Sortieren in Netzwerken aus Stacks und Queues

Dem Institut für Mathematik
der Technischen Universität Berlin
vorgelegte

Bachelorarbeit

von
Sandro M. Roch
(Matrikelnummer 371629)

Mail: roch@math.tu-berlin.de

Erstgutachter:	Prof. Dr. Stefan Felsner
Zweitgutachter:	Prof. Dr. Martin Skutella
Tag der Einreichung:	12. Juli 2019

Eidesstattliche Versicherung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und eigenhändig sowie ohne unerlaubte fremde Hilfe und ausschließlich unter Verwendung der aufgeführten Quellen und Hilfsmittel angefertigt habe.

Berlin, den 12. Juli 2019

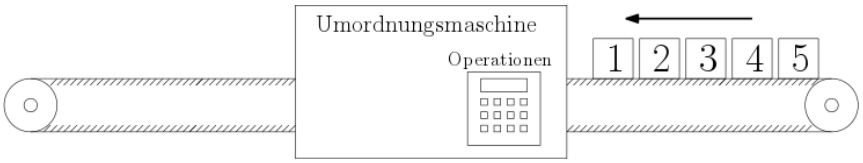
Sandro M. Roch

Inhaltsverzeichnis

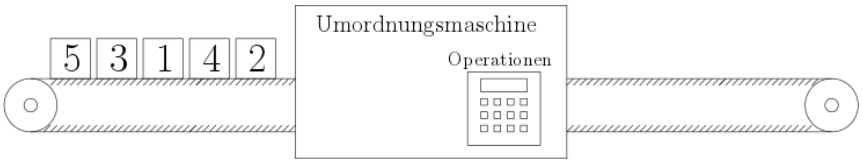
1. Einleitung

1.1. Motivation

Man stelle sich eine Maschine vor, die auf einem Fließband n Objekte annimmt, bestimmte Operationen durchführen kann, die die Reihenfolge unabhängig von den Objekten ändern, und sie dann auf der anderen Seite wieder ausgibt. Die Maschine vollführt also eine Umordnung, siehe beispielhaft Abbildung 1.1 mit $n = 5$.



(a) 5 Objekte vor der Umordnung



(b) 5 Objekte nach der Umordnung

Abbildung 1.1.: Illustration der „Umordnungsmaschine“

Wir bezeichnen mit $S_n := \{\pi : \{1, ..., n\} \to \{1, ..., n\} : \pi \text{ bijektiv}\}$ die Menge aller Permutationen π der Länge n, die wir auch als $\pi = [\pi(1), ..., \pi(n)]$ notieren. Formal kann man die neue Reihenfolge der Objekte als Permutation $\pi_w \in S_n$ auffassen, die durch eine Folge von Operationen w *erzeugt* wird, im Beispiel also $\pi_w = [5, 3, 1, 4, 2]$. Kennt man die Menge W aller Folgen von Operationen, die die Maschine ausführen kann, so stellt sich auf der einen Seite die Frage, welche Permutationen die Maschine erzeugen kann, also die Frage nach der Menge $\{\pi_w : w \in W\}$.

Stellt man nun für eine Permutation $\pi \in S_n$ die n Objekte nicht in ihrer natürlichen Reihenfolge von 1 bis n auf das Fließband vor die Maschine, sondern in der Reihenfolge $\pi(1), ..., \pi(n)$, und lässt die Maschine die Operationen $w \in W$ ausführen, so entspricht die Umordnung der Objekte der Rechtsmultiplikation von π mit π_w. Somit erhält man in der Ausgabe die Objekte in der Reihenfolge $(\pi \circ \pi_w)(1), ..., (\pi \circ \pi_w)(n)$. Nun

stellt sich auf der anderen Seite die Frage, welche Permutationen π von der Maschine *sortiert*, d.h. in die Reihenfolge $1, ..., n$ überführt werden können, also die Frage nach der Menge $\{\pi \in S_n : \exists w \in W : \pi \circ \pi_w = \mathrm{Id}_n\}$.

Beide Fragestellungen sind aufeinander rückführbar und in diesem Sinne äquivalent. Denn eine Permutation $\pi \in S_n$ kann genau dann sortiert werden, wenn es Operationen $w \in W$ so gibt, dass $\pi \circ \pi_w = \mathrm{Id}_n$, was äquivalent zu $\pi_w = \pi^{-1}$ ist, also genau dann, wenn π^{-1} erzeugt werden kann. Umgekehrt und analog dazu kann π genau dann erzeugt werden, wenn π^{-1} sortiert werden kann. Die Menge der sortierbaren Permutationen steht daher über die Invertierung mit der Menge der erzeugbaren Permutationen in Bijektion. Insbesondere stimmen sie in ihrer Anzahl überein. Von dieser Überlegung, die auf elegante Weise die Gruppensymmetrie ausnutzt, werden wir intensiven Gebrauch machen.

1.2. Weichennetzwerke

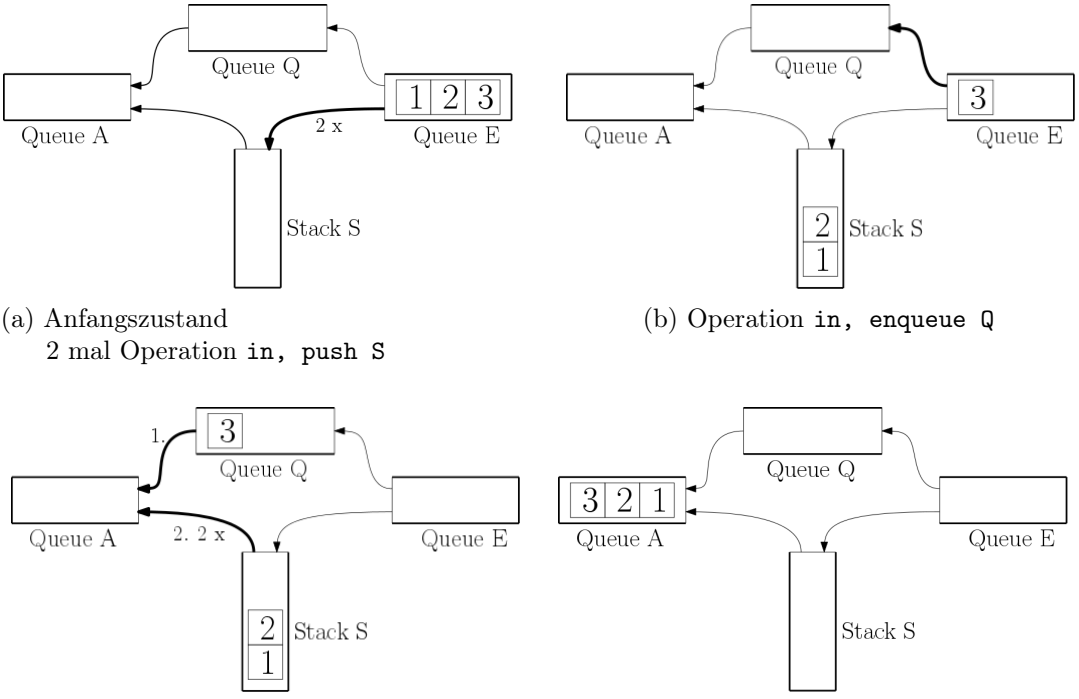
Um die im vorigen Abschnitt aufgeworfenen Fragestellungen zu konkretisieren, wenden wir uns nun Weichennetzwerken zu, die als spezielle „Umordnungsmaschinen“ im obigen Sinne aufgefasst werden können. Der Begriff von Weichennetzwerken (engl. switchyards) wurde erstmals von Tarjan eingeführt [Tar72].

Wie üblich bezeichnen wir als *Stack* (dt. Stapel) einen unbegrenzten Speicher von Elementen mit den Operationen `push` (Element hinzufügen) und `pop` (Element abheben), der nach dem Last-In-First-Out-Prinzip funktioniert. Ebenso bezeichnen wir als *Queue* (dt. Warteschlange) einen unbegrenzten Speicher von Elementen mit den Operationen `enqueue` (Element einreihen) und `dequeue` (Element entnehmen), der nach dem First-In-First-Out-Prinzip funktioniert.

Ein *Weichennetzwerk* (im Weiteren auch kurz *Netzwerk*) ist ein gerichteter Graph, dessen Knoten jeweils Stacks oder Queues und dessen Kanten mögliche Verschiebeoperationen zwischen diesen repräsentieren. Dabei gibt es zwei ausgezeichnete Queues, die Eingangsqueue E mit Eingangsgrad 0 und die Ausgangsqueue A mit Ausgangsgrad 0.

Von E laufen die n Elemente in einer vorgegebenen Reihenfolge in das Netzwerk ein. In jeder Operation des Weichennetzwerks wandert ein Element von einem Knoten über eine ausgehende Kante an einen anderen Knoten. Genau genommen besteht eine Operation aus zwei technischen Operationen: Am Startknoten, der ein nichtleerer Stack oder eine nichtleere Queue sein muss, wird die Operation `pop` bzw. `dequeue` ausgeführt und anschließend dieses Element dem Zielknoten, der auch ein Stack oder eine Queue ist, mittels der Operation `push` bzw. `enqueue` hinzugefügt. Das Abziehen

von Elementen aus E wird als *einlesen* (`in`) und das Einreihen von Elementen in A wird als *ausgeben* (`out`) bezeichnet. In einem Weichennetzwerk soll es außerdem stets möglich sein, durch weitere Operationen in den Zustand zu gelangen, in dem alle Elemente ausgegeben wurden. Die Reihenfolge der Elemente in A ist dann üblicherweise nicht die gleiche Reihenfolge, die die Elemente in E bildeten. Die Umordnung hängt von der Folge der Operationen ab, d.h. von der Information, zu welchem Zeitpunkt ein Element von welchem Startknoten zu welchem Zielknoten übertragen wird.



(a) Anfangszustand
2 mal Operation `in, push S`

(b) Operation `in, enqueue Q`

(c) Operation `dequeue Q, out`
2 mal Operation `pop S, out`

(d) Endzustand

Abbildung 1.2.: Beispielhafte Operationsfolge in einem Weichennetzwerk

Abbildung 1.2 zeigt die Ausführung einer möglichen Operationsfolge in einem Netzwerk bestehend aus Eingangsqueue E, Ausgangsqueue A, einer Queue Q und einem Stack S. Die hervorgehobenen Kanten sind die im nächsten Ausführungsschritt benutzten Kanten.

1.3. Anwendungen

Eine offensichtliche Anwendung der in dieser Arbeit untersuchten Theorie scheinen buchstäbliche Weichennetzwerke zu sein, wie sie etwa in Rangierbahnhöfen auftreten. Jeder Gleisabschnitt, der nur von einer Seite aus befahren werden kann, etwa ein

Abstellgleis oder Gleis eines Kopfbahnhofes, kann als Stack aufgefasst werden. Man beachte jedoch, dass Wagons in der Regel nicht autonom und in beide Richtungen fahren können, sondern einen Antriebswagen benötigen. Außerdem kann ein zweiseitig befahrbares Abstellgleis mitunter nicht als Queue, sondern nur als *Double-Ended Queue* modelliert werden. Beides geht über das in dieser Arbeit behandelte Modell des Weichennetzwerks hinaus.

Häufig lassen sich Weichennetzwerke aber im abstrakten Sinne in Fabriken finden: Objekte werden auf Fließbändern eingereiht (Queues) und auf Stapeln zur weiteren Verarbeitung zwischengelagert (Stacks). Zum Beispiel beschreiben Lübecke et al. Szenarien der Zwischenlagerung von Brammen in der Stahlproduktion [KLM+07].

Ferner treten speziell Netzwerke von Stacks in der Logistik von Lagern auf, etwa bei Containerterminals von Häfen. Hier werden Frachtcontainer z.B. von Schiffen auf Güterzüge umgeschlagen, wobei sie auf Stapeln zwischengelagert werden. Dabei gilt es, die Zahl der Verladungen von Containern zwischen den Stapeln zu minimieren, welche nötig werden könnten, um auf nicht oben liegende Container zuzugreifen.

1.4. Gliederung

In Kapitel 2 behandeln wir fundamentale Eigenschaften von Weichennetzwerken. Wir stellen unter anderem fest, dass zyklische Netzwerke alle Permutationen sortieren können. Daher untersuchen wir diese separat. In Kapitel 3 betrachten wir ausgewählte Instanzen von azyklischen Netzwerken und in Kapitel 4 ausgewählte Instanzen von zyklischen Netzwerken.

2. Eigenschaften von Weichennetzwerken

2.1. Sortierbare und erzeugbare Permutationen

Definition 2.1. Gegeben ein Weichennetzwerk N, dann heißt eine Permutation $\pi \in S_n$

1) *sortierbar in* N, falls die Elemente in der Reihenfolge $\pi(1), ..., \pi(n)$ von der Eingangsqueue kommend durch N laufen und in der Reihenfolge $1, ..., n$ ausgegeben werden können.

2) *erzeugbar in* N, falls die Elemente in der Reihenfolge $1, ..., n$ von der Eingangsqueue kommend durch N laufen und in der Reihenfolge $\pi(1), ..., \pi(n)$ ausgegeben werden können.

Die Beobachtung, die wir in Abschnitt 1.1 für „Umordnungsmaschinen“ gemacht haben, lässt sich nun auf Weichennetzwerke übertragen:

Lemma 2.2. *In einem Weichennetzwerk ist die Permutation $\pi \in S_n$ genau dann sortierbar, wenn π^{-1} erzeugbar ist, und π ist genau dann erzeugbar, wenn π^{-1} sortierbar ist. Die Mengen der sortierbaren und der erzeugbaren Permutationen der Länge n stehen über die Invertierung zueinander in Bijektion.*

2.2. Permutationsmuster

Zur Charakterisierung von Permutationen, die in einem gegebenen Netzwerk sortierbar oder erzeugbar sind, werden wir uns des Begriffs der Permutationsmuster bedienen.

Definition 2.3. Sei $\pi \in S_n$ eine Permutation.

1) Für Indizes $1 \leq i_1 < ... < i_k \leq n$ heißt die Sequenz $[\pi(i_1), ..., \pi(i_k)]$ *Teilsequenz* von π der Länge k.

2) Sei $S = [x_1, ..., x_k]$ eine Teilsequenz von π der Länge k und $\sigma \in S_k$. Die Teilsequenz S *realisiert* σ, falls S entsprechend σ angeordnet ist, d.h. für alle $1 \leq i < j \leq k$ gilt $x_i < x_j \iff \sigma(i) < \sigma(j)$.

3) Eine Permutation $\sigma \in S_k, k \leq n$ ist ein *Muster* von π, falls π eine Teilsequenz enthält, die σ realisiert. Wir schreiben in diesem Fall $\sigma \preceq_{\mathfrak{S}} \pi$.

4) Die Permutation π *vermeidet* $\sigma \in S_k$, wenn σ kein Muster von π ist.

Beispielsweise enthält die Permutation $\pi = [5, 2, 3, 4, 1]$ das Muster $\sigma = [3, 2, 1]$. Dieses wird durch die drei Teilsequenzen $[5, 2, 1]$, $[5, 3, 1]$ und $[5, 4, 1]$ realisiert. Dagegen vermeidet π das Muster $\sigma' = [4, 3, 2, 1]$, denn es gibt in π keine vier absteigend angeordneten Elemente. Man sieht leicht, dass die Relation $\preceq_{\mathfrak{S}}$ reflexiv, antisymmetrisch und transitiv ist, also eine Partialordnung der Menge $S_* := \cup_{n \in \mathbb{N}} S_n$ ist.

Die folgenden Lemmata benötigen wir für später.

Lemma 2.4. *Sei $\pi \in S_n, \sigma \in S_k, k \leq n$. Dann gilt $\sigma \preceq_{\mathfrak{S}} \pi \iff \sigma^{-1} \preceq_{\mathfrak{S}} \pi^{-1}$.*

Beweis. Sei $\sigma \preceq_{\mathfrak{S}} \pi$ und sei $[\pi(i_1), ..., \pi(i_k)]$ eine Teilsequenz von $[\pi(1), ..., \pi(n)]$, mit der σ realisiert wird. Für die Indizes der Teilsequenz gilt $i_1 < ... < i_k$ und für deren Elemente $\pi(i_{\sigma^{-1}(1)}) < ... < \pi(i_{\sigma^{-1}(k)})$.

