6 Algebraische Methoden

6.1 Der Satz von Schwartz-Zippel

In diesem Abschnitt betrachten wir Polynome mit mehreren Variablen $X_1, \ldots, X_n$ über einem Körper K. Ein solches Polynom f kann man als endliche Summe von Termen der Form

$$a_{\ell_1,\ldots,\ell_n} \cdot X_1^{\ell_1} X_2^{\ell_2} \cdots X_n^{\ell_n}$$

schreiben, wobei $a_{\ell_1,\ldots,\ell_n}$ ein Element von K ist, und in verschiedenen Termen die Exponentenfolge $\ell_1, \ldots, \ell_n$ verschieden ist. Der *Grad* eines solchen Terms ist $\ell_1 + \cdots + \ell_n$, der Grad $\deg(f)$ des Polynoms f ist das Maximum der Grade seiner Terme. Das Nullpolynom hat keinen Term, sein Grad ist $-\infty$.

Beispiele für Polynome über dem Körper $\mathbb{Z}_5$:

$$X_1^2 X_2^2 + 4X_1 X_4^3 + 2X_2^3 X_3^3,\ X_1 X_2 + 2X_2 + X_3,\ X_1^3 + 3X_1^2 X_2,\ 3,$$

mit den Graden 6, 2, 3 und 0. Polynome kann man addieren, subtrahieren und multiplizieren, und vereinfachen, indem man Terme, die zu derselben Exponentenfolge $\ell_1, \ldots, \ell_n$ gehören, zusammenfasst.

Ein Vektor $(a_1, \ldots, a_n) \in K^n$ heißt eine Nullstelle von f, wenn sich beim Einsetzen von a_i für X_i in f und Ausrechnen der Wert $f(a_1, \ldots, a_n) = 0$ ergibt. Beispielsweise ist $(1, 3)$ eine Nullstelle von $X_1^3 + 3X_1^2 X_2$ (über $K = \mathbb{Z}_5$).

Folgendes ist leicht zu beweisen (und zumindest für den Körper der reellen Zahlen aus der Schule bekannt):

Lemma 6.1
Wenn f ein Polynom mit einer Variablen $X = X_1$ ist, und $\deg(f) = d \geq 0$, so besitzt f höchstens d Nullstellen.

Wir können dies auf Polynome mit n Variablen verallgemeinern:

Lemma 6.2 ***Schwartz-Zippel***
Wenn f ein Polynom mit n Variablen und Grad $d \geq 0$ ist und $S \subseteq K$ eine endliche Menge ist, dann besitzt f in S^n höchstens $d|S|^{n-1}$ Nullstellen.

Diese Aussage kann nicht verbessert werden. Betrachte den Körper $\mathbb{Z}_p$ für eine Primzahl $p > s$. Das Polynom $f = (X_1 - 1)(X_1 - 2)(X_1 - 3) \cdots (X_1 - d)$ (mit n Variablen,

von denen $X_2, \ldots, X_n$ in f nicht vorkommen) hat in $\{1, \ldots, s\}^n$ die ds^{n-1} Nullstellen $(i, a_2, \ldots, a_n)$, mit $1 \leq i \leq d$ und $1 \leq a_2, \ldots, a_n \leq s$ beliebig.

Beweis von Lemma 6.2: Induktion über n. Der Induktionsanfang $n = 1$ ist durch Lemma 6.1 gegeben.

Sei also jetzt $n > 1$, und sei als Induktionsvoraussetzung angenommen, dass die Aussage für Polynome mit maximal $n-1$ Variablen stimmt.

Wenn $\deg(f) = 0$ ist, dann hat f überhaupt keine Nullstelle, und es ist nichts zu zeigen. Also können wir o. B. d. A. annehmen, dass f eine Variable enthält. Wir nehmen an, dass X_1 in f vorkommt. Wir wählen δ so, dass X_1^δ die höchste Potenz ist, mit der X_1 in f vorkommt, und klammern X_1^δ aus allen Termen aus, in denen dieser Faktor vorkommt. Dies liefert die Zerlegung

$$f(X_1, \ldots, X_n) = X_1^\delta \cdot q(X_2, \ldots, X_n) + r(X_1, \ldots, X_n),$$

wobei in $q \neq 0$ die Variable X_1 nicht vorkommt, der Grad von $q(X_2, \ldots, X_n)$ maximal $d - \delta$ ist und in $r(X_1, \ldots, X_n)$ der Faktor X_1 nur mit Exponenten $< \delta$ auftaucht. Der Grad von $r(X_1, \ldots, X_n)$ ist maximal d.

Wir definieren

$$A = \{(a_1, \ldots, a_n) \in S^n \mid q(a_2, \ldots, a_n) = 0\}$$

und

$$B = \{(a_1, \ldots, a_n) \in S^n \mid q(a_2, \ldots, a_n) \neq 0 \wedge f(a_1, \ldots, a_n) = 0\}.$$

Man sieht sofort, dass

$$f(a_1, \ldots, a_n) = 0 \;\Rightarrow\; (a_1, \ldots, a_n) \in A \cup B.$$

Wir müssen also zeigen, dass $|A \cup B| \leq |A| + |B| \leq d|S|^{n-1}$ ist.

Dazu beobachten wir: Nach Induktionsvoraussetzung ist

$$|A| \leq |S| \cdot (d - \delta)|S|^{n-2} = (d - \delta)|S|^{n-1}.$$

Die Zahl $|B|$ schätzen wir so ab: Für jeden Vektor $(a_2, \ldots, a_n)$ mit $q(a_2, \ldots, a_n) \neq 0$ ist $f(X_1, a_2, \ldots, a_n)$ ein Polynom in der einen Variablen X_1, vom Grad genau $\delta \geq 1$. Dieses Polynom hat (nach Lemma 6.1) maximal δ Nullstellen. Wir summieren über alle $(a_2, \ldots, a_n)$ und erhalten, dass $|B| \leq |S|^{n-1} \cdot \delta$ gilt. Wenn wir die Abschätzungen für A und B addieren, folgt $|A| + |B| \leq d|S|^{n-1}$, also hat f maximal $d|S|^{n-1}$ Nullstellen – das ist die Induktionsbehauptung. $\square$

6.2 Vergleich von Polynomprodukten

Im folgenden sei K irgendein (endlicher oder unendlicher) Körper.

Problemstellung: Gegeben seien Polynome $f_1, \ldots, f_r$ und $g_1, \ldots, g_s$ über K mit Variablen $X_1, \ldots, X_n$. Sei $f = f_1 \cdot \ldots \cdot f_r$ und $g = g_1 \cdot \ldots \cdot g_s$. Gilt $f = g$?

Um die Schwierigkeit der Fragestellung zu illustrieren, betrachten wir ein Beispiel. Die Polynome $f_1, \ldots, f_{2n}$ seien

$$X_1 + 1, \ldots, X_n + 1, X_1 - 1, \ldots, X_n - 1,$$

die Polynome $g_1, \ldots, g_n$ seien

$$X_1^2 - 1, \ldots, X_n^2 - 1.$$

Wir „sehen“, dass (aufgrund der Formel $(X_i + 1)(X_i - 1) = X_i^2 - 1$) die Produkte der beiden Folgen gleich sind. Algorithmisch ist dies aber schwierig zu behandeln. (Die Terme könnten anders angeordnet sein, und zudem mit weniger offensichtlichen anderen Faktoren multipliziert sein.) Der nächstliegende deterministische Algorithmus wird die beiden Seiten ausmultiplizieren und dann vergleichen. Im Beispiel entstehen dann aber exponentiell viele Terme, der Aufwand ist also immens.

