nämlich von $x_{p(i)}$.

Daher kann B_n sogar durch einen Sk der Größe

poly(n) und konstanter Tiefe erzeugt werden.

Durch Zusammensetzen von B_n und D_m erhalten wir einen uSK C_n der Größe poly(n) und Tiefe $O(\log n)$, der bei Eingabe von $x \in \{0,1\}^n$ testet, ob $x \in L$ ist.

Somit ist $NL \subseteq AC^1$.

Man kann sich leicht davon überzeugen, dass die konstruierte uSK-Familie $(C_n)_{n \in \mathbb{N}}$ logspace-uniform ist. □

Folgerung 6.33

(a) Für jedes $i \geq 1$ ist AC^i abgeschlossen unter logspace-Reduktionen
(d.h: Ist $A \leq_\ell B$ und $B \in AC^i$, so auch $A \in AC^i$)

(b) Für jedes $i \geq 2$ ist NC^i abgeschlossen unter logspace-Reduktionen

(c) Falls es ein P-vollständiges Problem L mit $L \in NC$ gibt, so ist $P \subseteq NC$.

Insbesondere gilt:
Falls CIRCUIT-EVAL $\in NC$, so $P \subseteq NC$.

Beweis:

(a) Nutze, dass gemäß Satz 6.32 gilt: $L \subseteq NL \subseteq AC^1$ und betrachte eine implizit logspace-berechenbare Funktion f, die A auf B reduziert.
Details: Übung!

(b) Analog zu (a)

(c) Folgt leicht aus (b), da $NC = \bigcup_{i \geq 0} NC^i$.
Zur Erinnerung: Von Folgerung 6.15 wissen wir, dass CIRCUIT-EVAL P-vollständig ist. □

Bemerkung 6.34

Aus Beispiel 6.28 wissen wir, dass PARITY $\in NC^1$ ist.

(Zur Erinnerung: $\text{PARITY} = \{x \in \{0,1\}^* : \text{die Anzahl der 1en in } x \text{ ist gerade}\}$.)

Man kann leicht zeigen, dass PARITY $\notin NC^0$ (Übung!).

Viel schwerer zu beweisen ist der folgende Satz:

Theorem 6.35 (Furst, Saxe, Sipser 1981; Ajtai 1983)

PARITY $\notin AC^0$

(Hier ohne Beweis)

Kapitel 7: Randomisierte Berechnungen

Randomisierte Algorithmen =
Algorithmen, die im Lauf ihrer Berechnungen "Zufallsbits" verwenden dürfen

7.1 Ein probabilistischer Primzahltest

Theorem 7.1

Es gibt einen randomisierten Algorithmus A, der bei Eingabe einer natürlichen Zahl n nach poly(log n) Schritten anhält, und für dessen Ausgabe A(n) gilt:

- falls n eine Primzahl ist, so ist A(n) = "ja"
- falls n keine Primzahl ist, so gilt mit Wahrscheinlichkeit $\geq \frac{1}{2}$: A(n) = "nein"

Zum Beweis von Theorem 7.1 benötigen wir einige Begriffe und Resultate aus der elementaren Zahlentheorie:

Definition 7.2 (Legendre-Symbol (m/p) und Jacobi-Symbol (m/n))

(a) Sei p eine Primzahl $\neq 2$ und sei $m \in \mathbb{N}$.
Das Legendre-Symbol (m/p) ist die wie folgt definierte Zahl aus $\{0, 1, -1\}$:

$$(m/p) := \begin{cases} 0 & \text{falls } p \text{ ein Teiler von } m \text{ ist} \\ 1 & \text{falls es ein } w \in \mathbb{N} \text{ gibt, s.d.} \\ & \quad m \equiv w^2 \pmod p \text{ ist} \\ -1 & \text{sonst} \end{cases}$$

Bemerkung: $(m/p) = 1$ bedeutet, dass m ein sog. quadratischer Rest modulo p ist

(b) Sei n eine ungerade natürliche Zahl > 1 und sei $p_1^{k_1} \cdots p_e^{k_e}$ die Primzahlzerlegung von n (d.h.: $p_1 < \cdots < p_e$ sind paarweise verschiedene Primzahlen, $k_1, \dots, k_e \geq 1$ und $n = p_1^{k_1} \cdots p_e^{k_e}$.
Sei $m \in \mathbb{N}$. Das Jacobi-Symbol (m/n) ist die wie folgt definierte Zahl aus $\{0, 1, -1\}$:

$$(m/n) := (m \mid p_1^{k_1} \cdots p_e^{k_e}) := (m/p_1)^{k_1} \cdot \ldots \cdot (m/p_e)^{k_e}$$