Dann ist $[\pi^{-1}(\pi(i_{\sigma^{-1}(1)})), ..., \pi^{-1}(\pi(i_{\sigma^{-1}(k)}))] = [i_{\sigma^{-1}(1)}, ..., i_{\sigma^{-1}(k)}]$ eine Teilsequenz von π^{-1}, welche das Muster σ^{-1} realisiert.

Für die Rückrichtung wende man die Hinrichtung auf π^{-1} und σ^{-1} an. □

Lemma 2.5. *Ist in einem Weichennetzwerk N die Permutation $\pi \in S_n$ sortierbar bzw. erzeugbar, so auch jedes Muster $\sigma \preceq_{\mathfrak{S}} \pi$.*

Beweis. Sei w die Operationsfolge, die π in N sortiert bzw. erzeugt. Sei $\sigma \preceq_{\mathfrak{S}} \pi$ und seien $\pi(i_1), ..., \pi(i_k)$ die Elemente einer Teilsequenz, die in π das Muster σ realisiert. Löschen aller Operationen in w, bei denen Elemente im Weichennetzwerk verschoben werden, die nicht $\pi(i_1), ..., \pi(i_k)$ sind, liefert neue Operationsfolge w', die σ sortiert bzw. erzeugt. □

2.3. Zyklische und azyklische Weichennetzwerke

Ein Weichennetzwerk ist *zyklisch*, wenn es einen gerichteten und nicht isolierten Kreis enthält. Nicht isoliert bedeutet dabei, dass der Kreis von Elementen tatsächlich erreicht

werden kann. Es sei angemerkt, dass bei Tarjan Weichennetzwerke stets azyklisch sind. Der Einfachheit halber bezeichnen wir auch zyklische Netzwerke als Weichennetzwerke. Der Grund, wieso Tarjan sich auf azyklische Netzwerke beschränkt, liegt in seiner folgenden Beobachtung in [Tar72], die auch für uns eine Unterscheidung von zyklischen und azyklischen Netzwerken sinnvoll macht:

Theorem 2.6 (Tarjan)**.** *In einem Weichennetzwerk N sind genau dann alle Permutationen beliebiger Länge sortierbar bzw. erzeugbar, wenn N zyklisch ist. Andernfalls konvergieren die Wahrscheinlichkeiten*

$$\mathbb{P}[\pi \text{ erzeugbar in } N] \to 0$$
$$\mathbb{P}[\pi \text{ sortierbar in } N] \to 0$$

bei einer uniform zufällig gezogenen Permutation $\pi \in S_n$ mit $n \to \infty$.

Beweis. Enthält N einen Kreis, so ist jede Permutation erzeugbar. Man lässt alle Elemente von E kommend in den Kreis laufen und dort wiederholt im Kreis zirkulieren. Von einem Knoten des Kreises gibt es einen Weg zu A. Sobald das Element $\pi(1)$ an diesem Knoten vorbei kommt, lässt man es über diesen Weg ausgeben, und fährt so fort mit dem Element $\pi(2)$ usw.

Enthalte N keinen Kreis. Sei e die Zahl der Kanten und v die Zahl der Knoten in N. Jedes der n Elemente einer Permutation kann im Netzwerk höchstens v mal bewegt werden, bis es ausgegeben wird, da aufgrund der Kreisfreiheit jeder Knoten von einem Element nur höchstens einmal besucht werden kann. Zu jedem Zeitpunkt gibt es höchstens e viele mögliche Operationen. Insgesamt gibt es höchstens $e^{v \cdot n}$ mögliche Operationsfolgen, von denen jede eine Permutation erzeugt. Der Anteil erzeugbarer Permutationen ist somit beschränkt durch $\frac{e^{v \cdot n}}{n!} \to 0$ für $n \to \infty$.

Mit Lemma 2.2 überträgt sich die Argumentation auf sortierbare Permutationen. □

Für azyklische Netzwerke N ist die typische Fragestellung, welche Permutationen der Länge n sortierbar (bzw. erzeugbar) sind, und wie viele es gibt. Die Menge der in N sortierbaren Permutationen (beliebiger Länge) ist nach Lemma 2.5 bzgl. $\preceq_{\mathfrak{S}}$ nach unten abgeschlossen. Ein möglicher Ansatz zur Charakterisierung ist die Bestimmung bzgl. $\preceq_{\mathfrak{S}}$ minimaler nicht sortierbarer Permutationen. Eine Permutation π ist genau dann sortierbar, wenn sie diese vermeidet. Es kann jedoch unendlich viele minimale nicht sortierbare Permutationen geben, daher ist deren Bestimmung oft schwierig.

Für zyklische Weichennetzwerke dagegen wird die Frage interessant, wie viele Operationen nötig sind, um eine beliebige Permutation $\pi \in S_n$ zu sortieren bzw. zu erzeugen und wie die zugehörigen minimalen Operationsfolgen gefunden werden können. Die minimale Zahl an Operationen, die benötigt wird, um in einem Weichennetzwerk jede Permutation $\pi \in S_n$ zu sortieren, wird als dessen *Komplexität* bezeichnet. Dies

bezeichnet hier also nicht etwa die Komplexität des algorithmischen Problems, eine minimale Operationsfolge zu berechnen.

2.4. Mitternachtseinschränkung

Der Begriff des Weichennetzwerks lässt sich auf viele Arten erweitern, etwa durch weitere Speichertypen wie Deques oder durch weitere Einschränkungen, wie zum Beispiel dass eine Kante nur benutzt werden darf, wenn bestimmte Stacks oder Queues leer sind. Eine solche Einschränkung ist die sogenannte *Mitternachts-Einschränkung* (engl. midnight-constraint), die wir betrachten werden. Wird diese bei einem Weichennetzwerk gefordert, so müssen zunächst alle Elemente eingelesen werden, bevor Elemente ausgegeben werden dürfen. Der Begriff ist dabei von König und Lübbecke [KL08] entnommen. Er stammt von einem Anwendungsbeispiel: Ein Zug erreicht spät abends einen Bahnhof, wo seine Wagons schnell in einem Rangierwerk abgestellt und erst am nächsten Tag zur Abfahrt auf dem Ausfahrtgleis zusammengestellt werden.

Die Mitternachtseinschränkung beeinflusst bei azyklischen Weichennetzwerken, welche Permutationen sortierbar bzw. erzeugbar sind, und bei zyklischen Weichennetzwerken die Anzahl benötigter Operationen zum Sortieren bzw. Erzeugen einer bestimmten Permutation.

Lemma 2.7. *Sei N ein zyklisches Weichennetzwerk. Beträgt die Komplexität zum Sortieren einer Permutation $\pi \in S_n$ in N mit Mitternachtseinschränkung $T(n)$, so beträgt die Komplexität zum Sortieren einer Permutation $\pi \in S_n$ ohne Mitternachtseinschränkung mindestens $T(n-1)$.*

Beweis. Sei $\pi \in S_{n-1}$ eine Permutation, die zum Sortieren in N mit Mitternachtseinschränkung $T(n-1)$ Operationen benötigt.

Betrachte $\tilde{\pi} := [\pi(1)+1, ..., \pi(n-1)+1, 1]$. Jede Operationsfolge zum Sortieren von $\tilde{\pi}$ liefert durch Löschen aller Operationen, die das Element 1 verschieben, eine Operationsfolge, die π unter Einhaltung der Mitternachtseinschränkung sortiert. □

Lemma 2.7 bedeutet, dass sich in einem zyklischen Netzwerk die Komplexitäten zum Sortieren mit und ohne Mitternachtseinschränkung asymptotisch nicht unterscheiden, so lange $T(n)$ zumindest polynomiell beschränkt ist. Dies vereinfacht die Analyse.

3. Spezielle Instanzen von azyklischen Weichennetzwerken

3.1. Einzelner Stack

Wir betrachten das in Abbildung 3.1 dargestellte Weichennetzwerk, bestehend nur aus E, A und einem einzelnen Stack S.

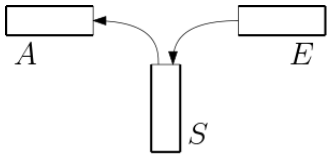


Abbildung 3.1.: Weichennetzwerk mit einzelnem Stack

Noch ohne den allgemeinen Begriff eines Weichennetzwerks im Sinn zu haben, wurde dieses Weichennetzwerk bereits 1973 von Knuth in seinem Buch *The Art of Computer Programming* in Übungsaufgaben erwähnt [Knu75, ch. 2.2.1, p. 243]. Die in den Aufgaben zu zeigenden Aussagen bilden im Wesentlichen die folgenden beiden Theoreme.

Theorem 3.1 (Knuth)**.** *In einem Weichennetzwerk bestehend aus einem einzelnen Stack ist eine Permutation $\pi \in S_n$ genau dann erzeugbar, wenn sie das Muster $[3, 1, 2]$ vermeidet.*

Wir verzichten an dieser Stelle auf den einfachen Beweis, da Theorem 3.1 später direkt aus dem allgemeineren Theorem 3.8 folgt, siehe anschließende Bemerkung dort.

Die *Catalan-Zahlen*, definiert für $n \in \mathbb{N}$ durch $C_n := \frac{1}{n+1}\binom{2n}{n}$ sind eine wichtige Folge von Zahlen, die verschiedene kombinatorische Objekte zählen, zum Beispiel Triangulierungen eines konvexen Polygons, volle Binärbäume und sogenannte Dyck-Pfade. Dyck-Pfade der Länge n, benannt nach Walther von Dyck, sind Pfade auf der Menge $\mathbb{Z}$, die bei 0 beginnen und enden, in jedem der n Schritte um $+1$ oder -1 wandern, jedoch niemals negative Werte annehmen. Beispielsweise stellt Abbildung 3.2 die beiden Dyck-Pfade der Länge 4 über einer Zeitachse dar. Allgemein ist die Anzahl der Dyck-Pfade der Länge $2n$ gegeben durch C_n. Für einen Beweis hierfür sowie für eine allgemeine Einführung in die Catalan-Zahlen sei auf [Sta15] verwiesen.

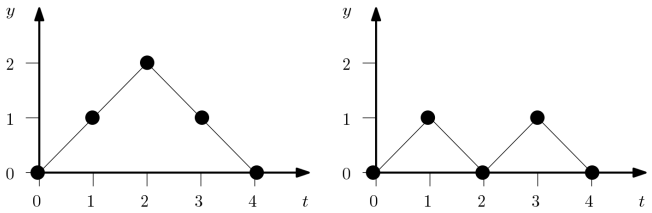


Abbildung 3.2.: Dyck-Pfade der Länge 4

Theorem 3.2 (Knuth)**.** *Die Anzahl der durch einen einzelnen Stack erzeugbaren Permutationen $\pi \in S_n$ ist gegeben durch die n-te Catalan-Zahl $C_n = \frac{1}{n+1}\binom{2n}{n}$.*

Beweis. Jede mögliche Operationsfolge des Weichennetzwerks besitzt Länge $2n$, genauso viele `out`- wie `in`-Operationen und zu jedem Zeitpunkt mindestens so viele vergangene `in`-Operationen wie vergangene `out`-Operationen. Man sieht, dass die Menge der Dyck-Pfade der Länge $2n$ mit der Menge möglicher Operationsfolgen in Bijektion steht, wenn die horizontale Achse die Zeitachse darstellt und die Höhe zu jedem Zeitpunkt die Zahl der Elemente im Stack ist.

Wenn $\pi \in S_n$ erzeugbar ist, dann gibt es genau eine Operationsfolge, die π erzeugt, da in jedem Zustand die nächste Operation bestimmt ist: Wurden etwa $\pi(1), ..., \pi(i-1)$ bereits ausgegeben, so muss im Falle, dass $\pi(i)$ oben auf dem Stack liegt, sofort `out` ausgeführt werden, und andernfalls mangels Alternativen `in` ausgeführt werden. Die erzeugbaren Permutationen stehen daher mit der Menge der möglichen Operationsfolgen und somit mit den Dyckpfaden der Länge $2n$ in Bijektion, wobei letztere von den Catalan-Zahlen gezählt werden. □

Korollar 3.3. *In einem Weichennetzwerk bestehend aus einem einzelnen Stack ist eine Permutation $\pi \in S_n$ genau dann sortierbar, wenn sie das Muster $[2, 3, 1]$ vermeidet. Die Anzahl der sortierbaren Permutationen der Länge n ist die n-te Catalan-Zahl.*

Beweis. Es ist $[3, 1, 2]^{-1} = [2, 3, 1]$. Daher sind die Inversen der $[3, 1, 2]$-vermeidenden Permutationen nach Lemma 2.4 genau die $[2, 3, 1]$-vermeidenden Permutationen. Die Aussage folgt aus Lemma 2.2, Theorem 3.1 und Theorem 3.2. □

3.2. Parallele Stacks

Wir betrachten das in Abbildung 3.3 dargestellte Weichennetzwerk, bestehend aus m parallel geschalteten Stacks. Wir untersuchen die Frage nach den sortierbaren und erzeugbaren Permutationen zunächst unter der im vorigen Kapitel erwähnten Mitternachtseinschränkung und anschließend ohne diese.

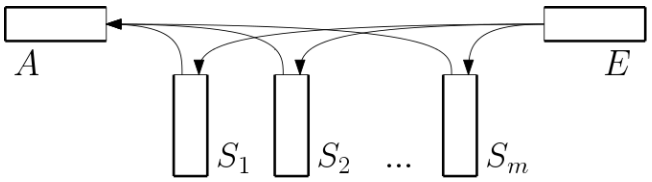
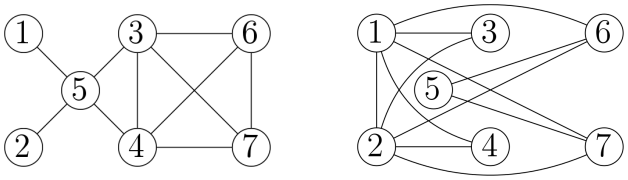


Abbildung 3.3.: Weichennetzwerk bestehend aus m parallelen Stacks

3.2.1. Parallele Stacks mit Mitternachtseinschränkung

Für eine natürliche Zahl m heißt ein Graph $G = (V, E)$ mit Knotenmenge V und Kantenmenge E *m-färbbar*, falls es eine Abbildung (*m-Färbung*) $\chi : V \to \{1, ..., m\}$ so gibt, dass $\chi(u) \neq \chi(v)$ für alle adjazenten Knotenpaare $\{u, v\} \in E$ gilt. Die *Chromatische Zahl* $\chi(G)$ bezeichnet die kleinste natürliche Zahl m, so dass G m-färbbar ist. Eine Teilmenge $C \subset V$ heißt *Clique* von G der Größe $|C|$ ($|C|$-Clique), falls $\{u, v\} \in E$ für alle $u, v \in C, u \neq v$ gilt. Die Größe der größten Clique in G heißt *Cliquenzahl* und wird mit $\omega(G)$ bezeichnet.

Für $\pi \in S_n$ betrachten Even und Itai in [EI71] den Graphen $G_\pi = (V_\pi, E_\pi)$ mit Knoten $V_\pi = \{1, ..., n\}$ und Kanten $E_\pi = \big\{\{i, j\} : 1 \leq i < j \leq n, \pi^{-1}(i) > \pi^{-1}(j)\big\}$, die Menge der *Fehlstände* von π. Solche Graphen werden *Permutationsgraphen* genannt. Ferner bezeichne $G_\pi^{\complement} = (V_\pi, E_\pi^{\complement})$ mit $E_\pi^{\complement} = \big\{\{i, j\} : 1 \leq i < j \leq n, \pi^{-1}(i) < \pi^{-1}(j)\big\}$ das Komplement dieses Graphen. Anders als Even und Itai fassen wir beide Graphen der Einfachheit halber zunächst als ungerichtet auf, sofern nicht anders erwähnt. Abbildung 3.4 zeigt ein Beispiel.



(a) Der Graph G_π (b) Der Graph $G_\pi^{\complement}$

Abbildung 3.4.: Beispiel für $\pi = [5, 1, 2, 7, 6, 4, 3]$

Fordert man für das Erzeugen einer Permutation $\pi \in S_n$ mit m parallelen Stacks, dass zunächst alle Elemente eingelesen werden, bevor Elemente ausgegeben werden (Mitternachtseinschränkung), so macht man folgende Beobachtung: Zwei Elemente, die über denselben Stack laufen, werden in umgekehrter Reihenfolge ausgegeben. Falls daher für $1 \leq i < j \leq n$ auch $\pi^{-1}(i) < \pi^{-1}(j)$ gilt, so müssen notwendigerweise die Elemente i und j über verschiedene Stacks laufen, um sie in der Reihenfolge π auszugeben. Der Graph $G_\pi^{\complement}$ enthält genau diese Paare $\{i, j\}$ als Kanten, ist also der Konfliktgraph von Knoten, die nicht zusammen über denselben Stack laufen dürfen.