Tatsächlich ist für das Problem „Vergleich von Polynomprodukten“ kein deterministischer Algorithmus bekannt, der polynomielle Laufzeit hat. Wir geben einen sehr einfachen randomisierten Algorithmus an.

Es sei

$$d = \max\{\deg(f_1) + \cdots + \deg(f_r), \deg(g_1) + \cdots + \deg(g_s)\}.$$

(Dann ist offenbar $\deg(f - g) \leq d$.) Wir wählen eine beliebige endliche Menge[1] $S \subseteq K$ mit $|S| \geq 2d$. Nun wählen wir $\boldsymbol{a} = (a_1, \ldots, a_n) \in S^n$ zufällig und berechnen

$$b = f_1(\boldsymbol{a}) \cdot \ldots \cdot f_r(\boldsymbol{a}) \quad \text{und} \quad c = g_1(\boldsymbol{a}) \cdot \ldots \cdot g_s(\boldsymbol{a}).$$

(Es wird in die einzelnen Faktoren eingesetzt und ausgewertet, dann in K muliplziert.)

Ausgabe: 0, wenn $b = c$, und 1, falls $b \neq c$.

[1] Falls $|K| < 2d$, muss man mit aus der Algebra bekannten Methoden eine Körpererweiterung $K' \supseteq K$ mit $|K'| \geq 2d$ konstruieren und in K' rechnen.

Algorithmus 6.1 ***Polynomprodukt***
Input: Polynome $f_1 \ldots, f_r$ und $g_1, \ldots, g_s$ über K mit Variablen $X_1, \ldots, X_n$.
Methode:
1 $d := \max\{\deg(f_1) + \cdots + \deg(f_r), \deg(g_1) + \cdots + \deg(g_s)\}$;
2 Wähle endliche Menge $S \subseteq K$ mit $|S| \geq 2d$;
3 Wähle $\boldsymbol{a} = (a_1, \ldots, a_n) \in S^n$ zufällig;
4 $b := f_1(\boldsymbol{a}) \cdot \ldots \cdot f_r(\boldsymbol{a})$;
5 $c := g_1(\boldsymbol{a}) \cdot \ldots \cdot g_s(\boldsymbol{a})$;
6 **if** $b = c$ **then return** 0 **else return** 1.

Es ist leicht zu sehen, dass Algorithmus 6.1 nur eine polynomielle Anzahl von Körperoperationen benötigt. (Selbst wenn die Exponenten in den Polynomen groß sind, ist – mit schneller Exponentiation, siehe Algorithmus 5.3 – die Anzahl der Operationen nur linear in der Bitlänge dieser Exponenten.)

Wir analysieren die Fehlerwahrscheinlichkeit.

1. Fall: $f = g$. – Dann ist $b = f(\boldsymbol{a}) = g(\boldsymbol{a}) = c$ für jedes $\boldsymbol{a}$, also ist die Ausgabe immer 0.

2. Fall: $f \neq g$. – Dann gilt:

$$\text{Ausgabe ist } 0 \;\Leftrightarrow\; (f-g)(\boldsymbol{a}) = 0.$$

Das heißt: Die Ausgabe ist fehlerhaft genau dann wenn der zufällig gewählte Vektor $\boldsymbol{a}$ eine Nullstelle von $f - g$ ist. Nun ist $f - g$ ein Polynom von einem Grad ≥ 0 (weil $f \neq g$) und höchstens d. Nach dem Satz von Schwartz-Zippel (Lemma 6.2) hat $f - g$ also höchstens $d|S|^{n-1}$ Nullstellen in S^n. Die Wahrscheinlichkeit, die falsche Ausgabe 0 zu erhalten, ist demnach höchstens

$$\frac{d|S|^{n-1}}{|S^n|} = \frac{d}{|S|} \leq \frac{1}{2}.$$

Es handelt sich bei Algorithmus 6.1 also um einen Monte-Carlo-Algorithmus mit einseitigem Fehler, Fehlerschranke $d/|S|$. Wenn wir ihn k-mal mit unabhängig gewählten Vektoren $\boldsymbol{a}$ durchführen, erhalten wir eine Fehlerschranke von $(d/|S|)^k$.

6.3 Polynomdeterminanten

Ganz ähnlich wie beim Vergleich von Polynomprodukten ist es bei Determinanten von Matrizen, deren Einträge Polynome sind. Gegeben sei also eine Matrix

$$A(X_1, \ldots, X_n) = \begin{pmatrix} f_{11} & \cdots & f_{1m} \\ \vdots & & \vdots \\ f_{m1} & \cdots & f_{mm} \end{pmatrix}$$

mit Polynomeinträgen $f_{ij} = f_{ij}(X_1, \ldots, X_n)$. Wir nehmen an, dass $d_0 \geq \deg(f_{ij})$ für alle i, j. (In Anwendungen sind die Einträge oft lineare Polynome, also $d_0 = 1$.) Aus der linearen Algebra kennt man die Determinante:

$$\det(A) = \sum_{\sigma \in S_m} \operatorname{sign}(\sigma) \cdot f_{1\sigma(1)} \cdot \ldots \cdot f_{m\sigma(m)}.$$

Dabei ist S_m die Menge aller Permutationen σ der Menge $\{1, \ldots, m\}$, und $\operatorname{sign}(\sigma)$ das Vorzeichen der Permutation σ.

Problem 1: Gegeben eine Polynommatrix A. Ist $\det(A) = 0$, also das Nullpolynom?

Problem 2: Gegeben eine Polynommatrix A und eine Folge $g_1, \ldots, g_s$ von Polynomen. Ist $\det(A) = g_1 \cdot \ldots \cdot g_s$?

Die Summe in der Definition der Determinante hat $n!$ Terme. Daher ist es normalerweise nicht möglich, das Polynom $\det(A)$ explizit zu berechnen. Auch hier hilft Randomisierung.

Sei $d = d_0 m$ oder eine andere obere Schranke für $\deg(\det(A))$. Wähle eine feste Menge $S \subseteq K$ mit $|S| \geq 2d$. Nun wähle zufällig eine Folge $\boldsymbol{a} = (a_1, \ldots, a_n) \in S^n$. Berechne

$$b = \det(A(\boldsymbol{a})) = \det \begin{pmatrix} f_{11}(\boldsymbol{a}) & \cdots & f_{1m}(\boldsymbol{a}) \\ \vdots & & \vdots \\ f_{m1}(\boldsymbol{a}) & \cdots & f_{mm}(\boldsymbol{a}) \end{pmatrix}$$

wie folgt: Erst wird $\boldsymbol{a}$ in jede Komponente eingesetzt, dann wird die Determinante in K ausgerechnet (Gauss-Elimination, $O(m^3)$ Körper-Operationen). Die Ausgabe ist 0, wenn $b = 0$ ist, und 1, wenn $b \neq 0$ ist. Ganz analog zur Analyse von Algorithmus 6.1 sieht man folgendes: Wenn $\det(A)$ das Nullpolynom ist, dann ist die Ausgabe 0, und wenn $\det(A) \neq 0$, dann gilt $\mathbf{Pr}(\text{Ausgabe ist } 0) \leq d/|S| \leq \frac{1}{2}$.

Ein analog gebauter Algorithmus löst auch Problem 2.

Eine graphentheoretische Anwendung der Polynomdeterminanten werden wir in Abschnitt 6.4 kennenlernen.