Beispiel 7.3

$$(11 \mid 91) = (11 \mid 7 \cdot 13) = (11/7) \cdot (11/13) = (4/7) \cdot (11/13) = 1 \cdot (-1) = -1$$

Denn: • $4 \equiv 2^2 \pmod 7$, also $(4/7) = 1$
• quadratische Reste modulo 13 sind: $1^2 = \underline{1}$, $2^2 = \underline{4}$, $3^2 = \underline{9}$, $4^2 = 16 \equiv \underline{3} \pmod{13}$,

$5^2 = 25 \equiv \underline{12} \pmod{13}$, $6^2 = 36 \equiv \underline{10} \pmod{13}$, $7^2 \equiv (-6)^2 \equiv 6^2 \equiv \underline{10} \pmod{13}$

$8^2 \equiv (-5)^2 \equiv 5^2 \equiv \underline{12} \pmod{13}$, $9^2 \equiv (-4)^2 \equiv 4^2 \equiv \underline{3} \pmod{13}$,

$10^2 \equiv (-3)^2 \equiv 3^2 \equiv \underline{9} \pmod{13}$, $11^2 \equiv (-2)^2 \equiv 2^2 \equiv \underline{4} \pmod{13}$, $12^2 \equiv (-1)^2 \equiv 1^2 \equiv \underline{1} \pmod{13}$

D.h. Die Zahlen 1, 3, 4, 9, 10, 12 sind sämtliche quadratischen Reste modulo 13. Insbes ist $(11|13) = -1$.

Aus der Zahlentheorie ist folgender Satz bekannt:

Satz 7.4

Sei n eine ungerade natürliche Zahl > 1.

(a) Falls n eine Primzahl ist, so gilt
für alle $m \in \{1, \dots, n-1\}$: $(m|n) \equiv m^{\frac{n-1}{2}} \pmod{n}$

(b) Falls n keine Primzahl ist, so gilt für
mindestens die Hälfte aller Zahlen $m \in \{2, \dots, n-1\}$
mit $ggT(m,n) = 1$, dass $(m|n) \not\equiv m^{\frac{n-1}{2}} \pmod{n}$.

Bemerkung: $ggT(m,n)$ bezeichnet hier den
größten gemeinsamen Teiler von m und n, d.h. die
größte natürliche Zahl g, so dass es natürliche Zahlen
m' und n' gibt mit $m = gm'$ und $n = gn'$.

(Hier ohne Beweis)

Unter Verwendung von Satz 7.4 können wir nun den zum
Beweis von Theorem 7.1 gesuchten randomisierten
Primzahltest angeben:

Beweis von Theorem 7.1:

Sei A der folgende randomisierte Algorithmus:

Eingabe: eine natürliche Zahl n
Frage: Ist n eine Primzahl?

Algorithmus:
Falls $n=0$ oder $n=1$, so STOPP mit Ausgabe "nein".
Falls $n=2$, so STOPP mit Ausgabe "ja"
Falls n eine gerade Zahl ist, so STOPP mit Ausgabe "nein"
Sonst (d.h.: n ist eine ungerade nat. Zahl >1):

Wähle zufällig eine Zahl $m \in \{2, \dots, n-1\}$.

Berechne $g := ggT(m,n)$

Falls $g \neq 1$, so STOPP mit Ausgabe "nein"
(denn: $g \neq n$, g ist ein Teiler von n und n somit keine Primzahl)

Sonst: Berechne $j := (m|n)$

Berechne $k := m^{\frac{n-1}{2}} \pmod n$

Falls $j \neq k$, so STOPP mit Ausgabe "nein"
(wegen Satz 7.4(a) ist n dann nämlich keine Primzahl)

Sonst: STOPP mit Ausgabe "ja".

Für diesen Algorithmus gilt:

- Falls n eine Primzahl ist, so gilt für jedes mögliche $m \in \{2, \dots, n-1\}$, dass $g = ggT(m,n) = 1$, und wegen Satz 7.4(a) gilt: $j = (m|n) \equiv m^{\frac{n-1}{2}} \equiv k \pmod n$.
  Daher ist – unabhängig von der konkreten Wahl von m – stets $A(n) =$ "ja".