Die m-Färbbarkeit von $G_\pi^{\complement}$ ist daher notwendig, um π erzeugen zu können. Folgendes Theorem von Even und Itai [EI71] zeigt, dass sie auch hinreichend ist.

Theorem 3.4 (Even, Itai)**.** *In einem Weichennetzwerk bestehend aus m parallelen Stacks, für das man fordert, dass zuerst alle Elemente der Eingangsqueue in die Stacks übertragen werden müssen, bevor aus den Stacks ein Element ausgegeben werden darf, ist eine Permutation $\pi \in S_n$ genau dann erzeugbar, wenn der Graph $G_\pi^{\complement}$ m-färbbar ist.*

Beweis. Falls π mit einer Operationsfolge erzeugt werden kann, so ist $G_\pi^{\complement}$ m-färbbar, siehe oben.

Existiert umgekehrt eine m-Färbung von $G_\pi^{\complement}$ und ordnet man den m Farben die m Stacks zu, so lässt sich π wie folgt erzeugen: Man legt die Elemente $1, ..., n$ von E kommend auf die Stacks ihrer Farbe in $G_\pi^{\complement}$ entsprechend. Nun können die Elemente $\pi(1), ..., \pi(n)$ in dieser Reihenfolge den Stacks entnommen und ausgegeben werden. Denn angenommen $\pi(1)$, ..., $\pi(i-1)$ wurden schon ausgegeben und $\pi(i)$ wird in dessen Stack von $\pi(j)$ überdeckt. Dann ist $\pi^{-1}(\pi(i)) = i < j = \pi^{-1}(\pi(j))$, da $\pi(j)$ noch nicht ausgegeben wurde, und $\pi(i) < \pi(j)$, da $\pi(j)$ von E nach $\pi(i)$ geladen worden sein muss. Dann ist $\{\pi(i), \pi(j)\} \in E_\pi^{\complement}$ im Widerspruch dazu, dass $\pi(i)$ und $\pi(j)$ im selben Stack sind. □

Even und Itai untersuchen stets nur erzeugbare Permutationen. Wir können ihr Resultat jedoch auf Sortierbarkeit übertragen:

Korollar 3.5. *In einem Weichennetzwerk bestehend aus m parallelen Stacks, für das man fordert, dass zuerst alle Elemente der Eingangsqueue in die Stacks übertragen werden müssen, bevor aus den Stacks ein Element ausgegeben werden darf, ist eine Permutation $\pi \in S_n$ genau dann sortierbar, wenn sie erzeugbar ist.*

Beweis. Eine Permutation $\pi \in S_n$ ist nach Lemma 2.2 und Theorem 3.4 genau dann sortierbar, wenn $G_{\pi^{-1}}^{\complement}$ m-färbbar ist. Für die Abbildung $\phi : V_\pi \to V_\pi, x \mapsto \pi(x)$ gilt, wie man leicht sieht, dass $\{i, j\} \in E_{\pi^{-1}} \iff \{\phi(i), \phi(j)\} \in E_\pi$. Damit ist ϕ ein Graphenisomorphismus zwischen $G_{\pi^{-1}}$ und G_π und damit auch ein Graphenisomorphismus zwischen $G_{\pi^{-1}}^{\complement}$ und $G_\pi^{\complement}$. Daher ist $G_{\pi^{-1}}^{\complement}$ genau dann m-färbbar, wenn $G_\pi^{\complement}$ m-färbbar ist, was nach Theorem 3.4 genau dann der Fall ist, wenn π erzeugbar ist. □

Berechnung der benötigen Zahl an Stacks

Nach dem eben gezeigten ist die benötigte Zahl an Stacks, um π zu erzeugen, die chromatische Zahl $\chi(G_\pi^{\complement})$. Zur Berechnung hilft uns, dass $G_\pi^{\complement}$ als entsprechend gerichteter

Graph genau der Vergleichbarkeitsgraph der folgenden Partialordnung auf V_π ist:

$$i \prec j \quad :\Leftrightarrow \quad i < j \ \wedge \ \pi^{-1}(i) < \pi^{-1}(j)$$

Als Schnitt zweier Partialordnungen ist dies in der Tat eine Partialordnung, und orientiert man jede Kante $\{i, j\} \in G_\pi^{\complement}$ mit $i < j$ als (i, j), so sind die Paare $i \prec j$ genau die gerichteten Kanten $(i, j) \in E_\pi^{\complement}$. Bekanntlich gilt für solche Ordnungsgraphen $\omega = \chi$. Folgender Algorithmus findet eine minimale Färbung von $G_\pi^{\complement}$. Dieser Algorithmus entspringt einer allgemeinen Methode, in einem solchen Schnitt von Partialordnungen eine Antikettenzerlegung bzw. eine längste Kette zu finden.

Eingabe : $\pi \in S_n$
Ausgabe : Minimale Färbung $\chi : V_\pi \to \{1, ..., n\}$

Initialisiere BENUTZT$[i] \leftarrow$ `FALSE`, $S[i] \leftarrow$ `NULL` für alle Farben $i = 1, ..., n$;
Initialisiere $\chi(i) \leftarrow$ `NULL` für alle Knoten $i = 1, ..., n$;

for $i = 1, ..., n$ **do**
  if *Es gibt k mit $S[k] > \pi(i)$* **then**
    Wähle kleinstes k mit $S[k] > \pi(i)$;
  else
    Wähle kleinstes k mit BENUTZT$[k] =$ `FALSE`;
    BENUTZT$[k] \leftarrow$ `TRUE`;
  end
  $\chi(\pi(i)) \leftarrow k$;
  $S[k] \leftarrow \pi(i)$;
end

Algorithmus 1 : Algorithmus zur Berechnung einer minimalen Färbung von $G_\pi^{\complement}$

Theorem 3.6. *Algorithmus 1 arbeitet korrekt und kann in Laufzeit $\mathcal{O}(n \log n)$ implementiert werden.*

Beweis. Der Algorithmus färbt nacheinander die Knoten $\pi(1), ..., \pi(n)$. Benutzte Farben werden im Array `BENUTZT` markiert. Man beachte, dass nach jeder Iteration der `for`-Schleife für jede benutzte Farbe k gilt: Alle mit Farbe k gefärbten Knoten $\pi(i)$ erfüllen $S[k] \leq \pi(i)$. In $G_\pi^{\complement}$ adjazente Knoten bekommen dann niemals dieselbe Farbe. Denn falls in zwei Iterationen $i < j$ beidesmal $\chi(\pi(i)) = k$ bzw. $\chi(\pi(j)) = k$ gesetzt wird, so gilt zu Beginn der j-ten Iteration $\pi(j) < S[k] \leq \pi(i)$, daher ist $\{\pi(i), \pi(j)\}$ keine Kante in $G_\pi^{\complement}$.

Die Färbung ist minimal: Sei r die Zahl der benutzten Farben. Wir zeigen, dass es $1 \leq i_1 < ... < i_r \leq n$ mit $\pi(i_1) < ... < \pi(i_r)$ gibt, denn dann bilden die Knoten $\pi(i_1), ..., \pi(i_r)$ in $G_\pi^{\complement}$ eine r-Clique. Man beobachte dafür, dass nach jeder Iteration die Werte von S für benutzte Farben streng monoton steigen. Sei nun $\pi(i_r)$ beliebiges mit Farbe r gefärbtes Element. In der i_r-ten Iteration wurde Farbe r gewählt, daher muss

$S[r-1] < \pi(i_r) < S[r]$ gegolten haben und somit muss in einer früheren Iteration $i_{r-1} < i_r$ die Variable $S[r-1]$ auf $\pi(i_{r-1}) < \pi(i_r)$ gesetzt worden sein, als $\pi(i_{r-1})$ mit Farbe $r-1$ gefärbt wurde. Iteriere dieses Argument.

Mittels von Suchbäumen kann die Wahl des kleinsten k mit $S[k] > \pi(i)$ einschließlich der Verzweigung selbst in Laufzeit $\mathcal{O}(\log n)$ implementiert werden. Dadurch ergibt sich eine Gesamtlaufzeit von $\mathcal{O}(n \log n)$. □

Aus der minimalen Färbung lässt sich eine Operationsfolge bestimmen, die π mittels $\chi(G_\pi^{\complement})$ Stacks erzeugt. Ist man dagegen nur an der minimalen Anzahl der benötigten Stacks interessiert, so genügt es, $\omega(G_\pi^{\complement}) = \chi(G_\pi^{\complement})$ zu bestimmen. Cliquen in $G_\pi^{\complement}$ sind genau die aufsteigenden Teilsequenzen von π. Die Zahl der benötigten Stacks ist also die Länge der längsten aufsteigenden Teilsequenz in π, im Folgenden als $L(\pi)$ bezeichnet. Hunt und Szymanski zeigen, dass $L(\pi)$ unter Verwendung von Zeigermaschinen in Laufzeit $\mathcal{O}(n \log \log n)$ bestimmt werden kann [HS77].

Es wurde intensiv erforscht, wie $L(\pi)$ für eine zufällige Permutation $\pi \in S_n$ verteilt ist. Für den Erwartungswert $\mathbb{E}(L(\pi))$ einer uniform zufällig gewählten Permutation $\pi \in S_n$ gilt:

$$\lim_{n\to\infty} \frac{\mathbb{E}(L(\pi))}{\sqrt{n}} = 2$$

Außerdem konvergiert $\frac{L(\pi)}{\sqrt{n}} \to 2$ für $n \to \infty$ in Wahrscheinlichkeit. Für diese Aussagen sei auf [Rom15, ch. 1, pp. 7-68] verwiesen.

3.2.2. Parallele Stacks ohne Mitternachtseinschränkung

Wenn zum Erzeugen von $\pi \in S_n$ auch Elemente ausgegeben werden dürfen, bevor alle eingelesen wurden, so müssen Elemente $1 \leq i < j \leq n$ mit $\pi^{-1}(i) < \pi^{-1}(j)$ nicht notwendigerweise über verschiedene Stacks laufen, da i ausgegeben werden kann, bevor j auf denselben Stack gelegt wird. Falls es jedoch ein Element $k > i, j$ mit $\pi^{-1}(k) < \pi^{-1}(i), \pi^{-1}(j)$ gibt, so wird erzwungen, dass sich i, j gleichzeitig in den Stacks befinden. In diesem Fall können i, j weiterhin nicht über denselben Stack laufen. Der Konfliktgraph $G_\pi^{\complement}$ aus dem vorigen Abschnitt muss daher abgeschwächt werden: Für $\pi \in S_n$ definiert man den Konfliktgraphen $H_\pi = (V_\pi, E_\pi)$ mit Knoten $V_\pi = \{1, ..., n\}$ und Kanten

$$E_\pi = \big\{\{i, j\} : \exists k : 1 \leq i < j < k \leq n : \pi^{-1}(k) < \pi^{-1}(i) < \pi^{-1}(j)\big\}$$

Um H_π zu konstruieren, betrachtet man alle Teilsequenzen von π der Länge 3, die das Muster $[3, 1, 2]$ realisieren, und setzt jeweils eine Kante zwischen dem zweiten und

dritten Element der Teilsequenz. Auf diese Weise erhält man alle Kanten von H_π, eventuell jedoch mehrfach. Abbildung 3.5 zeigt ein Beispiel. Auch hier zeigen Even und Itai in [EI71] durch folgendes Theorem, dass die m-Färbbarkeit von H_π bereits hinreichend für die Sortierbarkeit von π ist.

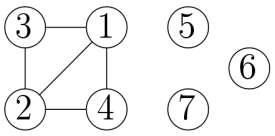


Abbildung 3.5.: Der Graph H_π beispielhaft für $\pi = [5, 1, 2, 7, 6, 4, 3]$

Theorem 3.7 (Even, Itai)**.** *In einem Weichennetzwerk bestehend aus m parallelen Stacks, in dem auch bevor alle Elemente in die Stacks übertragen wurden Elemente ausgegeben werden dürfen, ist eine Permutation $\pi \in S_n$ genau dann erzeugbar, wenn der Graph H_π m-färbbar ist.*

Beweis. Nach Konstruktion müssen in H_π adjazente Knoten über verschiedene Stacks laufen. Dafür, dass π erzeugt werden kann, ist die m-Färbbarkeit von H_π notwendig.

Existiere umgekehrt eine m-Färbung von H_π. Dann kann π durch folgenden Algorithmus erzeugt werden: Für $i = 1, ..., n$ wird in der i-ten Iteration $\pi(i)$ ausgegeben. Wenn dabei $\pi(i)$ auf einem der m Stacks zuoberst liegt, so wird $\pi(i)$ sofort ausgegeben. Andernfalls werden so lange Elemente von E in den durch ihre Farbe bestimmten Stack geladen, bis $\pi(i)$ auf einem der Stacks oben liegt und daraufhin ausgegeben wird.

Angenommen, der Algorithmus schlägt in der i-ten Iteration fehl. Dies ist genau dann der Fall, wenn zu Beginn der i-ten Iteration sich $\pi(i)$ schon in einem der Stacks befindet, jedoch von mindestens einem anderen Element überdeckt wird, etwa von $\pi(j)$. Es ist dann $\pi(j) > \pi(i)$, da $\pi(j)$ nach $\pi(i)$ geladen worden sein muss, und $i < j$, da im Falle $j < i$ das Element $\pi(j)$ schon in der früheren j-ten Iteration hätte ausgegeben worden sein müssen. Sei die k-te Iteration jene, in der $\pi(j)$ auf den Stack (über $\pi(i)$) gelegt wurde. In dieser wurde $\pi(k)$ nach $\pi(i)$ und $\pi(j)$ von E geladen und vor $\pi(i)$ und $\pi(j)$ ausgegeben, daher ist $\pi(k) > \pi(i), \pi(j)$ und $k < i, j$. Insgesamt also $\pi(i) < \pi(j) < \pi(k)$ und $k < i < j$ bzw. $\pi^{-1}(\pi(k)) < \pi^{-1}(\pi(i)) < \pi^{-1}(\pi(j))$. Doch dann ist $\{\pi(i), \pi(j)\} \in E_\pi$ im Widerspruch dazu, dass $\pi(i)$ und $\pi(j)$ im selben Stack sind. □

Theorem 3.7 ist eine Verallgemeinerung von Theorem 3.1 (Weichennetzwerke bestehend aus einem einzelnen Stack). Denn mit $m = 1$ ist der Graph H_π genau dann m-färbbar, falls er keine Kanten enthält, was genau dann zutrifft, wenn π das Muster $[3, 1, 2]$ vermeidet.

Schranken an die benötigte Zahl an Stacks

Wir untersuchen im Folgenden die benötigte Anzahl an parallelen Stacks, um $\pi \in S_n$ zu erzeugen (ohne Mitternachtseinschränkung), also nach Theorem 3.7 die chromatische Zahl $\chi(H_\pi)$.

Lemma 3.8. *Sei $n \geq 2$. In einem Weichennetzwerk bestehend aus m parallelen Stacks, in dem auch bevor alle Elemente in die Stacks übertragen wurden Elemente ausgegeben werden dürfen, können genau dann alle Permutationen $\pi \in S_n$ erzeugt werden, wenn $m \geq n-1$.*

Beweis. Klar ist: Mit $n-1$ Stacks können alle Permutationen $\pi \in S_n$ erzeugt werden. Im Allgemeinen genügen $n-2$ Stacks hingegen nicht: Betrachte $\pi = [n, 1, 2, ..., n-1]$. Dann ist in H_π der Knoten n isoliert und die Knoten $1, ..., n-1$ bilden eine Clique. Dann kann H_π nicht mit $n-2$ (oder weniger) Farben gefärbt werden, also nach Theorem 3.7 π nicht erzeugt werden. □

Die zum Erzeugen einer Permutation $\pi \in S_n$ benötigte Zahl an Stacks liegt demnach zwischen 1 und $n-1$. Wir versuchen im Folgenden diese Schranke zu verbessern unter der zusätzlichen Annahme, dass $\omega(H_\pi)$ klein ist. Dazu nutzen wir die Beobachtung von Even und Itai, dass H_π der speziellen Klasse der Überlappungsgraphen angehört.

Definition 3.9. Ein Graph $G = (V, E)$ mit $V = \{v_1, ..., v_n\}$ ist ein *Überlappungsgraph* (engl. Overlap-Graph), falls es eine Abbildung $\chi : V \to \{[a, b] \subset \mathbb{R} : a \leq b\}, v_i \mapsto [a_i, b_i]$ gibt mit $\{v_i, v_j\} \in E \iff a_i < a_j < b_i < b_j$.