6.4 Perfektes Matching in bipartiten Graphen

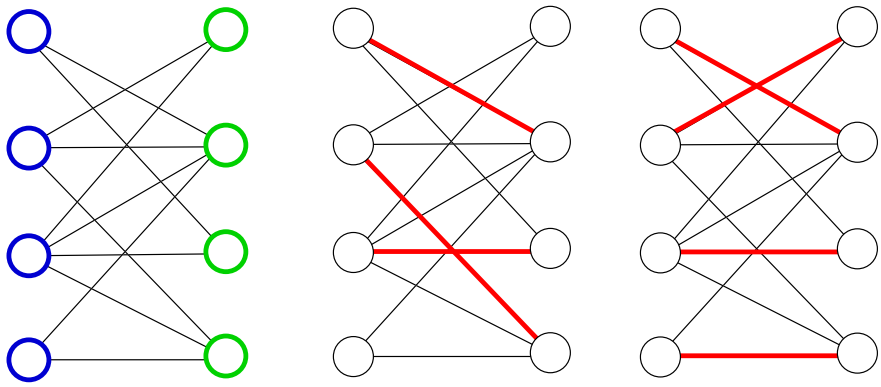

Abbildung 1: Ein bipartiter Graph, mit nicht erweiterbarem Matching, mit perfektem Matching

In diesem Abschnitt betrachten wir bipartite Graphen $G = (U, V, E)$. Dabei ist $U = V = \{1, \ldots, n\}$ und $E \subseteq U \times V$. Eine Menge $M \subseteq E$ heißt ein Matching („paarweise Zuordnung“) in G, wenn für $(u, v), (u'v') \in E$ gilt: $u = u' \Rightarrow v = v'$ und $v = v' \Rightarrow u = u'$. In Worten: Kanten in M sind entweder knotendisjunkt oder gleich. Ein Matching M heißt *perfekt*, wenn M aus n Kanten besteht – dann ist jeder Knoten in U genau einem Knoten in V zugeordnet.

Problem 1: Gegeben sei ein bipartiter Graph G.
Frage: Besitzt G ein perfektes Matching?

Problem 2: Gegeben sei ein bipartiter Graph G.
Aufgabe: Berechne ein perfektes Matching, wenn dies möglich ist.

Es gibt Standardalgorithmen, die diese Probleme lösen. Sie beruhen auf Flussberechnungen oder verwandten Verfahren. (Siehe Vorlesung „Effiziente Algorithmen“ im Master Informatik.) Wir stellen einen randomisierten Algorithmus für das Entscheidungsproblem vor. Dieser hat allerdings schlechtere Laufzeiten als die entsprechenden Flussalgorithmen. Der Vorteil liegt darin, dass er parallelisierbar ist, was für die Flussalgorithmen nicht gilt.

Definition 6.3
$G = (U, V, E)$ sei ein bipartiter Graph auf 2 mal n Knoten. Die **Edmonds-Matrix** $\boldsymbol{A_G}$ ist eine $n \times n$-Matrix mit Einträgen

$$f_{ij} = \left\{ \begin{array}{ll} X_{ij}, & \text{falls } (u_i, v_j) \in E, \\ 0, & \text{falls } (u_i, v_j) \notin E, \end{array} \right\}, \quad \text{für } 1 \leq i, j \leq n.$$

Dabei sind die Variablen X_{ij}, $1 \leq i, j \leq n$, alle verschieden.

Beispiel: Der bipartite Graph G aus Abb. 1 hat die Adjazenzmatrix bzw. Edmonds-Matrix

$$M_G = \begin{pmatrix} 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 \end{pmatrix} \quad \text{bzw.} \quad A_G = \begin{pmatrix} 0 & X_{12} & X_{13} & 0 \\ X_{21} & X_{22} & 0 & X_{24} \\ X_{31} & X_{32} & X_{33} & X_{34} \\ 0 & X_{42} & 0 & X_{44} \end{pmatrix}$$

Man sieht sofort, dass die Determinante von A_G entweder das Nullpolynom ist (also keinen Term enthält) oder Grad n hat.

Satz 6.4 ***Edmonds***
G hat ein perfektes Matching $\Leftrightarrow$ $\det(A_G) \neq 0$.

Die Determinante wird dabei über einem beliebigen Körper K berechnet. Der Satz sagt also, dass man nur testen muss, ob $\det(A_G)$ das Nullpolynom ist, um die Existenz eines perfekten Matchings zu testen.

Beweis von Satz 6.4: Aus der Definition der Determinante folgt:

$$\det(A_G) = \sum_{\sigma \in S_n} \underbrace{\operatorname{sign}(\sigma) \cdot f_{1\sigma(1)} \cdots f_{n\sigma(n)}}_{=: t_\sigma}.$$

Alle Terme in dieser Summe, die die 0 als Faktor haben, fallen heraus. Ein gegenseitiges Auslöschen anderer Terme ist unmöglich. Mit anderen Worten: Ein Summand $\operatorname{sign}(\sigma) \cdot f_{1\sigma(1)} \cdots f_{n\sigma(n)}$ ist genau dann in der Summe $\det(A_G)$ enthalten, wenn $f_{i\sigma(i)} = X_{i\sigma(i)}$ für alle i ist, und das heißt, dass E alle Kanten $(u_i, v_{\sigma(i)})$ enthält.

Das heißt: Die Summe enthält einen Term genau dann, wenn es ein σ gibt derart dass E alle Kanten $(u_i, v_{\sigma(i)})$ enthält. Dies ist äquivalent zur Existenz eines perfekten Matchings. □

Beispiel: Beim bipartiten Graph G aus Abb. 1 hat $\det(A_G)$ den Term $X_{13}X_{21}X_{34}X_{42}$, entsprechend dem perfekten Matching $M = \{(1,3), (2,1), (3,4), (4,2)\}$.

Algorithmus 6.2 ***Perfektes Matching, bipartit***
Input: Bipartiter Graph $G = (U, V, E)$ mit n Knoten auf beiden Seiten.
Methode:
1 Wähle Primzahl $p > 2n$; // Körper $\mathbb{Z}_p$, $S = \mathbb{Z}_p$
2 Bilde die Edmonds-Matrix A_G (eine $n \times n$-Matrix);
3 Wähle $\boldsymbol{a} = (a_{11}, \ldots, a_{1n}, \ldots, a_{n1}, \ldots, a_{nn}) \in \mathbb{Z}_p^{n \times n}$ zufällig;
4 $B := A_G(\boldsymbol{a})$; // bilde $A_G(\boldsymbol{a})$ durch Einsetzen von $\boldsymbol{a}$ in A_G
5 $b := \det(B)$; // Gauss-Elimination über $\mathbb{Z}_p$
6 **if** $b = 0$ **then return** 0 **else return** 1.