- Falls n keine Primzahl ist, so gilt:
  - für alle $m \in \{2, \dots, n-1\}$ mit $g := ggT(m,n) \neq 1$ ist $A(n) =$ "nein"
  - für mindestens die Hälfte aller $m \in \{2, \dots, n-1\}$ mit $g = ggT(m,n) = 1$ ist $A(n) =$ "nein" (wegen Satz 7.4(b))

Insgesamt ist also für mindestens die Hälfte aller zufälligen Wahlen von m das Ergebnis $A(n) =$ "nein" — d.h. es gilt mit Wahrscheinlichkeit $\geq \frac{1}{2}$, dass $A(n) =$ "nein" ist.

Somit weist unser Algorithmus A das von Theorem 7.1 geforderte Verhalten auf.

Um die gewünschte Laufzeit von $poly(\log n)$ zu erhalten, benötigen wir Verfahren, die

(1) bei Eingabe der Binärdarstellung von n durch $poly(\log n)$ viele Münzwürfe die Binärdarstellung einer zufällig, gleichverteilt aus $\{2, \dots, n-1\}$ gewählten Zahl m erzeugen und dabei nur $poly(\log n)$ Berechnungsschritte machen

(2) bei Eingabe der Binärdarstellungen zweier Zahlen m und n in $poly(\log m + \log n)$ Berechnungsschritten die Binärdarstellung von $ggT(m,n)$ erzeugen

(3) bei Eingabe der Binärdarstellungen zweier Zahlen m und n (mit $n > 1$, ungerade, und $m \in \{2, \dots, n-1\}$) in $poly(\log m + \log n)$ Berechnungsschritten das Jacobi-Symbol $(m|n) \in \{0, 1, -1\}$ erzeugen

(4) bei Eingabe der Binärdarstellungen zweier Zahlen m und n mit $m \in \{2, \dots, n-1\}$ die Zahl $k := m^{\frac{n-1}{2}} \pmod n$ erzeugen.

zu (1): Übungsaufgabe.

zu (2): Hierzu verwenden wir den bekannten Euklidischen Algorithmus zur ggT-Berechnung:

ggT(m,n):

Falls $m = 0$, so STOPP mit Ausgabe n

Falls $n = 0$, so STOPP mit Ausgabe m

Sonst:

Falls $n \geq m$, so:

(*) berechne $r \in \{0,\dots,m-1\}$ mit $n \equiv r \pmod m$

STOPP mit Ausgabe ggT(m,r)

Sonst:

(**) berechne $r \in \{0,\dots,n-1\}$ mit $m \equiv r \pmod n$

STOPP mit Ausgabe ggT(r,n)

Beispiel:

$$ggT(51,91) = ggT(51,40) = ggT(11,40) = ggT(11,7)$$
$$= ggT(4,7) = ggT(4,3) = ggT(1,3)$$
$$= ggT(1,0) = 1$$

Laufzeitanalyse: Man sieht leicht, dass bei (*) $r \leq \frac{m}{2}$ ist (denn: $n = k \cdot m + r$ mit $k \geq 1$, $0 \leq r < m$) und bei (**) $r \leq \frac{m}{2}$ ist.

Daher wird in jedem Rekursionsschritt die größere der beiden Zahlen halbiert und daher ist die rekursive Berechnung von ggT(m,n) nach $O(\log m + \log n)$ Rekursionsschritten beendet.

Die für (*) bzw (**) nötige Division mit Rest ($r := n \bmod m$ bzw. $r := m \bmod n$) kann in Zeit

poly($\log m + \log n$) durchgeführt werden (Details: Übung!)

Zu (3): Zur Berechnung des Jacobi-Symbols $(m|n)$ können wir ähnlich wie bei (2) vorgehen und dabei folgende aus der Zahlentheorie bekannte Rechenregeln (hier ohne Beweis) benutzen:

F.a. m, n mit n ungerade, $n > 1$ und $\text{ggT}(m,n) = 1$ gilt:

(a) $(1|n) = 1$

(b) $(2|n) = (-1)^{\frac{n^2-1}{8}}$

(c) Falls m gerade, so $(m|n) = (2|n) \cdot \left(\frac{m}{2}|n\right)$

(d) Falls $m > n$ ist, so gilt für $r := m \bmod n$:

$$(m|n) = (r|n)$$

(e) Falls $m \leq n$ ist, so gilt:

Falls $2 \mid (m-1)\cdot(n-1)$, so $(m|n) = (n|m)$

Sonst: $(m|n) = -(n|m)$

(sog. "quadratisches Reziprozitätsgesetz")

Details zum Algorithmus Jacobi(m,n), der in poly($\log m + \log n$) Berechnungsschritten das Jacobi-Symbol $(m|n)$ berechnet: Übung!

Zu (4): Übung!

Insgesamt erhalten wir, dass Algorithmus A stets nach poly($\log n$) Berechnungsschritten anhält. □