In Überlappungsgraphen können die Knoten also als abgeschlossene Intervalle dargestellt werden, wobei zwei Knoten genau dann adjazent sind, wenn sich deren Intervalle echt überlappen, d.h. wenn sie sich schneiden, aber nicht ineinander enthalten sind. Even und Itai stellen in [EI71] fest, dass H_π ein sog. *Kreisgraph* ist, d.h. seine Knoten lassen sich als Sehnen auf einem Kreis darstellen, wobei zwei Knoten adjazent sind, wenn sich die Sehnen schneiden. Kreisgraphen sind bekannterweise genau die Überlappungsgraphen. Wir nutzen im folgenden ihre Idee, H_π als Kreisgraph darzustellen, um H_π als Überlappungsgraph darzustellen.

Für $\pi \in S_n$ heiße ein Knoten i in H_π *künstlich*, falls es kein $i < j \leq n$ gibt mit $\pi^{-1}(j) < \pi^{-1}(i)$. Künstliche Knoten sind isolierte Knoten, da sie nach Definition von H_π keine Kanten haben können. Man beachte, dass $\pi(1)$ und n immer künstliche Knoten in H_π sind. Die künstlichen Knoten bilden in π eine aufsteigende Teilsequenz. Im Beispiel von Abbildung 3.5 sind die Knoten 5 und 7 künstlich. Der Knoten 6 ist isoliert, jedoch nicht künstlich. Wir geben nun einen Algorithmus an, der für H_π eine Darstellung als Überlappungsgraph erzeugt.

Eingabe : $\pi \in S_n$
Ausgabe : Intervalle $[a_i, b_i] \subset \mathbb{R}$ einer Darstellung von H_π als Überlappungsgraph, in der Knoten i in H_π durch Intervall $[a_i, b_i]$ dargestellt wird

Setze $b_i \leftarrow \pi^{-1}(i)$ für alle $i = 1, ..., n$;
Setze $i \leftarrow 1$, $k \leftarrow \pi(1)$;

while *TRUE* **do**
  while $i < k$ **do**
    Wähle a_i beliebig aus $(b_k, b_k + 1)$, jedoch größer als alle anderen a's, die schon aus $(b_k, b_k + 1)$ gewählt wurden;
    $i \leftarrow i + 1$;
  end
  $a_k \leftarrow b_k$;
  if $k = n$ **then**
    BREAK;
  else
    $k \leftarrow$ Nächster künstlicher Knoten in π nach k;
  end
  $i \leftarrow i + 1$;
end

Algorithmus 2 : Algorithmus zum Erzeugen einer Darstellung von H_π als Überlappungsgraph

Beispiel 3.10. Wir betrachten noch einmal das Beispiel $\pi = [5, 1, 2, 7, 6, 4, 3]$ mit den künstlichen Knoten 5 und 7. Abbildung 3.6 zeigt die Ausgabe des Algorithmus auf einer nicht maßstabsgetreuen reellen Zahlenachse. Darüber sind die nichtleeren Intervalle eingezeichnet.

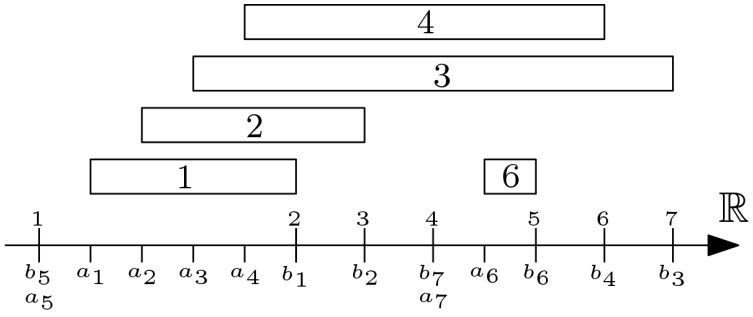


Abbildung 3.6.: Überlappungsgraph H_π für $\pi = [5, 1, 2, 7, 6, 4, 3]$

Lemma 3.11.

1) In Algorithmus 2 durchläuft Variable k alle künstlichen Knoten in ihrer Reihenfolge in π, d.h. aufsteigend, und es wird jeweils $a_k = b_k$ gesetzt. Nach Ausführung gilt $a_i < a_j$ für alle nicht künstlichen Knoten $i < j$.

2) Es gilt $a_i \leq b_i$ f.a. $i = 1, ..., n$.

3) Für jede Kante $\{i, j\} \in E_\pi$ gilt $a_i < a_j < b_i < b_j$.

4) Für alle i, j mit $a_i < a_j < b_i < b_j$ ist $\{i, j\} \in E_\pi$.

5) Der Algorithmus findet für jedes $\pi \in S_n$ eine korrekte Darstellung von H_π als Überlappungsgraph.

Beweis. Zu 1): Klar.

Zu 2): Für künstliche Knoten siehe 1). Sei also i nicht künstlich und angenommen $a_i > b_i$. Sei k der kleinste künstliche Knoten mit $i < k$, also der Wert der Variable k, wenn a_i gesetzt wird. Dann ist $a_i \in (b_k, b_k + 1)$ und aufgrund der Annahme $b_i < b_k$ (b's ganzzahlig), woraus nach Konstruktion $\pi^{-1}(i) < \pi^{-1}(k)$ folgt.

Da i nicht künstlich ist, gibt es ein $j_1 > i$ mit $\pi^{-1}(j_1) < \pi^{-1}(i)$. Ist j_1 auch nicht künstlich, so iteriere dies bis zu einem künstlichen Knoten j_r mit $j_r > ... > j_1 > i$ und $\pi^{-1}(j_r) < ... < \pi^{-1}(j_1) < \pi^{-1}(i)$. Transitivität liefert $\pi^{-1}(j_r) < \pi^{-1}(k)$ und daher $j_r < k$, da k künstlich ist. Widerspruch zur Wahl von k als den kleinsten künstlichen Knoten mit $i < k$.

Zu 3): Sei $\{i, j\} \in E_\pi$ und o.B.d.A. $i < j$. Nach Konstruktion gibt es ein k mit $i < j < k$ und $\pi^{-1}(k) < \pi^{-1}(i) < \pi^{-1}(j)$. Wir können annehmen, dass k künstlich ist, ansonsten finden wir ein solches k iterativ wie im Beweis von 2). Nach Ausführung des Algorithmus gilt wegen $i < j$ nach 1), dass $a_i < a_j$. Wegen $i, j < k$ werden a_i, b_i spätestens gesetzt, wenn k der aktuelle künstliche Knoten ist, folglich $a_i, a_j < b_k + 1$. Ferner gilt $b_k < b_i < b_j$, da $\pi^{-1}(k) < \pi^{-1}(i) < \pi^{-1}(j)$. Insgesamt gilt $a_i < a_j < b_i < b_j$.

Zu 4): Seien i, j mit $a_i < a_j < b_i < b_j$. Aus $a_i < a_j$ folgt nach 1) $i < j$ und aus $b_i < b_j$ folgt $\pi^{-1}(i) < \pi^{-1}(j)$. Sei k der aktuelle künstliche Knoten, wenn a_j gesetzt wird. Dann ist $j < k$ und $b_k < a_j < b_k + 1$. Aus $a_j < b_i$ folgt dann $b_k < b_i$ und damit $\pi^{-1}(k) < \pi^{-1}(i)$. Insgesamt erhält man $i < j < k$ und $\pi^{-1}(k) < \pi^{-1}(i) < \pi^{-1}(j)$, also ist $\{i, j\} \in E_\pi$.

Zu 5): Nach 2) bilden die ausgegebenen Intervalle in der Tat einen Überlappungsgraphen, der nach 3) und 4) mit H_π identisch ist. □

Interessanterweise zeigen Even und Itai auch, dass es für jeden Überlappungsgraphen G eine Permutation $\pi \in S_n$ mit $G = H_\pi$ gibt, die Graphenklassen also übereinstimmen.

Unger zeigt in [Ung88] die NP-Vollständigkeit des Problems der Bestimmung der chromatischen Zahl von Überlappungsgraphen (Kreisgraphen). Es scheint daher schwierig, einen polynomiellen Algorithmus zur Berechnung der exakten Anzahl benötigter Stacks zum Erzeugen einer Permutation zu finden. Wir verfolgen jedoch weiter unser Ziel einer oberen Schranke.

Lemma 3.12. *Für jeden Überlappungsgraphen $G = (V, E)$ gilt $\chi(G) \leq \omega(G) \cdot \log(|V|)$.*

Beweis. Für den Beweis verweisen wir auf [Čer07]. □

Lemma 3.13. *Sei $\pi \in S_n$. H_π hat genau dann eine r-Clique, falls $[r+1, 1, ..., r] \preceq_{\mathfrak{S}} \pi$.*

Beweis. Bilden die Knoten $i_1, ..., i_r$ in H_π eine r-Clique und gelte o.B.d.A. $i_1 < ... < i_r$. Insbesondere sind dann $\{i_1, i_2\}, \{i_2, i_3\}, ..., \{i_{r-1}, i_r\} \in E_\pi$, woraus nach Definition $\pi^{-1}(i_1) < ... < \pi^{-1}(i_r)$ folgt. Außerdem ist $\{i_1, i_r\} \in E_\pi$, also gibt es nach Definition ein k mit $i_r < k$ und $\pi^{-1}(k) < \pi^{-1}(i_1)$. Insgesamt also $i_1 < ... < i_r < k$ und $\pi^{-1}(k) < \pi^{-1}(i_1) < ... < \pi^{-1}(i_r)$, was einem $[r+1, 1, ..., r]$-Muster entspricht.

Enthalte umgekehrt π das Muster $[r+1, 1, ..., r]$, so gibt es $i_1 < ... < i_{r+1}$ mit $\pi^{-1}(i_{r+1}) < \pi^{-1}(i_1) < ... < \pi^{-1}(i_r)$. Dann bilden nach Definition $i_1, ..., i_r$ in H_π eine r-Clique. □

Wir liefern nun die angekündigte Schranke für Permutationen $\pi \in S_n$ mit kleiner Cliquenzahl $\omega(H_\pi)$.

Theorem 3.14. *Sei $\pi \in S_n, r \geq 1$ und π vermeide das Muster $[r+2, 1, 2, ..., r+1]$. Dann kann π in einem Weichennetzwerk bestehend aus $r \cdot \log(n)$ parallelen Stacks, in dem auch bevor alle Elemente geladen wurden Elemente ausgegeben werden dürfen, erzeugt werden.*

Beweis. Aus Lemma 3.13 folgt $\omega(H_\pi) \leq r$ und nach Lemma 3.12 $\chi(H_\pi) \leq r \cdot \log(n)$, woraus nach Theorem 3.7 die Aussage folgt. □

Für Werte $r \geq \frac{n}{\log(n)}$ in Theorem 3.14 ist die Aussage trivial. Wenn eine Permutation in m parallelen Stacks unter Einhaltung der Mitternachtseinschränkung erzeugt werden kann, so auch ohne Mitternachtseinschränkung. Daher ist nach Theorem 3.4 auch $\chi(G_\pi^{\complement}) = L(\pi)$ eine obere Schranke an die benötigte Zahl an Stacks, welche oftmals, jedoch nicht immer besser ist als die Schranke aus Theorem 3.14.

3.3. Parallele Queues

Wir betrachten das in Abbildung 3.7 dargestellte Weichennetzwerk, bestehend aus m parallel geschalteten Queues.

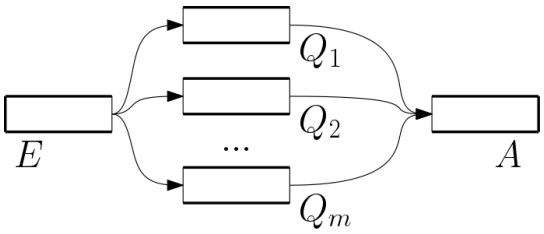


Abbildung 3.7.: Weichennetzwerk bestehend aus m parallelen Queues

Zunächst sieht man, dass die Mitternachtseinschränkung keine wirkliche Einschränkung ist: Lässt sich eine Permutation $\pi \in S_n$ durch eine Folge von Einlese- und Ausgabeoperationen erzeugen, so lässt sie sich auch erzeugen, indem man erst alle Einleseoperationen und danach alle Ausgabeoperationen in jeweils gleicher Reihenfolge ausführt. Die so erzeugte Permutation ändert sich dadurch anders als bei Stacks nicht. Ansonsten erweist sich dieser Fall als analog zu den parallelen Stacks. Zwei Elemente $i < j$ dürfen genau dann nicht über dieselbe Queue laufen, wenn $\pi^{-1}(i) > \pi^{-1}(j)$. Dadurch erhält man G_π statt $G_\pi^{\complement}$ als Konfliktgraph und analog zu Theorem 3.4 und Korollar 3.5 und unter Beachtung voriger Bemerkung beweist man das folgende Theorem.

Theorem 3.15 (Even, Itai)**.** *Sei N ein Weichennetzwerk bestehend aus m parallelen Queues. Dann sind für $\pi \in S_n$ äquivalent:*

1) *Der Graph G_π ist m-färbbar.*
2) *π kann in N erzeugt werden.*
3) *π kann in N sortiert werden.*
4) *π kann in N erzeugt werden, indem erst alle Elemente eingelesen werden, bevor Elemente ausgegeben werden.*
5) *π kann in N sortiert werden, indem erst alle Elemente eingelesen werden, bevor Elemente ausgegeben werden.*

Außerdem berechnet man die minimale Zahl an Queues, um $\pi = [\pi(1), ..., \pi(n)]$ zu erzeugen, indem man Algorithmus 1 auf die Permutation $\tilde{\pi} := [\pi(n), ..., \pi(1)]$ anwendet, da $G_{\tilde{\pi}}^{\complement} = G_\pi$ gilt.

3.4. Stacks in Reihe

Wir betrachten das in Abbildung 3.8 dargestellte Weichennetzwerk, bestehend aus m in Reihe geschalteten Stacks.

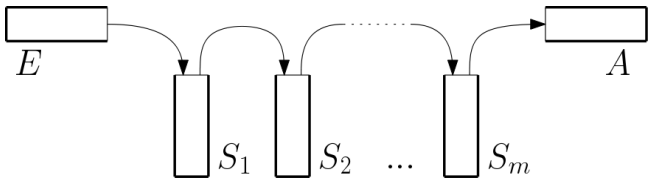


Abbildung 3.8.: Weichennetzwerk bestehend aus m Stacks in Reihe

Dieser Fall ist der schwierigste unter allen hier behandelten. Ohne die Mitternachtseinschränkung stellt Knuth in [Knu75] fest, dass sich alle Permutationen $\pi \in S_n$ mittels $\lceil \log_2(n) \rceil$ in Reihe geschalteten Stacks sortieren lassen: Für $n = 1, 2$ ist dies klar (Kein Stack bzw. 1 Stack). Für $n > 2$ folgt dies mit Induktion: Man lädt die Elemente $\pi(1), ..., \pi(n)$ von E in den ersten Stack. Elemente $\lceil \frac{n}{2} \rceil + 1, ..., n$ belässt man dabei zunächst auf dem ersten Stack, während man Elemente $1, ..., \lceil \frac{n}{2} \rceil$ sofort nach dem Laden in die $\lceil \log_2(n) \rceil - 1$ weiteren Stacks schiebt, wo sie nach Induktionsvoraussetzung sortiert werden können und in der Reihenfolge $1, ..., \lceil \frac{n}{2} \rceil$ ausgegeben werden können. Anschließend werden die auf dem ersten Stack geparkten Elemente über die $\lceil \log_2(n) \rceil - 1$ nun frei gewordenen Stacks nach Induktionsvoraussetzung sortiert und in der Reihenfolge $\lceil \frac{n}{2} \rceil + 1, ..., n$ ausgegeben.