Wie vorher sehen wir: Wenn G keine perfektes Matching hat, also $\det(A_G)$ das Nullpolynom ist, dann gilt $\det(A_G(\boldsymbol{a})) = 0$ für alle $\boldsymbol{a}$, also ist die Antwort auf jeden Fall 0. Wenn G ein perfektes Matching hat, also $\det(A_G)$ nicht das Nullpolynom ist, dann gilt nach dem Satz von Schwartz-Zippel:

$$\mathbf{Pr}(\text{Antwort ist } 0) \leq \frac{\deg(\det(A_G))}{|\mathbb{Z}_p|} = \frac{n}{|\mathbb{Z}_p|} < \frac{1}{2}.$$

Bemerkung: (1) Das Bilden der Edmonds-Matrix im Algorithmus ist überflüssig; man kann die Matrix $A_G(\boldsymbol{a})$ direkt aus der Adjazenzmatrix M_G erhalten, indem man jeden 1-Eintrag in M_G durch ein Zufallselement aus $\mathbb{Z}_p$ ersetzt.
(2) Anstatt in $\mathbb{Z}_p$ könnte man auch in $\mathbb{Q}$ rechnen; man setzt dann z. B. $S = \{1, \ldots, 2n\}$. Der Nachteil ist, dass die Zahlen, mit denen man rechnet, bis zu $(n \cdot \log n)^2$ Bits haben können, so dass die einzelnen arithmetischen Operationen teuer sind.

Bemerkung: Man kann zeigen, dass man mit polynomiell vielen Prozessoren die Determinante einer $n \times n$-Matrix mit $O((\log n)^2)$ Ringoperationen berechnen kann (*Algorithmus von Berkowitz*). Damit kann man auch die Existenz eines perfekten Matchings in paralleler Zeit $O((\log n)^2)$ [Körperoperationen] testen – aber nur randomisiert. Für dieses Problem war bis 2016 kein schneller deterministischer paralleler Algorithmus bekannt.[2]

Problem 2, die Konstruktion eines perfekten Matchings, lässt sich wie folgt lösen: Zunächst testet man, ob G ein perfektes Matching besitzt. Nur im positiven Falle fährt man wir folgt fort. Setze $G' = (U, V, E') = G$. Man behandelt die Kanten $(u, v) \in E'$

[2] Aktualisierung 2018: Die Arbeit „Stephen A. Fenner, Rohit Gurjar, Thomas Thierauf: Bipartite perfect matching is in quasi-NC. STOC 2016. S. 754–763“ präsentiert einen deterministischen parallelen Algorithmus, der die Existenz eines perfekten Matchings mit einem „fast effizienten“ parallelen Algorithmus testet. Als effizient gelten normalerweise Algorithmen, die mit $n^{O(1)}$ („polynomiell vielen“) Prozessoren und paralleler Rechenzeit $(\log n)^{O(1)}$ auskommen. Der neue Algorithmus benutzt $n^{O(\log n)}$ („quasipolynomiell viele“) Prozessoren und $O((\log n)^2)$ parallele Zeit.

nacheinander, und verändert dabei eventuell den Graphen G'. Die Untersuchung von Kante $e = (u, v) \in E'$ verläuft so: teste, ob $G'' = (U, V, E' - \{e\})$ ein perfektes Matching besitzt. Wenn dies der Fall ist, streiche e aus E'. Am Ende testet man, ob E' ein perfektes Matching ist. Falls ja, wird E' ausgegeben, falls nein, kann man von vorne starten (wie bei Las-Vegas-Algorithmen üblich). Die Fehlerwahrscheinlichkeit bei einem Versuch kann man auf $1/n$ senken, indem man für jeden Test Algorithmus 6.2 $\log(|E| \cdot n)$-mal wiederholt.

Leider ist diese Methode zum Finden von perfekten Matchings wieder iterativ ($m = |E|$ Runden), also nicht leicht parallelisierbar. Effiziente parallele Algorithmen benötigen weitere Techniken, die im nächsten Abschnitt vorgestellt werden, im Kontext von Matchings in beliebigen ungerichteten Graphen.

6.5 Perfekte Matchings in beliebigen ungerichteten Graphen

Wir betrachten ungerichtete Graphen $G = (V, E)$ mit $V = \{1, \ldots, n\}$. Die Adjazenzmatrix A_G von G ist symmetrisch und hat Nullen auf der Hauptdiagonale. Es geht zunächst um den Test, ob G ein perfektes Matching $M \subseteq E$ besitzt, d. h. eine Kantenmenge M, in der jeder Knoten genau einmal vorkommt. Offensichtlich ist es hierfür notwendig, dass $n = |V|$ eine gerade Zahl ist. Die Größe $|M|$ eines perfekten Matchings muss $n/2$ sein.

Definition 6.5
Die **Tutte-Matrix** bmT_G zu einem Graphen G mit Knoten $1, \ldots, n$ ist eine $n \times n$-Matrix mit Einträgen

$$f_{ij} = \left\{ \begin{array}{ll} X_{ij}, & \textit{falls } i < j \textit{ und } (i,j) \in E, \\ -X_{ji}, & \textit{falls } j < i \textit{ und } (i,j) \in E, \\ 0, & \textit{sonst.} \end{array} \right\}, \quad \textit{für } 1 \leq i, j \leq n.$$

Beispiel: Zum Graphen $G = (\{1, 2, 3, 4\}, E)$ mit $E = \{(1, 2), (1, 3), (1, 4), (2, 4)\}$ gehört die Tutte-Matrix

$$T_G = \begin{pmatrix} f_{11} & f_{12} & f_{13} & f_{14} \\ f_{21} & f_{22} & f_{23} & f_{24} \\ f_{31} & f_{32} & f_{33} & f_{34} \\ f_{41} & f_{42} & f_{43} & f_{44} \end{pmatrix} = \begin{pmatrix} 0 & X_{12} & X_{13} & X_{14} \\ -X_{12} & 0 & 0 & X_{24} \\ -X_{13} & 0 & 0 & 0 \\ -X_{13} & -X_{24} & 0 & 0 \end{pmatrix}.$$

Offensichtlich gilt ${T_G}^{\mathsf{T}} = -T_G$, die Tutte-Matrix ist also *schiefsymmetrisch*. Die Einträge f_{ii} auf der Diagonalen sind 0. Auch für Tutte-Matrizen gibt es einen Zusammenhang zwischen Determinantenpolynom und der Existenz von perfekten Matchings in G.

Satz 6.6 ***Tutte***
Sei G ein beliebiger ungerichteter Graph. Dann gilt:
G hat ein perfektes Matching $\Leftrightarrow$ $\det(T_G) \neq 0$.

Beweis von Satz 6.6: Man schreibt

$$\det(T_G) = \sum_{\sigma \in S_n} \operatorname{sign}(\sigma) \cdot f_{1\sigma(1)} \cdots f_{n\sigma(n)}. \tag{1}$$

„$\Rightarrow$“: Sei M ein perfektes Matching in G. Dann betrachtet man die Permutation σ_M, die i auf j und j auf i abbildet genau dann wenn $(i,j) \in M$ ist. In der Summe für die Determinante kommt der Term $t_{\sigma_M} = \operatorname{sign}(\sigma_M) \cdot f_{1,\sigma_M(1)} \cdots f_{n,\sigma_M(n)}$ vor, und keiner der Faktoren ist 0, weil jedes Paar $(i, \sigma_M(i))$ einer Kante in E entspricht. Weil $\sigma^{-1} = \sigma$ gilt und die Permutation σ aus den Faktoren des Terms $f_{1\sigma(1)} \cdots f_{n\sigma(n)}$ abgelesen werden kann, kann es auch keinen anderen Term geben, der t_M auslöscht, also ist $\det(T_G)$ nicht das Nullpolynom.

„$\Leftarrow$“: Nun nehmen wir an, dass $\det(T_G)$ nicht das Nullpolynom ist. Wir betrachten die Summe in (1).

Jeder Term $t_\sigma = \operatorname{sign}(\sigma) \cdot f_{1,\sigma(1)} \cdots f_{n,\sigma(n)}$, der nicht 0 ist, definiert eine „*Spur*“

$$\operatorname{tr}_\sigma = \{(i, \sigma(i)) \mid 1 \leq i \leq n\},$$

eine Menge von geordneten Paaren, die wir als gerichtete Kanten auffassen. Jede dieser Kanten läuft entlang einer (ungerichteten) Kante in G. Weil σ Permutation ist, hat jeder Knoten i Eingangsgrad und Ausgangsgrad 1, also bildet tr_σ eine Menge von Kreisen, die Zyklen von σ entsprechen. Weil $f_{ii} = 0$ ist, haben diese Kreise eine Mindestlänge von 2.

Wir betrachten zunächst Permutationen σ, in deren Spuren ein Kreis ungerader Länge vorkommt. Wenn ein solches σ vorliegt, wählen wir den Kreis K_σ ungerader Länge in tr_σ, der den Knoten mit kleinster Nummer enthält. Wir drehen in tr_σ die Umlaufrichtung des Kreises K_σ herum. Dadurch entsteht die Spur $\operatorname{tr}_{\sigma'}$ einer eindeutig bestimmten Permutation σ'. Auf diese Weise entsteht eine eineindeutige Zuordnung $\sigma \mapsto \sigma'$ unter den Permutationen mit Spuren, die einen ungeraden Kreis haben. Diese Zuordnung ist *involutorisch*, d. h., sie ist gleich ihrer eigenen Umkehrung. Auf diese Weise werden die Permutationen σ mit einem ungeraden Kreis in tr_σ einander paarweise zugeordnet. – Es gilt dabei $\operatorname{sign}(\sigma) = \operatorname{sign}(\sigma')$, da das Vorzeichen einer Permutation nur von der Anzahl der geraden Kreise abhängt[3] und diese bei σ und σ'

[3] Genauer: $\operatorname{sign}(\sigma)$ ist 1 bzw. -1, je nachdem ob tr_σ eine gerade oder ungerade Anzahl gerader Kreise hat.

gleich ist. Sei K_σ der Kreis in tr_σ, dessen Kanten umgedreht werden. Dann gilt:

$$\begin{aligned} t_\sigma + t_{\sigma'} &= \mathrm{sign}(\sigma) \cdot f_{1,\sigma(1)} \cdots f_{n,\sigma(n)} + \mathrm{sign}(\sigma') \cdot f_{1,\sigma'(1)} \cdots f_{n,\sigma'(n)} && (2) \\ &= \mathrm{sign}(\sigma) \cdot \prod_{(i,\sigma(i)) \notin K_\sigma} f_{i,\sigma(i)} \cdot \Big(\prod_{(i,\sigma(i)) \in K_\sigma} f_{i,\sigma(i)} + \prod_{(i,\sigma(i)) \in K_\sigma} f_{\sigma(i),i} \Big) && (3) \\ &= \mathrm{sign}(\sigma) \cdot \prod_{(i,\sigma(i)) \notin K_\sigma} f_{i,\sigma(i)} \cdot \Big(\prod_{(i,\sigma(i)) \in K_\sigma} f_{i,\sigma(i)} + \prod_{(i,\sigma(i)) \in K_\sigma} (-f_{i,\sigma(i)}) \Big). && (4) \end{aligned}$$

Die letzte Gleichheit folgt dabei aus der Definition der Tutte-Matrix. Nun hat Kreis K_σ ungerade Länge, also haben die Produkte $\prod_{(i,\sigma(i)) \in K_\sigma} f_{i,\sigma(i)}$ und $\prod_{(i,\sigma(i)) \in K_\sigma} (-f_{i,\sigma(i)})$ ungerade viele Faktoren, und es folgt

$$\prod_{(i,\sigma(i)) \in K_\sigma} f_{i,\sigma(i)} + \prod_{(i,\sigma(i)) \in K_\sigma} (-f_{i,\sigma(i)}) = 0,$$

und damit $t_\sigma + t_{\sigma'} = 0$. Das heißt, dass die Permutationen σ, deren Spuren einen ungeraden Kreis enthalten, insgesamt nichts zu $\det(T_G)$ beitragen.

Wir haben angenommen, dass $\det(T_G)$ nicht das Nullpolynom ist. Nach dem eben erreichten Zwischenergebnis muss es dann eine Permutation σ mit $t_\sigma \neq 0$ geben, deren Spur tr_σ nur aus Kreisen gerader Länge besteht. Aus einer solchen Spur lässt sich aber leicht ein perfektes Matching M_σ konstruieren: Man nimmt aus jedem Kreis gerader Länge jede zweite Kante und ignoriert dann die Richtung. □

Der Algorithmus für den Test, ob $G = (V, E)$ ein perfektes Matching enthält, ist nun praktisch derselbe wie bei bipartiten Graphen. In der folgenden Formulierung lassen wir den Umweg über die Tutte-Matrix weg.

Algorithmus 6.3 ***Test auf perfektes Matching, ungerichteter Graph***
Input: (Ungerichteter) Graph $G = (V, E)$ mit n Knoten.
Problem: Teste, ob G ein perfektes Matching hat.
Methode:

```
0   Wenn n ungerade ist: return 0.
1   Wähle Primzahl p > 2n; // Körper Z_p, S = Z_p
2   Für Kante (i,j), i < j, wähle zufällige Zahl a_ij aus Z_p;
3   Für Kante (i,j), i > j, setze a_ij := -a_ji;
4   Für (i,j) ∉ E setze a_ij := 0;
5   Bilde Matrix A = (a_ij)_{1≤i,j,≤n};
6   b := det(A); // z. B. Gauss-Elimination über Z_p
7   if b = 0 then return 0 else return 1.
```

Zeitbedarf und Wahrscheinlichkeitsanalyse sind praktisch identisch zu dem bei Algorithmus 6.2. Ebenso gelten hier auch die Bemerkungen zur Parallelisierung.

Es sei bemerkt, dass deterministische Algorithmen zum Test auf die Existenz von perfekten Matchings in allgemeinen ungerichteten Graphen zwar existieren, aber ungleich komplizierter sind als die für bipartite Graphen.

Nehmen wir nun an, wir haben mit Algorithmus 6.3, eventuell nach mehreren Wiederholungen, festgestellt, dass Graph G ein perfektes Matching besitzt. Nun wollen wir ein solches perfektes Matching *berechnen*. Man könnte dazu vorgehen wie im Fall von bipartiten Graphen (s. Ende von Abschnitt 6.4), d. h. die Kanten (i, j) nacheinander darauf testen, ob der Graph ohne (i, j) immer noch ein perfektes Matching besitzt und (i, j) zu streichen, falls dies so ist.

Wir betrachten einen alternativen Algorithmus, der auch gut parallelisierbar ist. (Literatur für das Folgende: K. Mulmuley, U. V. Vazirani, V. V. Vazirani, Matching is as easy as matrix inversion, *Combinatorica* 7 (1): 105113. 1987.) Wir benötigen eine Tatsache aus der Algorithmik für lineare Algebra.

Fakt 6.7 ***Samuelson, Berkowitz***
$O(n^3)$ Prozessoren können in $O((\log n)^2)$ Zeit (d. h. Additionen und Multiplikationen) die Determinante einer Matrix $A \in K^{n \times n}$ berechnen.

Die Idee für den Matchingalgorithmus ist dann im Prinzip einfach: Für jede Kante $(i, j) \in E$ testet eine Gruppe von $O(n^3)$ Prozessoren (die jeweils eine Addition oder Multiplikation in K in einem Schritt ausführen können), z. B. durch Berechnen einer passenden Determinante, ob $(i, j) \in E$ Element eines „gesuchten“ perfekten Matchings ist.