Tarjan verbessert diesen Ansatz in [Tar72] geringfügig. Durch Computerberechnung oder manuelle Fallunterscheidungen findet man heraus, dass sich alle $\pi \in S_n$ mit $n \leq 6$ mittels 2 Stacks in Reihe sortieren lassen. Daraus erhält man durch einen analogen Induktionsschritt folgendes Theorem:

Theorem 3.16 (Tarjan)**.** *In einem Weichennetzwerk bestehend aus $m \geq 2$ Stacks in Reihe, in dem auch Elemente ausgegeben werden dürfen, bevor alle Elemente eingelesen wurden, können alle Permutationen $\pi \in S_n$ mit $n \leq 3 \cdot 2^{m-1}$ sortiert werden.*

Die Mitternachtseinschränkung erhöht die erforderliche Zahl an Stacks um höchstens 2. Denn angenommen eine Permutation $\pi \in S_n$ lässt sich in m Stacks in Reihe ohne Mitternachtseinschränkung sortieren. Nun kann man an das Ende der Reihe von Stacks zwei weitere Stacks S_{m+1}, S_{m+2} anfügen und dieselbe Operationsfolge ausführen, wobei statt dem Ausgeben Elemente nun in S_{m+1} landen. Anschließend schiebt man die Elemente in umgekehrter Reihenfolge von S_{m+1} nach S_{m+2} und gibt sie in korrekter Reihenfolge und unter Einhaltung der Mitternachtseinschränkung aus.

4. Spezielle Instanzen von zyklischen Weichennetzwerken

4.1. Mehr als zwei parallel verbundene Stacks

Wir wenden uns nun Weichennetzwerken zu, die neben den Queues E und A aus $k \geq 3$ parallelen Stacks bestehen, von denen jeder mit jedem über eine bidirektionale Kante (d.h. je eine Kante pro Richtung) verbunden ist. Ferner sind E und A direkt mit allen Stacks verbunden. Abbildung 4.1 zeigt als Beispiel den Fall $k = 4$.

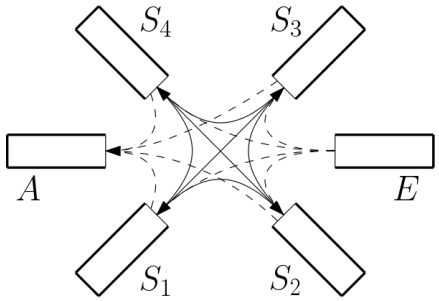


Abbildung 4.1.: Weichennetzwerk bestehend aus 4 verbundenen Stacks. Der Übersichtlichkeit halber sind die Verbindungen mit E und A nur gestrichelt eingezeichnet.

Wir geben in diesem Abschnitt die Ausführungen von Felsner und Pergel in [FP08] auszugsweise wieder.

Lemma 4.1. *Sei $k \geq 3$. In einem Weichennetzwerk bestehend aus k verbundenen Stacks lässt sich eine Permutation $\pi \in S_n$ mit $\mathcal{O}(n \log n)$ Operationen sortieren.*

Beweis. Folgender Algorithmus sortiert π: Von der Eingangsqueue kommend werden die n Elemente von π möglichst gleichmäßig so auf die k Stacks $S_1, ..., S_k$ verteilt, dass auf Stack S_1 die $\frac{n}{k}$ kleinsten Elemente liegen, auf Stack S_2 die $\frac{n}{k}$ nächstkleineren Elemente liegen und so weiter und schließlich auf Stack S_k die $\frac{n}{k}$ größten Elemente liegen (Runde entsprechend). Im Weiteren sprechen wir von *Blöcken*: Jeder Stack enthält nun einen Block von etwa $\frac{n}{k}$ Elementen. In jeder Iteration wird der Block gewählt, welcher die kleinsten noch nicht ausgegebenen Elemente enthält. Besteht der Block aus nur einem Element, so wird dieses sofort ausgegeben. Andernfalls wird der

Block aufgelöst und seine Elemente wieder der Größe nach gleichmäßig auf die $k-1$ anderen Stacks verteilt, wo sie neue Blöcke über den bereits in den Stacks befindlichen Blöcken bilden. Iteriere, bis alle Elemente ausgegeben wurden.

Für die asymptotische Analyse der benötigten Operationen genügt es anzunehmen, dass die n Elemente zu Beginn in exakt k Blöcke zu je $\frac{n}{k}$ Elementen zerlegt werden und in jeder Iteration ein Block mit $n' \geq 1$ Elementen stets in exakt $k-1$ Blöcke zu je $\frac{n'}{k-1}$ Elementen zerlegt wird. Dies ist nämlich zunächst nur für n der Form $n = k(k-1)^r$ für ein $r \in \mathbb{N}$ korrekt, durch ein Monotonieargument lässt sich die folgende asymptotische Abschätzung jedoch auf alle $n \in \mathbb{N}$ übertragen. Ein Element wird im Weichennetzwerk eingefügt und so oft auf andere Stacks verschoben, bis der Block, in dem es sich befindet, nur noch aus diesem Element besteht, und daraufhin ausgegeben. Insgesamt wird ein Element also $2 + \log_{k-1}(\frac{n}{k}) \in \mathcal{O}(\log n)$ mal gezogen und die Gesamtzahl an Operationen beträgt $\mathcal{O}(n \log n)$. □

Lemma 4.2. *Sei $k \geq 2$. In einem Weichennetzwerk bestehend aus k verbundenen Stacks für das man fordert, zuerst alle Elemente einzulesen, bevor Elemente ausgegeben werden, liegt die Komplexität zum Sortieren einer Permutation $\pi \in S_n$ in $\Omega(n \log n)$.*

Beweis. Sei $n \in \mathbb{N}$ und $t \in \mathbb{N}$ so, dass sich jede Permutation $\pi \in S_n$ in N mit höchstens t Operationen sortieren lässt. Eine Operation in N kann als Paar (i, j) mit $1 \leq i, j \leq k$ kodiert werden, wobei (i, j) mit $i \neq j$ ein Element vom i-ten Stack in den j-ten Stack verschiebt und (i, i) ein Element von E in den i-ten Stack lädt bzw. vom i-ten Stack ein Element ausgibt, falls bereits alle Elemente eingelesen sind. Es gibt somit höchstens k^{2t} mögliche Operationsfolgen der Länge t, von denen jede genau eine Permutation sortiert. Unter Nutzung der Stirlingschen Formel $n! \approx (\frac{n}{e})^n \sqrt{2\pi n}$ muss für n hinreichend groß $k^{2t} \geq n! > (\frac{n}{e})^n$ gelten, woraus $2t \geq n \log_k(n) - n \log_k(e) \in \Omega(n \log n)$ folgt. □

Theorem 4.3 (Felsner, Pergel)**.** *Sei $k \geq 3$. In einem Weichennetzwerk N bestehend aus k verbundenen Stacks ist die Komplexität, eine Permutation $\pi \in S_n$ zu sortieren, $\Theta(n \log n)$.*

Beweis. Nach Lemma 4.2 in Verbindung mit Lemma 2.7 ist die Komplexität nach unten beschränkt durch $\Omega(n \log n)$. Die obere Schranke liefert Lemma 4.1. □

4.2. Zwei parallel verbundene Stacks

Wir betrachten dasselbe Weichennetzwerk wie im vorigen Abschnitt nun für den Fall $k = 2$. Wir bezeichnen dabei die beiden Stacks als linken Stack S_L und rechten Stack S_R. Die 6 möglichen Operationen des Weichennetzwerks sind $\texttt{in}_\texttt{L}$, $\texttt{in}_\texttt{R}$ (Einlesen und Einfügen in den linken bzw. rechten Stack), $\texttt{shift}_\texttt{L}$, $\texttt{shift}_\texttt{R}$ (Abheben eines Elements

von S_R und Einfügen in S_L bzw. anders herum), $\texttt{out}_\texttt{L}$ und $\texttt{out}_\texttt{R}$ (Ausgeben des obersten Elements von S_L bzw. S_R).

Eine Operationsfolge zum Sortieren mit Mitternachtseinschränkung gliedert sich in zwei Phasen, dem Einlesen und dem Ausgeben der Elemente. Die Konfiguration, in der sich S_L und S_R nach dem Einlesen befinden, lässt sich als Permutation $\tau \in S_n$ beschreiben. Hierfür stellt man sich S_L und S_R als linken und rechten Stack horizontal Kopf an Kopf verbunden vor und liest dann die n Elemente von links nach rechts. Diese Permutation τ bezeichnen wir im Folgenden als die *Mitternachtspermutation* und den Sortiervorgang als *Sortieren über* τ.

Definition 4.4. Für $\pi \in S_n$ definieren wir:

1) Für $i = 1, ..., n-1$:

$$I_\pi^i := \begin{cases} \{j : \pi^{-1}(i) < \pi^{-1}(j) < \pi^{-1}(i+1)\} & : \pi^{-1}(i) < \pi^{-1}(i+1) \\ \{j : \pi^{-1}(i+1) < \pi^{-1}(j) < \pi^{-1}(i)\} & : \pi^{-1}(i+1) < \pi^{-1}(i) \end{cases}$$

2) $$s(\pi) := \sum_{i=1}^{n-1} |I_\pi^i| = \sum_{i=1}^{n-1} -1 + |\pi^{-1}(i) - \pi^{-1}(i+1)|$$

3) $$s_l(\pi) := \sum_{i=1}^{n-1} |\{j \in I_\pi^i : j < i\}| \qquad s_u(\pi) := \sum_{i=1}^{n-1} |\{j \in I_\pi^i : j > i\}|$$

Anschaulich zählt $s(\pi)$, wie viele Elemente man überspringt, wenn man in der Sequenz $[\pi(1), ..., \pi(n)]$ von 1 nach 2, von 2 nach 3 und so weiter und letztlich von $n-1$ nach n springt. Dagegen zählen $s_l(\pi)$ bzw. $s_u(\pi)$ nur die Sprünge über kleinere bzw. größere Elemente. Also gilt $s(\pi) = s_l(\pi) + s_u(\pi)$.

Beispiel 4.5. Sei $\sigma = [5, 3, 2, 4, 1]$. Dann ist $s_l(\sigma) = 3, s_u(\sigma) = 1$ und damit $s(\sigma) = s_l(\sigma) + s_u(\sigma) = 4$. Abbildung 4.2 zeigt die Sprünge.

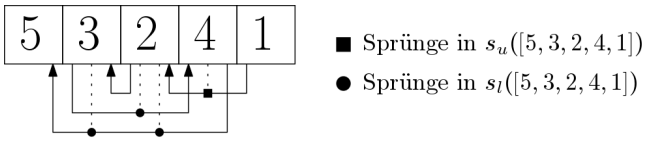


Abbildung 4.2.: Beiträge von $s_l(\sigma)$ und $s_u(\sigma)$ mit $\sigma = [5, 3, 2, 4, 1]$

.

Triviale obere Schranken sind gegeben durch

$$s_l(\pi), s_u(\pi) \leq \frac{(n-1)(n-2)}{2} \qquad \text{für alle } \pi \in S_n.$$

Für Permutationen der Form $[n, ..., 4, 2, 1, 3, 5, ..., n-1, (n+1)]$ wird die Schranke für s_l angenommen und von ganz ähnlichen Permutationen wird die Schranke für s_u angenommen. Beide Schranken sind also scharf. Interessanter ist die Frage nach einer scharfen Schranke für s. Folgendes Lemma liefert die Antwort.

Lemma 4.6. *Für alle* $n \in \mathbb{N}, n \geq 2$ *gilt* $\max\{s(\pi) : \pi \in S_n\} = \lfloor \frac{n^2}{2} \rfloor - n$.

Die Anregung und Idee zu folgendem Beweis stammt von S. Felsner im Rahmen der Betreuung dieser Arbeit. Siehe auch die verwandte OEIS-Folge A047838 [Som].

Beweis. Sei $\pi \in S_n$ beliebig. Für $j = 1, ..., n$ bezeichne $\tilde{I}_\pi^j := \{i : j \in I_\pi^i\}$ die Menge der Sprünge, zu denen Element j beiträgt. In der Permutation π stehen für beliebiges j links des Elements $\pi(j)$ genau $j-1$ Elemente, daher sind höchstens $2(j-1)$ Sprünge über $\pi(j)$ möglich, also $|\tilde{I}_\pi^j| \leq 2(j-1)$. Über die Zahl der Elemente rechts von $\pi(j)$ erhält man analog $|\tilde{I}_\pi^j| \leq 2(n-j)$. Insgesamt folgt

$$s(\pi) = \sum_{i=1}^{n-1} |I_\pi^i| = \sum_{j=1}^{n} |\tilde{I}_\pi^j| \leq \sum_{j=1}^{n} \min\{2(j-1), 2(n-j)\}$$

Für ungerade n beachte man zusätzlich für das mittlere Element $k := \pi(\lfloor \frac{n}{2} \rfloor + 1)$ die Schranke $|\tilde{I}_\pi^k| \leq 2 \cdot \lfloor \frac{n}{2} \rfloor - 1$. Dann folgt mit obigem Ansatz durch einfaches Ausrechnen

$$s(\pi) \leq 2 \cdot \left\lfloor \frac{n}{2} \right\rfloor \left(\left\lfloor \frac{n}{2} \right\rfloor - 1 \right) + (n \bmod 2) \cdot \left(2 \cdot \left\lfloor \frac{n}{2} \right\rfloor - 1 \right) = \left\lfloor \frac{n^2}{2} \right\rfloor - n$$

Diese Schranke ist scharf. Für gerades n konstruiere man $\pi \in S_n$ so, dass in einer Hälfte der Permutation $[\pi(1), ..., \pi(n)]$ die ungeraden und in der anderen Hälfte die geraden Elemente stehen und dabei die Elemente 1 und n die mittleren beiden Stellen einnehmen. Beispielsweise für $n = 6$ die Permutation $\pi = [4, 2, 6, 1, 5, 3]$. Setze hilfsweise $I_\pi^0 := \emptyset, I_\pi^n := \emptyset$. Dann gilt:

$$\begin{aligned} s(\pi) = \sum_{i=1}^{n-1} |I_\pi^i| &= \sum_{\substack{i=1 \\ i \equiv 1 \pmod 2}}^{n} |I_\pi^{i-1} \cap (2\mathbb{Z}+1)| + |I_\pi^i \cap (2\mathbb{Z}+1)| \\ &\quad + \sum_{\substack{i=1 \\ i \equiv 0 \pmod 2}}^{n} |I_\pi^{i-1} \cap (2\mathbb{Z})| + |I_\pi^i \cap (2\mathbb{Z})| \\ &= \sum_{i=0}^{\frac{n}{2}-1} 2i + \sum_{i=0}^{\frac{n}{2}-1} 2i = 4 \cdot \frac{\frac{n}{2}(\frac{n}{2}-1)}{2} = \left\lfloor \frac{n^2}{2} \right\rfloor - n \end{aligned}$$

Für ungerades n konstruiere man zunächst eine Permutation für $n-1$ wie oben beschrieben und füge das Element n mittig ein. Man rechnet ähnlich nach, dass auch diese Permutation obige Schranke annimmt. □

Der Nutzen der eingeführten Notationen $s(\pi), s_l(\pi), s_u(\pi)$ wird durch folgendes Lemma deutlich.

Lemma 4.7. *Seien $\pi, \tau \in S_n$ mit $\pi(n) = 1$. Die Zahl der benötigten Operationen, um π über τ zu sortieren, beträgt genau $s_l(\pi^{-1} \circ \tau) + s_u(\tau) + 2n$.*

Beweis. Die Elemente, die sich in den beiden Stacks S_L und S_R befinden, kann man sich wie auf einem Band vorstellen, das mit $\texttt{shift}_\texttt{L}$ bzw. $\texttt{shift}_\texttt{R}$ nach links bzw. rechts verschoben wird, damit an den richtigen Stellen Elemente eingefügt (beim Einlesen) oder ausgegeben (beim Ausgeben) werden können.

Sei nun die Mitternachtspermutation τ vorgegeben. Für das Ausgeben werden $s_u(\tau)+n$ Operationen benötigt: Als letztes Element wird $\pi(n) = 1$ eingelesen, somit liegt beim Beginn des Ausgebens 1 auf einem der Stacks zuoberst und kann sofort ausgegeben werden. Für das weitere Ausgeben wird so oft $\texttt{shift}_\texttt{L}$ oder $\texttt{shift}_\texttt{R}$ ausgeführt, bis 2 ausgegeben werden kann usw. Die benötigten `shift`-Operationen zwischen der Ausgabe von i und $i + 1$ verschieben noch nicht ausgegebene Elemente, entsprechen also Elementen in $[\tau(1), ..., \tau(n)]$, die zwischen i und $i + 1$ liegen und größer als $i + 1$ sind. Daher werden $s_u(\tau)$ `shift`-Oprationen und n `out`-Operationen benötigt.