Dabei stellt sich aber folgendes Problem: Ein Graph G kann viele verschiedene perfekte Matchings haben, und man müsste die Prozessoren dazu bringen, dass alle ihren Test bezüglich eines festen perfekten Matchings durchführen. Um eine solche Auswahl zu treffen, verwendet man wieder Randomisierung. Man versieht die Kanten in E mit zufällig gewählten Gewichten, und zwar so, dass es mit einer gewissen Wahrscheinlichkeit nur *ein* perfektes Matching M_0 mit minimalem Gesamtgewicht gibt. Dann zeigt man, dass eine Determinantenberechnung pro Kante (i, j) genügt, um zu testen, ob $(i, j) \in M_0$ gilt.

Definition 6.8
Ein (endliches) Mengensystem $(X, \mathcal{F})$ besteht aus einer Menge X mit $|X| = m \geq 1$ und einer Menge $\mathcal{F}$ von Teilmengen von X. Wenn eine Gewichtsfunktion

$$w \colon X \ni x \mapsto w(x) \in \mathbb{N}$$

gegeben ist, definieren wir $w(S)$ als $\sum_{x \in S} w(x)$, für $S \in \mathcal{F}$.

In unserer Anwendung wird $X = E$ sein und $\mathcal{F}$ die Menge aller perfekten Matchings in G. Gewichte für die Kanten werden zufällig gewählt.

Lemma 6.9 ***Isolationslemma***
Wenn man $w(x)$, für $x \in X$ aus $\{1, \ldots, 2m\}$ zufällig wählt, dann gilt:

$$\mathbf{Pr}(\text{in } \mathcal{F} \text{ gibt es eine eindeutig bestimmte Menge mit minimalem Gewicht}) \geq \frac{1}{2}.$$

(Anstelle des Faktors 2 kann man auch einen beliebigen anderen Faktor $r \geq 2$ wählen und erhält Wahrscheinlichkeit $\geq 1 - 1/r$.)

Beweis: Wir können o. B. d. A. annehmen, dass jedes $x \in X$ in einigen der $S \in \mathcal{F}$ vorkommt, in anderen nicht. (Wenn ein x in allen S vorkommt oder in keinem, dann hat $w(x)$ auf die Anordung der Gewichte $w(S)$, $S \in \mathcal{F}$ keinen Einfluss, und wir können dieses x für die Zwecke des Beweises ignorieren.)

Betrachte ein $x \in X$ und definiere:

$$\begin{aligned} W_x &:= \text{minimales Gewicht } w(S - \{x\}) \text{ über alle } S \in \mathcal{F} \text{ mit } x \in S; \\ \bar{W}_x &:= \text{minimales Gewicht } w(S) \text{ über alle } S \in \mathcal{F} \text{ mit } x \notin S. \\ \alpha_x &:= \bar{W}_x - W_x. \end{aligned}$$

Man beachte, dass die Zufallsvariablen W_x, $\bar{W}_x$ und α_x nur von $w(x')$ mit $x' \neq x$ abhängen, nicht von $w(x)$. Also sind α_x und $w(x)$ unabhängig.

Wir beobachten:

(i) Wenn $w(x) < \alpha_x$, dann muss *jede* Menge $S \in \mathcal{F}$, die minimales Gewicht hat, x enthalten:
Sei S_0 die Menge mit kleinstem Gewicht in $\mathcal{F}$, die x enthält. Dann ist

$$w(S_0) = w(S_0 - \{x\}) + w(x) = W_x + w(x).$$

Für jede Menge S', die x nicht enthält, gilt dann:
$w(S') \geq \bar{W}_x = \bar{W}_x - (W_x + w(x)) + w(S_0) = \alpha_x - w(x) + w(S_0) > w(S_0)$,
also kann $w(S')$ nicht minimal sein.

(ii) Wenn $w(x) > \alpha_x$, dann kann *keine* Menge S, die minimales Gewicht hat, x enthalten:
Sei S_0 die Menge mit kleinstem Gewicht in $\mathcal{F}$, die x enthält. Wie in (i) gilt $w(S_0) = W_x + w(x)$. Weil $0 > \alpha_x - w(x) = \bar{W}_x - (W_x + w(x)) = \bar{W}_x - w(S_0)$, gibt es eine Menge S' ohne x mit $w(S') < w(S_0)$. Die minimale Menge ist also unter denen zu suchen, die x nicht enthalten.

Wir nennen $x \in X$ *unentschieden*, wenn $w(x) = \alpha_x$.

Es gilt $\mathbf{Pr}(x \text{ unentschieden}) \leq \frac{1}{2m}$, weil α_x und $w(x)$ unabhängig sind und $w(x)$ zufällig in $\{1, \ldots, 2m\}$ ist.

Es folgt (Union Bound): $\mathbf{Pr}(\exists x \in X : x \text{ unentschieden}) \leq \frac{|X|}{2m} \leq \frac{1}{2}$.

Nehmen wir nun an, alle x sind *nicht* unentschieden, d. h. erfüllen $w(x) \neq \alpha_x$. Wir haben Folgendes gesehen:

(i) Wenn $w(x) < \alpha_x$, dann ist $x \in S$ für *jede* Menge S mit minimalem Gewicht.

(ii) $w(x) > \alpha_x$, dann ist $x \notin S$ für *jede* Menge S mit minimalem Gewicht.

Daraus folgt: Es gibt genau eine Menge mit minimalem Gewicht, nämlich $S = \{x \mid w(x) < \alpha_x\}$. □

Wir wenden das Isolationslemma auf die Menge $X = E$ und $\mathcal{F} = \{M \subseteq E \mid M \text{ ist perfektes Matching}\}$ an. Wir wählen also für jede Kante (i, j) in M ein zufälliges Gewicht w_{ij} aus $\{1, \ldots, 2m\}$, mit $m = |E|$. Dadurch bekommt jedes Matching M ein Gewicht $w(M) = \sum_{(i,j) \in M} w_{ij}$.

Nach dem Isolationslemma gibt es mit Wahrscheinlichkeit mindestens $\frac{1}{2}$ nur ein einziges Matching M_0 mit minimalem Gewicht.

Nun wollen wir für jede Kante $(i, j) \in E$ eine Prozessorengruppe (unabhängig von den anderen) testen lassen, ob $(i, j) \in E$ gilt. Hierzu bilden wir zunächst aus T_G eine neue Matrix $B = (B_{ij})_{1 \leq i,j \leq n}$, indem wir X_{ij} in T_G durch $2^{w_{ij}}$ ersetzen. Das heißt:

$$B_{ij} = \left\{ \begin{array}{ll} 2^{w_{ij}}, & \text{falls } i < j \text{ und } (i,j) \in E, \\ -2^{w_{ji}}, & \text{falls } j < i \text{ und } (i,j) \in E, \\ 0, & \text{sonst.} \end{array} \right\}, \quad \text{für } 1 \leq i, j \leq n.$$

Wir betrachten $\det(B)$.

Lemma 6.10
Wenn der Gewichtssatz $(w_{ij})_{1 \leq i,j \leq n}$ ein eindeutig bestimmtes minimales Matching M_0 bestimmt, dann ist $\det(B) \neq 0$. Genauer: Für $w_0 = w(M_0)$ gilt: 2^{2w_0} ist Teiler von $\det(B)$, aber 2^{2w_0+1} ist kein Teiler von $\det(B)$.