Wenn man unnötige `shift`-Operationen vermeidet, sind durch τ auch die Operationen des Einlesevorgangs eindeutig bestimmt bis auf die Information, auf welchen der beiden Stacks $\pi(n) = 1$ gelegt wird. Letzteres kann man beliebig wählen, es wirkt sich nicht auf die Zahl der Operationen des nachfolgenden Ausgebens aus. Abbildung 4.3 zeigt ein Beispiel. Die Permutation $\pi^{-1} \circ \tau$ gibt die Reihenfolge der Elemente von π an, wie sie in $[\tau(1), ..., \tau(n)]$ stehen. Die benötigten `shift`-Operationen zwischen dem Einfügen von $\pi(i)$ und $\pi(i + 1)$ verschieben bereits eingelesene Elemente, entsprechen also Elementen in $[\pi^{-1} \circ \tau(1), ..., \pi^{-1} \circ \tau(n)]$, die zwischen i und $i + 1$ liegen und kleiner als i sind. Daher werden $s_l(\pi^{-1} \circ \tau)$ `shift`-Operationen und n `in`-Operationen benötigt. □

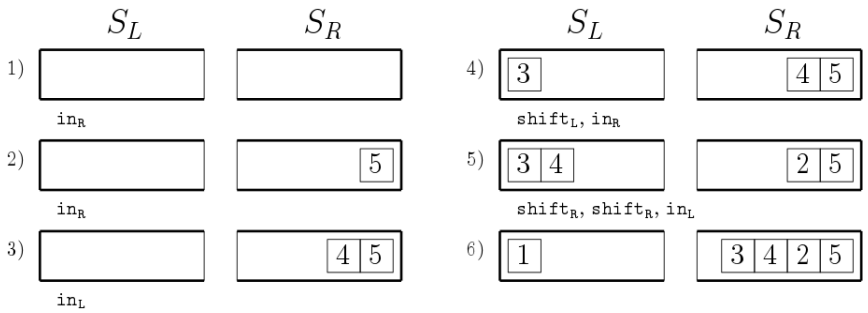


Abbildung 4.3.: Einlesen beim Sortieren von $\pi = [5, 4, 3, 2, 1]$ über $\tau = [1, 3, 4, 2, 5]$. Beachte $\pi^{-1} \circ \tau = [5, 3, 2, 4, 1]$, $s_l(\pi^{-1} \circ \tau) = 3$, vgl. Beispiel 4.5

.

Das Finden einer minimalen Operationsfolge zum Sortieren von $\pi \in S_n$ mit Mitternachtseinschränkung ist nach Lemma 4.7 äquivalent zum Bestimmen einer Mitternachtspermutation $\tau \in S_n$, so dass $s_l(\pi^{-1} \circ \tau) + s_u(\tau) + 2n$ minimal. Offensichtlich sind $s_l(\pi^{-1} \circ \tau), s_u(\tau) \in \mathcal{O}(n^2)$ und damit ist die Komplexität beschränkt durch $\mathcal{O}(n^2)$. Wir gehen im Folgenden dem in dieser Situation typischen Ansatz nach, den Erwartungswert der benötigten Zahl an Operationen zu bestimmen, wenn π über eine zufällige Mitternachtspermutation τ sortiert wird.

Lemma 4.8. *Es werde $\pi \in S_n$ uniform zufällig gewählt, d.h. $\mathbb{P}(\pi) = \frac{1}{n!}$ für jede Wahl von π. Dann ist*

$$\mathbb{E}(s(\pi)) = \frac{(n-1)(n-2)}{3}$$

Die Idee des folgenden Beweises stammt von King (Siehe [Kin]). Die Aussage lässt sich auch aus [Cla05] folgern.

Beweis. Sei $\pi \in S_n$ zufällig. Für $i = 1, ..., n-1$ bezeichne

$$\chi_\pi^i(j) := \begin{cases} 1 & : j \in I_\pi^i \\ 0 & : j \notin I_\pi^i \end{cases} \qquad \text{für } j = 1, ..., n$$

die Indikatorfunktion von I_π^i. Für alle i, j mit $j \in \{i, i+1\}$ ist $\chi_\pi^i(j) = 0$ und für alle $j \notin \{i, i+1\}$ ist $\mathbb{E}[\chi_\pi^i(j)] = \frac{1}{3}$, da es 6 mögliche Anordnungen der Elemente $i, j, i+1$ gibt, welche aus Symmetriegründen in π gleich wahrscheinlich auftreten, und von denen genau $\pi^{-1}(i) < \pi^{-1}(j) < \pi^{-1}(i+1)$ und $\pi^{-1}(i+1) < \pi^{-1}(j) < \pi^{-1}(i)$ zu $\chi_\pi^i(j) = 1$ führen.

Daraus folgt:

$$\begin{aligned}
\mathbb{E}[s(\pi)] &= \mathbb{E}\left[\sum_{i=1}^{n-1} |I_\pi^i|\right] = \mathbb{E}\left[\sum_{i=1}^{n-1}\sum_{j=1}^{n} \chi_\pi^i(j)\right] \\
&= \mathbb{E}\left[\sum_{i=1}^{n-1} \sum_{\substack{j=1 \\ j \notin \{i,i+1\}}}^{n} \chi_\pi^i(j)\right] \\
&= \sum_{i=1}^{n-1} \sum_{\substack{j=1 \\ j \notin \{i,i+1\}}}^{n} \mathbb{E}[\chi_\pi^i(j)] = \sum_{i=1}^{n-1} \sum_{\substack{j=1 \\ j \notin \{i,i+1\}}}^{n} \frac{1}{3} \\
&= \frac{(n-1)(n-2)}{3}
\end{aligned}$$

□

Theorem 4.9. *Sei $\pi \in S_n$ mit $\pi(n) = 1$. Dann werden in einem Weichennetzwerk bestehend aus zwei verbundenen parallelen Stacks zum Sortieren von π über eine uniform zufällig gewählte Mitternachtspermutation $\tau \in S_n$ im Erwartungsfall $\frac{1}{3}(n-1)(n-2)+2n$ Operationen benötigt.*

Beweis. Wählt man eine uniform zufällige Mitternachtspermutation $\tau \in S_n$, so ist nach Lemma 4.7 die erwartete Anzahl an Operationen zum Sortieren von π über τ gegeben durch

$$\begin{aligned}
\mathbb{E}[s_l(\pi^{-1} \circ \tau) + s_u(\tau) + 2n] &= \mathbb{E}[s_l(\pi^{-1} \circ \tau)] + \mathbb{E}[s_u(\tau)] + 2n \\
&= \mathbb{E}[s_l(\tau)] + \mathbb{E}[s_u(\tau)] + 2n \\
&= \mathbb{E}[s_l(\tau) + s_u(\tau)] + 2n \\
&= \mathbb{E}[s(\tau)] + 2n \\
&= \frac{(n-1)(n-2)}{3} + 2n
\end{aligned}$$

□

Interessant ist, dass der Erwartungswert in Theorem 4.9 unabhängig von π ist. Es muss daher stets eine Mitternachtspermutation $\tau \in S_n$ geben, über die π in höchstens $\frac{(n-1)(n-2)}{3} + 2n$ Operationen sortiert wird. Der Leitkoeffizient $\frac{1}{3}$ lässt sich jedoch sehr einfach auf $\frac{1}{4}$ verbessern: Sei n geradzahlig und $\pi \in S_{n+1}$ mit $\pi(n+1) = 1$. Beim Einlesen können die $\frac{n}{2} + 1$ kleinsten Elemente auf S_L und die $\frac{n}{2}$ größten Elemente auf S_R gelegt werden. Das Element 1 kann sofort ausgegeben werden. Als nächstes müssen die verbleibenden $\frac{n}{2}$ Elemente auf S_L ausgegeben werden. Dafür sind höchstens $\sum_{i=1}^{\frac{n}{2}-1} i = \frac{n^2}{8} - \frac{n}{4}$ `shift`-Operationen nötig, ebenso zum anschließenden Ausgeben der $\frac{n}{2}$ Elemente auf S_R. Zuzüglich der $n+1$ `in`- und $n+1$ `out`-Operationen sind dies insgesamt $\frac{1}{4}(n^2 + 6n + 8)$ Operationen. Ähnlich ist die Analyse bei Permutationen π gerader Länge.

Zieht man τ uniform zufällig, so scheint aufgrund von Computerexperimenten die Verteilung von $s_l(\pi^{-1} \circ \tau) + s_u(\tau)$ für große n gegen eine Normalverteilung zu streben. In einem Versuch wurde die Permutation $\pi = [6, 5, 8, 4, 10, 2, 7, 9, 3, 1]$ zufällig gewählt. Die Verteilung von $s_l(\pi^{-1} \circ \tau) + s_u(\tau)$ wird durch die Punkte im Diagramm in Abbildung 4.4 dargestellt. Sie wurde durch Iteration über alle 10! Mitternachtspermutationen τ berechnet. Die Kurve im Hintergrund ist eine angepasste Gauss-Kurve mit Erwartungswert 24 (Vgl. Theorem 4.9) und Varianz $\approx 12,7$ (angepasst).

Eine interessante offene Frage von Felsner und Pergel lautet, ob zum Sortieren einer Permutation $o(n^2)$ Operationen immer genügen. Wir geben im Folgenden den Ansatz von Felsner und Pergel in [FP08] wieder, die Komplexität von unten zu beschränken.

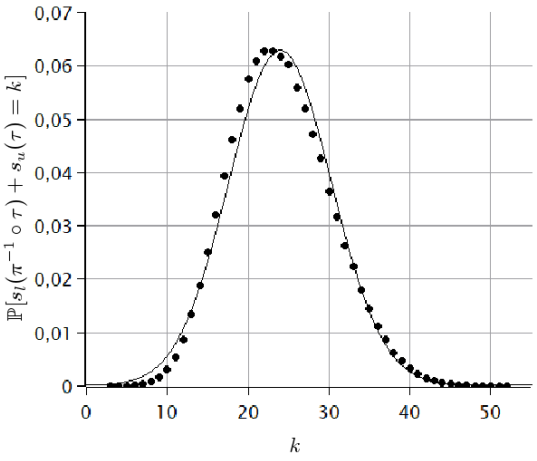


Abbildung 4.4.: Verteilung von $s_l(\pi^{-1} \circ \tau) + s_u(\tau)$
bei zufälliger Mitternachtspermutation τ

.

Definition 4.10. Für ein $\alpha \in (0, 1)$ nennen wir die Permutationen $\tau, \tau' \in S_n$ *α-nah*, wenn für jedes $p \in \mathbb{N}_0$ gilt:

$$\begin{aligned} &\{\tau(i) : \lfloor pn^\alpha \rfloor + 1 \leq i \leq \lfloor (p+1)n^\alpha \rfloor, i \leq n\} \\ &= \{\tau'(i) : \lfloor pn^\alpha \rfloor + 1 \leq i \leq \lfloor (p+1)n^\alpha \rfloor, i \leq n\} \end{aligned}$$

Anschaulich wird die Permutation τ in etwa $n^{1-\alpha}$ Blöcke der Größe höchstens n^α zerlegt und τ' heißt dann α-nah zu τ, wenn τ' identisch mit τ ist bis auf die Reihenfolgen der Elemente innerhalb der Blöcke. Abbildung 4.5 zeigt ein Beispiel.

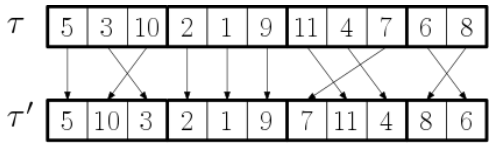


Abbildung 4.5.: Beispiel für zwei α-nahe Permutationen τ, τ'
mit $n = 11, \alpha = \frac{1}{2}, n^\alpha = n^{1-\alpha} \approx 3.3$

Lemma 4.11. *Sei $\pi \in S_n$ und $\tau, \tau' \in S_n$ seien α-nah. Falls das Sortieren von π über τ höchstens $c \cdot n^{1+\alpha}$ Operationen benötigt, so benötigt das Sortieren von π über τ' höchstens $(c+4)n^{1+\alpha}$ Operationen.*

Beweis. Die beim Sortieren von π über τ verwendeten `shift`-Operationen entsprechen Sprüngen über größere Elemente in τ und über kleinere Elemente in $\pi^{-1} \circ \tau$ (Vgl. Lemma 4.7) und zusätzlich einem Sprung in τ von $\pi(n)$ nach 1, da wir nicht notwendigerweise $\pi(n) = 1$ fordern. Jeder dieser $2n - 1$ Sprünge verlängert sich in τ' gegenüber τ um

höchstens $2n^\alpha$. Insgesamt sind höchstens $(4n-2)n^\alpha \leq 4n^{\alpha+1}$ mehr Operationen beim Sortieren von π über τ' als über τ nötig. □

Lemma 4.11 nützt uns wie folgt: Die Komplexität zum Sortieren von $\pi \in S_n$ ist nach oben beschränkt durch $\mathcal{O}(n^2)$ und sicherlich nach unten beschränkt durch $\Omega(n)$. Wir interessieren uns daher für eine untere Schranke $\Omega(n^{1+\alpha})$ mit möglichst großem $\alpha > 0$. Für $\alpha \in (0,1)$ und $\tau \in S_n$ gibt es $(n^\alpha)!^{\frac{n}{n^\alpha}} = (n^\alpha)!^{n^{1-\alpha}} \geq (\frac{n^\alpha}{e})^n$ Permutationen $\tau' \in S_n$, die α-nah zu τ sind[1]. Dann erhalten wir mit Lemma 4.11, dass wenn es eine Mitternachtspermutation τ gibt, über die π in höchstens $c \cdot n^{1+\alpha}$ Operationen sortiert wird, so gibt es $(\frac{n^\alpha}{e})^n$ verschiedene Mitternachtspermutationen τ', über die π in höchstens $(c+4)n^{1+\alpha}$ Operationen sortiert wird. Ist daher das kleinste t, so dass jede Permutation $\pi \in S_n$ über $(\frac{n^\alpha}{e})^n$ verschiedene Mitternachtspermutationen mit höchstens t Operationen sortiert werden kann, nach unten beschränkt durch $\Omega(n^{1+\alpha})$ für ein $\alpha \in (0,1)$, so auch die gesuchte Komplexität des Sortierens. Diese Beobachtung wird in folgendem Theorem von Felsner und Pergel [FP08] benutzt:

Theorem 4.12 (Felsner, Pergel)**.** *In einem Weichennetzwerk bestehend aus 2 verbundenen parallelen Stacks ist die Komplexität zum Sortieren einer Permutation $\pi \in S_n$ nach unten beschränkt durch $\Omega(n^{2-\epsilon})$ für alle $\epsilon > 0$.*

Beweis. Es genügt, Permutationen $\pi \in S_n$ mit $\pi(n) = 1$ zu betrachten, die die Mitternachtseinschränkung erzwingen, siehe Lemma 2.7. Folgendes Abzählargument untersucht das kleinste $t \in \mathbb{N}$, so dass alle Permutationen $\pi \in S_n$ mit $\pi(n) = 1$ in genau t Operationen sortiert werden können. Damit dadurch auch Permutationen erfasst werden, die eigentlich in weniger als t Operationen sortiert werden können, führen wir Pseudooperationen ein, die etwa vor allen anderen Operationen ausgeführt werden können und am Zustand des Weichennetzwerks nichts ändern, sondern nur die Operationsfolge künstlich verlängern.

Operationsfolgen zum Sortieren (Ohne unnötiges Hin- und Herschieben von Elementen) sind dann vollständig durch die Zahl der Pseudooperationen p und die Stack-zu-Stack-Verschiebungen beschrieben, die zwischen dem Einlesen bzw. Ausgeben zweier aufeinanderfolgender Elemente geschehen (Vgl. Beweis Lemma 4.7). Wir können letztere als Folge von Tupeln $(a_i, b_i), i = 1, ..., 2n-2$ kodieren, wobei $a_i \in \{l, r\}$ die Verschieberichtung und b_i die Zahl der verschobenen Elemente angeben.

Die Gesamtzahl der Operationen t setzt sich zusammen aus den Verschiebungen, den n Einlese- und n Ausgabeoperationen, und der Zahl der Pseudooperationen p, also $t = 2n + p + \sum_{i=1}^{2n-2} b_i$ bzw. $p + \sum_{i=1}^{2n-2} b_i = t - 2n$. Für festes t bilden die Zahlen $(b_i)_{i=1,...,2n-2}$ und p daher eine Komposition von $t-2n$ in $2n-1$ Summanden. Die Zahl möglicher Kompositionen ist gegeben durch $\binom{t-2}{2n-2} \leq \binom{t}{2n}$. Für die Wahl der

[1] Für die folgende asymptotische Untersuchung kann vernachlässigt werden, dass manche Ausdrücke nicht für alle Werte n und α ganzzahlig sind.

a_i gibt es jeweils zwei Möglichkeiten, somit gibt es höchstens $2^{2n-2}\binom{t}{2n} \leq 2^{2n}\binom{t}{2n}$ Möglichkeiten zur Wahl einer Operationsfolge zum Sortieren. Es muss $2^{2n}\binom{t}{2n} \geq n!$ gelten[2], da Operationsfolgen zum Sortieren verschiedener Permutationen verschieden sind. Wir können diese Schranke noch mit Hilfe von Lemma 4.11 und der anschließenden Bemerkung verbessern, indem wir für ein beliebiges $\alpha \in (0,1)$ fordern, dass jede Permutation π über $(\frac{n^\alpha}{e})^n$ verschiedene Mitternachtspermutationen und damit über $(\frac{n^\alpha}{e})^n$ verschiedene Operationsfolgen sortiert werden soll, und erhalten

$$2^{2n}\binom{t}{2n} \geq n!\left(\frac{n^\alpha}{e}\right)^n$$

Anwenden der Stirlingschen Formel liefert für große n:

$$2^{2n}t^{2n} \geq \left(\frac{n}{e}\right)^n\left(\frac{2n}{e}\right)^{2n}\left(\frac{n^\alpha}{e}\right)^n$$

Division durch 2^{2n} ergibt $t^{2n} \geq \left(\frac{n^{3+\alpha}}{e^4}\right)^n$, woraus durch Ziehen der $2n$-ten Wurzel folgt:

$$t \geq \frac{1}{e^2}n^{\frac{3+\alpha}{2}}$$

Also liegt t in $\Omega(n^{\frac{3}{2}+\frac{\alpha}{2}})$. Da $\alpha \in (0,1)$ beliebig gewählt werden kann, folgt die Aussage. □

4.2.1. Die Suche nach einer optimalen Mitternachtspermutation

Wir stellen uns die praktische Frage, wie mit Hilfe von Computerberechnungen eine Mitternachtspermutation τ gefunden werden kann, über die π in minimal vielen Operationen sortiert werden kann. Der naive Ansatz, alle Mitternachtspermutationen zu testen, scheitert für größere n an der Laufzeit.

Ein naheliegender Ansatz ist das Starten mit einer beliebigen Mitternachtspermutation und schrittweise Durchführen von lokalen Änderungen, die die Zahl benötigter Operationen senken. Versuchsweise wurde dies wie folgt implementiert: Starte mit Mitternachtspermutation $\tau = [1, ..., n]$. Rotiere benachbarte Elemente in τ so, dass $s_l(\pi^{-1} \circ \tau') + s_u(\tau')$ unter den durch einzelne Rotationen entstehenden Mitternachtspermutationen τ' minimiert wird. Iteriere dies, bis keine Verbesserung mehr möglich ist. Präzise werden in jeder Iteration sämtliche folgenden Rotationen getestet: Für alle $i < j$ die Rotationen

$$\begin{aligned} &\tau' := [\tau(1), ..., \tau(i-1), \tau(j), \tau(i), ..., \tau(j-1), \tau(j+1), ..., \tau(n)] \\ \text{und} \quad &\tau' := [\tau(1), ..., \tau(i-1), \tau(i+1), ..., \tau(j), \tau(i), \tau(j+1), ..., \tau(n)]. \end{aligned}$$

[2]Eigentlich $(n-1)!$, für die Asymptotik genügt jedoch, mit $n!$ zu rechnen.

Computerberechnungen zeigen: Für alle $\pi \in S_n$ mit $n \leq 5$ und $\pi(n) = 1$ wird dadurch eine optimale Mitternachtspermutation gefunden. Für $n \geq 6$ endet man jedoch möglicherweise für $\pi = [3, 5, 2, 4, 6, 1]$ bei $\tau = [5, 3, 2, 1, 4, 6]$ mit $s_l(\pi^{-1} \circ \tau) + s_u(\tau) = 2$, obwohl mit $\tau' = [3, 2, 1, 6, 4, 5]$ auch $s_l(\pi^{-1} \circ \tau') + s_u(\tau') = 1$ möglich wäre. Es scheint, als könnten mit dieser Methode für kleine n schnell gute Mitternachtspermutationen gefunden werden, jedoch für große n die Lücke zwischen Optimum und gefundener Lösung größer werden.

Dieselbe Idee wurde auch für lokale Änderungen an τ ausprobiert, die aus dem ein- oder zweimaligen Vertauschen von jeweils zwei Elementen in τ bestehen. Das Ergebnis ist ähnlich. Für $\pi \in S_n, n \leq 5, \pi(n) = 1$ erhält man stets eine optimale Mitternachtspermutation, jedoch für $\pi = [2, 5, 3, 6, 4, 1]$ möglicherweise $\tau = [2, 1, 3, 4, 6, 5]$ mit $s_l(\pi^{-1} \circ \tau) + s_u(\tau) = 2$, obwohl mit $\tau' = [5, 3, 2, 1, 4, 6]$ auch $s_l(\pi^{-1} \circ \tau') + s_u(\tau') = 1$ möglich wäre.

4.2.2. Einschränkung der `shift`-Operationen

Wir betrachten weiterhin ein Weichennetzwerk bestehend aus zwei parallel verbundenen Stacks und fordern zusätzlich zur Mitternachtseinschränkung, dass `shift`-Operationen zwischen den beiden Stacks erst ausgeführt werden, wenn alle Elemente eingelesen wurden. Ziel ist es also, $\pi(1), ..., \pi(n)$ so auf S_L und S_R zu verteilen, dass die benötigten `shift`-Operationen während des Ausgebens minimiert werden. Deren Zahl untersuchen wir im Folgenden genauer. Sie werden wie in Lemma 4.7 für $\pi(n) = 1$ durch $s_u(\tau)$ gezählt, wobei τ die durch die Verteilung der Elemente auf S_L und S_R bestimmte Mitternachtspermutation ist.

Seien $\pi(i), \pi(j)$ beliebige aufeinanderfolgende Elemente, d.h. $|\pi(i) - \pi(j)| = 1$, mit $i < j$. Abhängig von der Verteilung der Elemente auf S_L und S_R geraten ggf. gewisse Elemente $\pi(k) > \pi(i), \pi(j)$ in $[\tau(1), ..., \tau(n)]$ zwischen die Elemente $\pi(i)$ und $\pi(j)$. Dies sind die Beiträge zu $s_u(\tau)$. Wann ein Element $\pi(k) > \pi(i), \pi(j)$ in $[\tau(1), ..., \tau(n)]$ zwischen die Elemente $\pi(i)$ und $\pi(j)$ gerät, untersuchen wir durch folgende Fallunterscheidung:

- Fall $k < i$: $\pi(k)$ wird vor $\pi(i)$ und $\pi(j)$ eingelesen und kann in τ nicht zwischen $\pi(i)$ und $\pi(j)$ geraten. Kein Beitrag zu $s_u(\tau)$.

- Fall $i < k < j$: $\pi(k)$ gerät in τ zwischen $\pi(i)$ und $\pi(j)$ genau dann, wenn $\pi(i)$ und $\pi(k)$ auf denselben Stack gelegt werden. In diesem Fall entsteht ein Beitrag zu $s_u(\tau)$.

- Fall $j < k$: $\pi(k)$ gerät in τ zwischen $\pi(i)$ und $\pi(j)$ genau dann, wenn $\pi(i)$ und $\pi(j)$ auf verschiedene Stacks gelegt werden. In diesem Fall entsteht ein Beitrag zu $s_u(\tau)$.

Die Frage nach einer optimalen Operationsfolge unter der behandelten Einschränkung lässt sich nun als Schnitt-Problem eines ungerichteten, gewichteten Graphen formulieren. Betrachte dazu die Knotenmenge $V_\pi = \{1, ..., n\}$ und den Graphen $G_\pi := (V_\pi, E_\pi \cup E'_\pi)$[3] mit den Kantenmengen

$$E_\pi := \big\{\{\pi(i), \pi(k)\} : \exists j : 1 \leq i < k < j \leq n : |\pi(i) - \pi(j)| = 1 \wedge \pi(k) > \pi(i), \pi(j)\big\}$$
$$E'_\pi := \big\{\{\pi(i), \pi(j)\} : \exists k : 1 \leq i < j < k \leq n : |\pi(i) - \pi(j)| = 1 \wedge \pi(k) > \pi(i), \pi(j)\big\}$$

mit den Gewichten

$$w\big(\{\pi(i), \pi(k)\}\big) := |\{j : 1 \leq i < k < j \leq n : |\pi(i) - \pi(j)| = 1 \wedge \pi(k) > \pi(i), \pi(j)\}|$$

für $\{\pi(i), \pi(k)\} \in E_\pi$ und

$$w\big(\{\pi(i), \pi(j)\}\big) := -|\{k : 1 \leq i < j < k \leq n : |\pi(i) - \pi(j)| = 1 \wedge \pi(k) > \pi(i), \pi(j)\}|$$

für $\{\pi(i), \pi(j)\} \in E'_\pi$.

Man beachte $w(\{\pi(i), \pi(k)\}) \in \{1, 2\}$ für alle $\{\pi(i), \pi(k)\} \in E_\pi$ und

$$E'_\pi \subset \{(i, i+1) : i = 1, ..., n-1\}.$$

Abbildung 4.6 zeigt ein Beispiel.

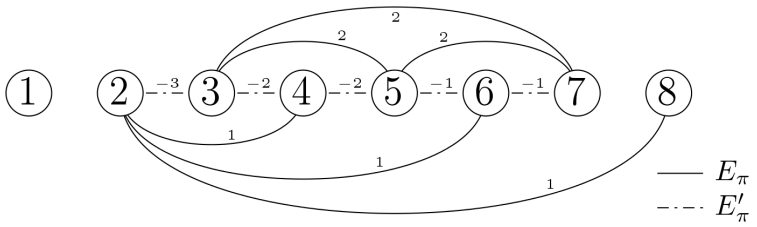


Abbildung 4.6.: Beispiel für den Graphen G_π für $\pi = [3, 5, 7, 2, 4, 6, 8, 1]$

Für einen gewichteten Graphen $G = (V, E)$ mit Gewichten $w \in \mathbb{R}^E$ sei für eine Teilmenge $V' \subset V$ der Knotenmenge

$$\gamma_G(V') := \sum_{\substack{e=\{v_1,v_2\}\in E \\ v_1,v_2\in V'}} w(e)$$

die Gewichtssumme der Kanten innerhalb von V'. Dementsprechend bezeichnen $\gamma_{G_\pi}(L)$ bzw. $\gamma_{G_\pi}(R)$ für eine Partition $V_\pi = L \dot\cup R$ die Gewichtssummen der Kanten innerhalb L bzw. R und $\gamma_{(V_\pi, E'_\pi)}(V_\pi) \leq 0$ die Gewichtssumme aller Kanten des Graphen G_π in E'_π. Wir benötigen diese Notation im folgenden Lemma.

[3]Der Graph G_π ist nicht identisch mit dem gleichnamigen Graphen aus Kapitel 3.

Lemma 4.13. *Sei N ein Weichennetzwerk bestehend aus 2 verbundenen parallelen Stacks für das man fordert,* `shift`*-Operationen erst auszuführen, nachdem alle Elemente eingelesen wurden. Seien L und $R = V_\pi \setminus L$ die Mengen der Elemente, die beim Einlesen auf S_L bzw. S_R gelegt werden. Dann werden zum Sortieren einer Permutation $\pi \in S_n$ mit $\pi(n) = 1$ genau $\gamma_{G_\pi}(L) + \gamma_{G_\pi}(R) - \gamma_{(V_\pi, E'_\pi)}(V_\pi)$* `shift`*-Operationen benötigt.*

Beweis. Der Beweis ergibt sich aus obiger Fallunterscheidung: Ein Beitrag des zweiten Falls entspricht einem Gewichtsbeitrag einer Kante $e \in E_\pi$, die innerhalb einer der Partitionsklassen L oder R liegt. Für Beiträge des dritten Falls müssten analog den Kanten $e \in E'_\pi$ die positiven Gewichte $-w(e) > 0$ zugewiesen werden und die Gewichte jener Kanten aus E'_π summiert werden, die zwischen den Partitionsklassen L und R verlaufen. Äquivalent dazu summiert man alle diese Gewichte $(= -\gamma_{(V_\pi, E'_\pi)}(V_\pi))$ und subtrahiert die Gewichte von Kanten in E'_π, die innerhalb von L oder R liegen. Daher das negative Gewicht dieser Kanten. □

Man kann sich das Problem als Suche nach einer „Färbung“ (eher *Labeling*, *Beschriftung*) der Knoten von G_π mittels 2 Farben vorstellen, in der man die Gewichtssumme *monochromatischer* Kanten minimiert oder äquivalent dazu die Gewichtssumme *bichromatischer* Kanten maximiert. Ein solches Problem wird *MaxCut*-Problem genannt (dt. Maximaler-Schnitt-Problem). Es ist prinzipiell durch Negation der Kantengewichte überführbar in das Finden eines minimalen Schnitts. Man beachte jedoch, dass aufgrund der dann mitunter entstehenden negativen Kantengewichte die bekannten und effizienten MinCut- / MaxFlow-Algorithmen nicht anwendbar sind. Tatsächlich ist das allgemeine MaxCut-Problem NP-vollständig (Karps 21. Problem, siehe [Kar72]). Aufgrund der gemischten Vorzeichen der Gewichte sind selbst die bekannten MaxCut-Approximationsalgorithmen (0.5-Approximation durch Greedy-Wahl der Partitionsklassen oder 0.87-Approximation mit der Goemans-Williamson-Methode, siehe [GW95]) zumindest nicht direkt anwendbar.

Im Hinblick auf die Frage, wie „schwierig“ das MaxCut-Problem für unsere spezielle Graphenklasse $\{G_\pi : \pi \in S_n, n \in \mathbb{N}\}$ ist, führen wir darauf zurück das MaxCut-Problem für *modifizierte Permutationsgraphen* (Siehe folgende Definition), welche eine flexible und gut verständliche Graphenklasse darstellen.

Definition 4.14. Sei $G' = (V', E'), V' = \{1, ..., n\}$ ein Permutationsgraph, d.h. es gibt eine Permutation $\pi \in S_n$ so, dass $E' = \big\{\{i, j\} : i < j,\ \pi^{-1}(j) < \pi^{-1}(i)\big\}$, und seien $w'(e) = 1$ für alle $e \in E'$ dessen Kantengewichte.

Dann heißt der Graph $G = (V, E)$ *modifizierter Permutationsgraph*, falls G aus G' auf folgende Art hervorgeht:

$$\begin{aligned} V &:= V' = \{1, ..., n\} \\ E &:= E' \cup \big\{\{i, i+1\} : 1 \le i \le n-1\big\} \\ w(e) &:= w'(e) = 1 \qquad \text{für alle } e \in E' \\ w\big(\{i, i+1\}\big) &:= \frac{i-n}{2} \qquad \text{für alle } \{i, i+1\} \in E \setminus E' \end{aligned}$$

Lemma 4.15. *Ein polynomieller Algorithmus für das MaxCut-Problem der Graphenklasse $\{G_\pi : \pi \in S_n, n \in \mathbb{N}\}$ liefert einen polynomiellen Algorithmus für das MaxCut-Problem von modifizierten Permutationsgraphen.*

Beweis. Sei G beliebiger modifizierter Permutationsgraph und $\pi \in S_n$ die dem zugrundeliegenden Permutationsgraphen zugehörige Permutation. Betrachte die Permutation

$$\tilde{\pi} := [2 \cdot \pi(n), 2 \cdot \pi(n-1), ..., 2 \cdot \pi(1), 1, 3, 5, ..., 2n+1]$$

und den zugehörigen Graphen $G_{\tilde{\pi}} = (\{1, ..., 2n+1\}, E_{\tilde{\pi}} \cup E'_{\tilde{\pi}})$. Definiere die folgende „Umbenennung“ der Knoten:

$$\begin{aligned} l : \{1, 2, ..., n, 1', 2', ..., (n+1)'\} &\to \{1, ..., 2n+1\} \\ k &\mapsto 2k \\ k' &\mapsto 2k' - 1 \end{aligned}$$

Dann sind die Knoten $l(1), ..., l(n)$ bzw. $l(1'), ..., l((n+1)')$ in $G_{\tilde{\pi}}$ jene, die in der Definition von $G_{\tilde{\pi}}$ von den ersten n Elementen bzw. von den letzten $n+1$ Elementen in $\tilde{\pi}$ stammen.