(Aus dem Lemma ergibt sich, dass man aus $\det(B)$ den Wert w_0 ablesen kann.)

Beweis: Wir argumentieren mit denselben Strukturen wie im Beweis des Satzes von Tutte. Es sei $B = (B_{ij})_{1 \leq i,j \leq n}$. Dann gilt

$$\det(B) = \sum_{\sigma \in S_n} \operatorname{sign}(\sigma) \cdot B_{1\sigma(1)} \cdots B_{n\sigma(n)}. \tag{5}$$

Im Polynom $\det(T_G)$ heben sich die Terme t_σ, deren Spuren tr_σ mindestens einen ungeraden Kreis haben, gegenseitig weg. Das gilt dann auch für die Terme in (5), deren Permutationen σ zu ungeraden Kreisen führen.

Wir betrachten jetzt einen beliebigen Summanden $\mathrm{sign}(\sigma)\cdot B_{1\sigma(1)}\cdots B_{n\sigma(n)}$, für dessen Permutation σ die Spur tr_σ nur Kreise gerader Länge besitzt. Diese Kreise kann man stets in zwei (nicht notwendig disjunkte) Matchings M_1 und M_2 zerlegen (jede zweite Kante eines Kreises kommt in M_1, die anderen in M_2). Aus der Wahl der Faktoren B_{ij} folgt

$$|B_{1\sigma(1)}\cdots B_{n\sigma(n)}| = \prod_{(i,j)\in M_1} 2^{w_{ij}} \cdot \prod_{(i,j)\in M_2} 2^{w_{ij}} = 2^{\sum_{(i,j)\in M_1} w_{ij} + \sum_{(i,j)\in M_1} w_{ij}} = 2^{w(M_1)+w(M_2)}.$$

Wenn $M_1 \neq M_0$ oder $M_2 \neq M_0$, dann ist $w(M_1) + w(M_2) > 2w_0$, also ist der Term zu σ durch 2^{2w_0+1} teilbar.

Es gibt in (5) nur einen Summanden mit geraden Kreisen, bei dem die Zerlegung der Kreise in zwei Matchings zu zwei Kopien von M_0 führt, und zwar ist das der Term zu der schon erwähnten Permutation σ_0 mit $\sigma_0(i) = j$ genau dann wenn $(i, j) \in M_0$ gilt. Der Term zu σ_0 hat Betrag 2^{2w_0}. Daher ist $\det(B)$ Summe von $\pm 2^{2w_0}$ und anderer Terme, die durch 2^{2w_0+1} teilbar sind. Die Behauptung des Lemmas folgt. □

Um zu entscheiden, ob eine Kante zu M_0 gehört oder nicht, muss man noch einen weiteren Trick anwenden. Zu Indexpaar (i, j), $i \neq j$, betrachten wir die „doppelte Streichungsmatrix“ $B^{(i,j)}$, die aus B entsteht, indem man Zeilen i und j und Spalten i und j durch Nullen ersetzt; nur an der Kreuzung von Zeile i mit Spalte j bleibt B_{ij} stehen, an der Kreuzung von Zeile j mit Spalte i bleibt B_{ji} stehen.

Bei der Bildung der Determinante $\det(B^{(i,j)})$ bleiben dann nur diejenigen Terme t_σ stehen, für die $\sigma(i) = j$ und $\sigma(j) = i$ gilt. Das heißt:

$$\det(B^{(i,j)}) = \sum_{\substack{\sigma\in S_n \\ \sigma(i)=j\wedge\sigma(j)=i}} \mathrm{sign}(\sigma)\cdot B_{1\sigma(1)}\cdots B_{n\sigma(n)}.$$

Wie vorher argumentiert man, dass Beiträge von Permutationen mit Spuren, die ungerade Kreise enthalten, sich gegenseitig aufheben. Beiträge zu $\det(B^{(i,j)})$ kommen also nur von Summanden, die $\sigma(i) = j$ und $\sigma(j) = i$ erfüllen und daneben nur gerade Kreise enthalten. Wieder kann man jeden dieser Terme als $2^{w(M_1)+w(M_2)}$ schreiben, wobei sowohl M_1 als auch M_2 die Kante (i, j) enthalten. Nun gibt es zwei Fälle.

1. Fall: $(i, j) \notin M_0$. – Dann ist $\det(B^{(i,j)})$ Summe von Termen $2^{w(M_1)+w(M_2)}$ mit $M_1 \neq M_0$ oder $M_2 \neq M_0$, so dass $w(M_1) + w(M_2) > 2w_0$ gilt. Also ist $\det(B^{(i,j)})$ durch 2^{2w_0+1} teilbar.

2. Fall: $(i,j) \in M_0$. Dann ist $\det(B^{(i,j)})$ Summe von $\pm 2^{2w_0}$ und anderen Termen, die alle durch 2^{2w_0+1} teilbar sind. Also ist $\det(B^{(i,j)})$ nicht durch 2^{2w_0+1} teilbar.

Wir haben gezeigt:

Lemma 6.11
Sei M_0 eindeutig bestimmtes Matching mit minimalem Gewicht w_0. Dann gilt:

$$(i,j) \in M_0 \qquad \Leftrightarrow \qquad \frac{\det(B^{(i,j)})}{2^{2w_0}} \text{ ist ungerade.}$$

Es ergibt sich der folgende Algorithmus für die Ermittlung eines perfekten Matchings.

Algorithmus 6.4 ***Finde perfektes Matching in ungerichtetem Graphen***
Input: Graph $G = (V, E)$ mit n Knoten und m Kanten; G hat perfektes Matching
Aufgabe: Finde ein perfektes Matching in G.
Methode:

```
1   Für Kante (i,j), i < j, wähle zufällige Zahl w_ij aus {1,...,2m}; B_ij := 2^{w_ij};
2   Für Kante (i,j), i > j, setze B_ij := -B_ji;
3   Für (i,j) ∉ E setze B_ij := 0;
4   Bilde Matrix B = (B_ij)_{1≤i,j,≤n};
5   b := det(B);
6   ermittle maximales k so dass 2^k Teiler von b ist;
7   if k ungerade then return „Fehlschlag“;
9   für jede Kante (i,j) ∈ E tue folgendes: // parallele Ausführung möglich
10    B^(i,j) := doppelte Streichungsmatrix von B (wie beschrieben);
11    q_ij := det(B^(i,j))/2^k;
12  M := {(i,j) | q_ij ungerade};
13  if |M| = n/2 und jedes i ∈ V kommt in M vor
14    then return M else return „Fehlschlag“
```

Satz 6.12
Algorithmus 6.4 ist ein Las-Vegas-Algorithmus mit polynomieller Laufzeit, der zu gegebenem Graphen G mit perfektem Matching entweder ein perfektes Matching oder „Fehlschlag“ ausgibt. Die Wahrscheinlichkeit für „Fehlschlag“ ist durch $\frac{1}{2}$ beschränkt. Der Algorithmus ist parallelisierbar (polynomiell viele Prozessoren, $O((\log n)^3)$ Zeit).