Alle Kanten aus $E_{\tilde{\pi}}$ verlaufen innerhalb der Knoten $l(1), ..., l(n)$, nämlich existiert für $1 \le i < j \le n$ die Kante $\{l(i), l(j)\} \in E_{\tilde{\pi}}$ genau dann, wenn $\tilde{\pi}^{-1}(2i) < \tilde{\pi}^{-1}(2j)$ ist, was wegen der Anordnung der ersten n Elemente in $\tilde{\pi}$ genau dann der Fall ist, wenn $\pi^{-1}(j) < \pi^{-1}(i)$ gilt. In diesem Fall ist $w\big(\{l(i), l(j)\}\big) = 2$. Außerdem gilt:

$$\begin{aligned} E'_{\tilde{\pi}} = &\big\{\{l((i+1)'), l(i+1)\} : i = 0, ..., n-1\big\} \\ \cup &\big\{\{l(i), l((i+1)')\} : i = 1, ..., n-1\big\} \end{aligned}$$

$$\begin{aligned} w\big(\{l((i+1)', l(i+1)\}\big) &= i - n \qquad \text{für } i = 0, ..., n-1 \\ w\big(\{l(i), l((i+1)')\}\big) &= i - n \qquad \text{für } i = 1, ..., n-1 \end{aligned}$$

Nach Voraussetzung lässt sich MaxCut für $G_{\tilde{\pi}}$ polynomiell berechnen. Man sieht leicht, dass darauf MaxCut für den wie folgt veränderten Graphen polynomiell zurückgeführt werden kann: Lösche die Knoten $l(1')$ vom Grad 1 und $l((n+1)')$ vom Grad 0. Für $i = 2, ..., n$ sind die Knoten $l(i')$ jeweils mit ihren beiden einzigen Nachbarn $l(i)$ und $l(i+1)$ über Kanten des Gewichts $i - n$ verbunden. Lösche diese Knoten $l(2'), ..., l(n')$ und verbinde ihre Nachbarn direkt über eine Kante des Gewichts $i - n$. Dividiere alle Kantengewichte durch 2. Der so entstehende Graph entspricht dem Graphen G. □

Unser Ziel ist nun, die Zahl der benötigten `shift`-Operationen für den eingeschränkten Fall nach unten zu beschränken. Dafür betrachten wir für beliebiges $r \in \mathbb{N}$ die Permutation $\pi_r := [r, r-1, ..., 1]$. Der zugehörige Permutationsgraph besteht aus einer einzigen r-Clique. Diese findet sich wieder in den Kanten $E_{\tilde{\pi}}$ des Graphen $G_{\tilde{\pi}}$, wenn man $\tilde{\pi} \in S_{2r+1}$ wie im Beweis von Lemma 4.15 konstruiert. Um Lemma 4.13 anwenden zu können (Voraussetzung $\pi(n) = 1$), betrachten wir die präparierte Permutation $\hat{\pi}_r := [\tilde{\pi}(1)+1, ..., \tilde{\pi}(n)+1, 1]$ der Größe $n = 2r+2$. Der Graph $G_{\hat{\pi}_r}$ unterscheidet sich von $G_{\tilde{\pi}}$ nur geringfügig. Jedenfalls enthalten die Kanten $E_{\hat{\pi}_r}$ weiterhin eine r-Clique mit Kanten des Gewichts 2, nun bestehend aus den Knoten $V_C := \{3, 5, 7, ..., 2r+1\}$. Für $r = 3$ beispielsweise ist $\hat{\pi}_r = [3, 5, 7, 2, 4, 6, 8, 1]$, siehe Abbildung 4.6.

Eine Partition $V_{\hat{\pi}_r} = L \dot{\cup} R$ minimiert die Zahl der Kanten aus $E_{\hat{\pi}_r}$ innerhalb $L \cap V_C$ und $R \cap V_C$, wenn $||L \cap V_C| - |R \cap V_C|| \leq 1$, d.h. L und R enthalten möglichst $\frac{r}{2}$ der Knoten von V_C. Daher ist gemäß Lemma 4.13 die Zahl der `shift`-Operationen beim Ausgeben nach unten wie folgt beschränkt:

$$\begin{aligned}\gamma_{G_{\hat{\pi}_r}}(L) + \gamma_{G_{\hat{\pi}_r}}(R) - \gamma_{(V_{\hat{\pi}_r}, E'_{\hat{\pi}_r})}(V_{\hat{\pi}_r}) &\geq \gamma_{(V_{\hat{\pi}_r}, E_{\hat{\pi}_r})}(L) + \gamma_{(V_{\hat{\pi}_r}, E_{\hat{\pi}_r})}(R) \\ &\geq 2 \cdot 2 \cdot \frac{\frac{r}{2} \cdot \left(\frac{r}{2} - 1\right)}{2} \\ &= \frac{r(r-2)}{2} \sim \frac{n^2}{8} \in \Omega(n^2)\end{aligned}$$

Mit geringem Mehraufwand holt man aus diesem Ansatz noch etwas mehr heraus, wie im Folgenden gezeigt wird.

Lemma 4.16. *Sei $r \geq 2$. Löscht man in $[\hat{\pi}_r(1), ..., \hat{\pi}_r(2r+2)]$ ein Element $\pi(i) \neq 1$, so enthält die verbleibende Sequenz eine Teilsequenz der Länge $2r$, die die Permutation $\hat{\pi}_{r-1}$ realisiert.*

Beweis. Wenn Element $k \notin \{1, 2r+2\}$ gelöscht wird, so lösche zusätzlich $k+1$, und wenn Element $2r+2$ gelöscht wird, so lösche zusätzlich $2r+1$. Dies liefert eine Teilsequenz, die $\hat{\pi}_{r-1}$ realisiert. □

Theorem 4.17. *Sei N ein Weichennetzwerk bestehend aus 2 verbundenen parallelen Stacks für das man zusätzlich fordert, höchstens $c \cdot n$ `shift`-Operationen auszuführen,* ***bevor*** *alle Elemente eingelesen sind, wobei $c < \frac{1}{2}$. Dann ist die Komplexität zum Sortieren einer Permutation $\pi \in S_n$ mit $\pi(n) = 1$ gegeben durch $\Theta(n^2)$.*

Selbiges gilt, wenn man stattdessen zusätzlich fordert, höchstens $c \cdot n$ `shift`-Operationen auszuführen, ***nachdem*** *alle Elemente eingelesen sind, wobei $c < \frac{1}{2}$.*

Beweis. Betrachte wieder die Eingabepermutation $\hat{\pi}_r$ der Länge $n = 2r + 2$. Angenommen, beim Einlesen werden höchstens $c \cdot n$ `shift`-Operationen ausgeführt. $\pi_r(n) = 1$ wird nie verschoben, von den anderen Elementen werden höchstens $c \cdot n$ mindestens einmal verschoben. Durch wiederholtes Anwenden von Lemma 4.16 sieht man, dass $[\hat{\pi}_r(1), ..., \hat{\pi}_r(n)]$ eine Teilsequenz enthält, die $\pi_{r'}$ mit $r' \geq r - c \cdot n$ realisiert und deren Elemente beim Einlesen nicht verschoben werden. Mit

$$r' \geq r - c \cdot n = r - c \cdot (2r + 2) = (1 - 2 \cdot c)r - 2 \cdot c$$

folgt aus der Beobachtung eben, dass asymptotisch

$$\frac{r'(r' - 2)}{2} \geq \frac{(r' - 2)^2}{2} \geq \frac{((1 - 2c)r - 2c - 2)^2}{2} \sim \frac{(1 - 2c)^2}{8} n^2 \in \Omega(n^2)$$

`shift`-Operationen beim Ausgeben benötigt werden. Dies zeigt die erste Aussage.

Der Zusatz folgt daraus: Angenommen, Permutationen $\pi \in S_n$ mit $\pi(n) = 1$ können mit $o(n^2)$ Operationen sortiert werden, wenn man fordert, beim Ausgeben höchstens $c \cdot n$ `shift`-Operationen auszuführen. Dann gilt dies auch für beliebige Permutationen $\pi \in S_{n-1}$. Sei ω_π die Operationsfolge, die $\pi \in S_{n-1}$ entsprechend sortiert. Kehrt man die Reihenfolge von ω_π um und ersetzt man $\texttt{out}_\texttt{L}$ durch $\texttt{in}_\texttt{L}$, $\texttt{out}_\texttt{R}$ durch $\texttt{in}_\texttt{R}$, $\texttt{shift}_\texttt{L}$ durch $\texttt{shift}_\texttt{R}$, $\texttt{shift}_\texttt{R}$ durch $\texttt{shift}_\texttt{L}$, $\texttt{in}_\texttt{L}$ durch $\texttt{out}_\texttt{L}$ und $\texttt{in}_\texttt{R}$ durch $\texttt{out}_\texttt{R}$, so erhält man eine Operationsfolge ω'_π, die $[n - 1, ..., 1]$ in $[\pi(n - 1), ..., \pi(1)]$ überführt und dabei beim Einlesen höchstens $c \cdot n$ `shift`-Operationen verwendet. ω'_π sortiert die Permutation $[n - 1, ..., 1] \circ \pi^{-1} \circ [n - 1, ..., 1]$. Da jede Permutation so dargestellt werden kann, kann jede Permutation $\pi \in S_{n-1}$ und damit jede Permutation $\pi \in S_n$ mit $\pi(n) = 1$ mit höchstens $c \cdot n$ `shift`-Operationen beim Einlesen in $o(n^2)$ Operationen sortiert werden, im Widerspruch zum ersten Teil. □

Man könnte vermuten, dass die Permutation $\hat{\pi}_r$ auch ohne Einschränkung der `shift`-Operationen zur Sortierung in zwei parallel verbundenen Stacks viele, vielleicht sogar quadratisch viele Operationen benötigt. Tatsächlich genügen $6r + 1 = 12n + 13 \in \mathcal{O}(n)$ Operationen: Nach Ausführung von $r-1$ mal $\texttt{in}_\text{L}$, einmal $\texttt{in}_\text{R}$, $r-2$ mal $\texttt{shift}_\text{R}$, einmal $\texttt{in}_\text{L}$ und $r+1$ mal $\texttt{in}_\text{R}$ erhält man $\hat{\tau}_r = [3, 2, 1, 2r + 2, 2r, 2r - 2, ..., 4, 5, 7, 9, ..., 2r + 1]$ als Mitternachtspermutation mit $s_u(\hat{\tau}_r) = r - 1$.

5. Fazit

In Kapitel 3 wurden einfache Instanzen azyklischer Weichennetzwerke betrachtet. Im Wesentlichen den Erkenntnissen von Even und Itai aus [EI71] folgend lässt sich die Frage nach der Sortierbarkeit einer Permutation in m parallelen Stacks mit oder ohne Mitternachtseinschränkung oder in m parallelen Queues jeweils auf ein Graphenfärbungsproblem zurückführen. Dieses lässt sich effizient berechnen, außer für den Fall der parallelen Stacks ohne Mitternachtseinschränkung. Für diesen Fall wurde unter Verwendung jüngerer Erkenntnis aus [Čer07] zumindest eine obere Schranke an die Zahl der benötigten Stacks zum Sortieren einer Permutation gefunden, die das Muster $[r+1, 1, ..., r]$ nur für kleine r enthält.

Als Beispiel für zyklische Weichennetzwerke wurden in Kapitel 4 k parallele und miteinander verbundene Stacks betrachtet. Entsprechend den Ausführungen von Felsner und Pergel in [FP08] wurde gezeigt, dass zum Sortieren von $\pi \in S_n$ mit $k \geq 3$ Stacks $\Theta(n \log n)$ Operationen nötig sind.

Der Fall $k = 2$ erweist sich als erheblich schwieriger. Hier zeigen Felsner und Pergel nur eine untere Schranke von $\Omega(n^{2-\epsilon})$ für beliebiges $\epsilon > 0$. Die spannende Frage, ob $o(n^2)$ Operationen zum Sortieren stets genügen, bleibt ebenso offen wie die Frage, wie eine optimale Operationsfolge gefunden werden kann. Operationsfolgen, die erst alle Elemente einlesen, bevor Elemente ausgegeben werden, werden von Felsner und Pergel durch deren sogenannte Mitternachtspermutation charakterisiert. Als Beitrag dieser Arbeit wurde die Berechnung der Anzahl benötigter Operationen für das Sortieren über eine Mitternachtspermutation formalisiert und gezeigt, dass deren Erwartungswert bei Wahl einer zufälligen Mitternachtspermutation unabhängig von der zu sortierenden Permutation ist. Man kann vermuten, dass deren Verteilung gegen eine Gauss-Verteilung konvergiert.

Um dem Problem für zwei verbundene Stacks näher zu kommen, wurde der Ansatz verfolgt, während des Einlesens (oder Ausgebens) der Elemente nur „wenige“ `shift`-Operationen zuzulassen. In Theorem 4.17 wurde gezeigt, dass dann zum Sortieren in der Tat $\Theta(n^2)$ Operationen nötig sind. Die Frage, ob Permutationen in zwei uneingeschränkten parallel verbundenen Stacks in $o(n^2)$ Operationen sortiert werden können, ist zwar weiterhin offen. Falls dies jedoch möglich sein sollte, so zeigt das Resultat, dass für manche Permutationen $\Omega(n)$ `shift`-Operationen während des Einlesens (oder Ausgebens) einer optimalen Operationsfolge benötigt würden. Es wäre dann für op-

timale Operationsfolgen nicht möglich, in einer der Phasen immer nur „sehr wenige“ `shift`-Operationen auszuführen. Diese Überlegung könnte sich als hilfreich für weitere Ansätze erweisen, die Frage zu beantworten.

Da Weichennetzwerke bestehend aus zwei parallel verbundenen Stacks, in denen erst `shift`-Operationen zwischen den Stacks ausgeführt werden dürfen, nachdem alle Elemente eingelesen wurden, auch für sich von Interesse sind, stellt Theorem 4.17 eine kleine Erweiterung des Wissensstandes dar. Neben der Komplexität von $\Theta(n^2)$ wurde ferner gezeigt, dass sich das Problem einer optimalen Operationsfolge auf das MaxCut-Problem einer bestimmten Graphenklasse zurückführen lässt.

Literaturverzeichnis

[Čer07] Jakub Černý, *Coloring circle graphs*, ENDM, vol. 29, 2007, pp. 457–461.

[Cla05] Lane Clark, *Asymptotic distribution of the sum of the lengths of ascents or of descents in permutations*, KAM-DIAMATIA Series (2005), no. 757.

[EI71] Shimon Even and Alon Itai, *Queues, stacks and graphs*, Theory of Machines and Computations, 1971, pp. 71–86.

[FP08] Stefan Felsner and Martin Pergel, *The complexity of sorting with networks of stacks and queues*, LNCS, vol. 5193, 2008, pp. 417–429.

[GW95] Michel Goemans and David Williamson, *Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming*, JACM **42** (1995), pp. 1115–1145.

[HS77] James W. Hunt and Thomas G. Szymanski, *A fast algorithm for computing longest common subsequences*, CACM **20** (1977), no. 5, pp. 350–353.

[Kar72] Richard M. Karp, *Reducibility among combinatorial problems*, Complexity of Computer Computations, 1972, pp. 85–103.

[Kin] Andrew D. King, *Distribution of distances in permutations*, MathOverflow, URL:https://mathoverflow.net/q/122328 (version: 2013-02-22).

[KL08] Felix G. König and Marco E. Lübbecke, *Sorting with complete networks of stacks*, LNCS, vol. 5369, 2008, pp. 895–906.

[KLM+07] Felix G. König, Macro Lübbecke, Rolf Möhring, Guido Schäfer, and Ines Spenke, *Solutions to real-world instances of PSPACE-complete stacking*, LNCS, vol. 4698, 2007, pp. 729–740.

[Knu75] Donald E. Knuth, *The art of computer programming*, 3. ed., vol. 1, 1975.

[Rom15] Dan Romik, *The surprising mathematics of longest increasing subsequences*, 2015.

[Som] Michael Somos, *Sequence A047838*, The On-Line Encyclopedia of Integer Sequences, URL:https://oeis.org/A047838 (version: 2019-07-11).

[Sta15] Richard P. Stanley, *Catalan numbers*, 2015.

[Tar72] Robert Tarjan, *Sorting using networks of queues and stacks*, JACM **19** (1972), pp. 341–346.

[Ung88] Walter Unger, *On the k-colouring of circle-graphs*, LNCS, vol. 294, 1988, pp. 61–72.