Beweis: Zur Korrektheit: Der Test in Zeilen 13–14 führt dazu, dass auf keinen Fall eine Menge M ausgegeben wird, die kein perfektes Matching in G ist. In allen anderen Fällen wird „Fehlschlag“ ausgegeben. Es handelt sich also um einen Las-Vegas-Algorithmus. Wir müssen die Wahrscheinlichkeit für die Ausgabe „Fehlschlag“ abschätzen. Nach dem Isolationslemma 6.9 erzeugt der Gewichtssatz w_{ij} mit Wahrscheinlichkeit höchstens $\frac{1}{2}$ mehrere perfekte Matchings in G mit minimalem Gewicht. Dies trägt also Wahrscheinlichkeit höchstens $\frac{1}{2}$ zur Fehlerwahrscheinlichkeit bei. (Zum Beispiel könnte k ungerade sein, oder Zeilen 9–12 berechnen eine Menge, die kein perfektes Matching ist.) Nun nehmen wir an, dass es nur ein perfektes Matching M_0 mit minimalem Gewicht gibt. Nach Lemma 6.10 ist dann k gerade und $w_0 = w(M_0) = k/2$. Nach Lemma 6.11 landet Kante (i, j) genau dann in der Menge M, wenn $(i, j) \in M_0$ ist. Es wird also ein perfektes Matching ausgegeben, nämlich M_0.

Zur Rechenzeit: Die Berechnung einer Determinante benötigt $O(n^3)$ Additionen und Multiplikationen in $\mathbb{Z}$. Es müssen $m + 1$ viele Determinanten berechnet werden. Durch die Berechnungen können allerdings Zahlen bis zu einer Ziffernanzahl von

$O(n^2 m \log n)$ entstehen, daher ist eine einzelne Multiplikation bis zu $O((n^2 m \log n)^2)$ teuer. Die gesamte Rechenzeit ist groß, aber immer noch polynomiell.

Zur parallelen Berechnung: Determinanten von $n \times n$-Matrizen können in paralleler „Zeit“ $O((\log n)^2)$ berechnet werden; dabei wird eine Addition oder Multiplikation in $\mathbb{Z}$ als Elementaroperation gezählt. Mit den schon erwähnten sehr langen Zahlen muss man auch für diese Elementaroperationen parallele Verfahren einsetzen. Mit $O(\ell^2)$ Prozessoren lassen sich zwei ℓ-ziffrige Zahlen in Zeit $O(\log \ell)$ addieren und multiplizieren. Damit kann man das gesamte Verfahren aus Algorithmus 6.4 parallel implementieren, mit polynomiell vielen Prozessoren und Rechenzeit $O((\log n)^3)$. □

6.6 Äquivalenztest für Read-Once-Branchingprogramme

Für die Erklärung des Modells „Read-Once-Branching-Programm“, das Äquivalenzproblem und die Algorithmen siehe die Vorlesung. Wir stellen hier nur ein technisches Lemma bereit (in der Vorlesung nicht bewiesen).

Ein Polynom $f(X_1, \ldots, X_n)$ über einem Körper heißt *multilinear*, wenn in jedem Term $a_{\ell_1,\ldots,\ell_n} X_1^{\ell_1} \cdots X_n^{\ell_n}$ die Exponenten $\ell_1, \ldots, \ell_n$ aus der Menge $\{0, 1\}$ kommen. Beispiele: $5X_1X_2X_4 + 2X_3 + 1$ ist multilinear, $X_1^2 + X_1X_2$ nicht.

Lemma 6.13
Es sei K ein Körper und f ein **multilineares** Polynom mit Variablen $X_1, \ldots, X_n$ und Koeffizienten aus K. Dann gilt: Wenn $f(a_1, \ldots, a_n) = 0$ für alle $a_1, \ldots, a_n \in \{0, 1\}$, dann ist f das Nullpolynom.

***Beweis*:** Wir zeigen durch Induktion über die Anzahl m der Variablen, die in f tatsächlich vorkommen: Wenn f nicht das Nullpolynom ist, dann gibt ein $(a_1, \ldots, a_n) \in \{0, 1\}^n$ mit $f(a_1, \ldots, a_n) \neq 0$.

I.A.: Sei $m = 0$. Dann ist f eine Konstante $c \neq 0$, also gilt $f(a_1, \ldots, a_n) = c \neq 0$ für alle $(a_1, \ldots, a_n) \in \{0, 1\}^n$.

I.V.: $m \geq 1$ und die Behauptung stimmt für alle $m' < m$.

I.Schritt: Angenommen, in f kommen $m \geq 1$ viele Variablen vor. Wir wählen eine davon, etwa X_1. Wir klammern X_1 aus allen Termen aus, in denen es vorkommt, und erhalten die Darstellung

$$f(X_1, \ldots, X_n) = X_1 \cdot g(X_2, \ldots, X_n) + h(X_2, \ldots, X_n).$$

(Höhere Potenzen von X_1 gibt es nicht, weil f multilinear ist.) Dabei ist g ein multilineares Polynom mit Variablen $X_2, \ldots, X_n$, das nicht das Nullpolynom ist, und in dem strikt weniger Variablen tatsächlich vorkommen als in f. Nach I.V. gibt es $a_2, \ldots, a_n \in \{0, 1\}$ mit $g(a_2, \ldots, a_n) \neq 0$. Die Werte $f(0, a_2, \ldots, a_n) = h(a_2, \ldots, a_n)$ und $f(1, a_2, \ldots, a_n) = g(a_2, \ldots, a_n) + h(a_2, \ldots, a_n)$ sind verschieden, also können wir $a_1 \in \{0, 1\}$ mit $f(a_1, a_2, \ldots, a_n) \neq 0$ wählen. □

6.7 Textsuche (String-Matching) mit Fingerprinting

6.7.1 Textvergleich

In diesem Abschnitt geht es um Algorithmen für Texte. Wir nehmen dabei o. B. d. A. stets an, dass wir ein Alphabet $\Sigma = \{0, \ldots, u-1\}$ benutzen, dessen Buchstaben natürliche Zahlen sind.

Zunächst betrachten wir ein Verfahren zum Vergleich zweier gleich langer Wörter $a = (a_1, \ldots, a_n)$ und $b = (b_1, \ldots, b_n)$. Wir wählen eine Primzahl $p > u$ und betrachten die Polynome

$$f_a = a_1 + a_2 X + a_3 X^2 + \cdots + a_n X^{n-1} \text{ und } f_b = b_1 + b_2 X + b_3 X^2 + \cdots + b_n X^{n-1},$$

jeweils über dem Körper $\mathbb{Z}_p$. Offenbar gilt: $f_a = f_b$ genau dann wenn $a = b$. Für $h = f_a - f_b$ gibt es zwei Fälle:

1. Fall: $a = b$. – Dann ist h das Nullpolynom; für jedes $r \in \mathbb{Z}_p$ gilt $f_a(r) = f_b(r)$.
2. Fall: $a \neq b$. – Dann ist h nicht das Nullpolynom. Da $\deg(h) \leq n-1$, hat h nach Lemma 6.1 nicht mehr als $n-1$ Nullstellen. Also gibt es höchstens $n-1$ Elemente $r \in \mathbb{Z}_p$ mit $f_a(r) = f_b(r)$.

Die Idee ist also, ein r aus $\mathbb{Z}_p$ zufällig zu wählen und zu testen, ob $f_a(r) = f_b(r)$ gilt. Die Werte $f_a(r)$ bzw. $f_b(r)$ können als (kurze) „Fingerabdrücke“ von $a = (a_1, \ldots, a_n)$ und $b = (b_1, \ldots, b_n)$ aufgefasst werden, die es gestattet, diese Wörter zu unterscheiden (wenn sie verschieden sind).

Algorithmus 6.5 ***Textvergleich mit Fingerprinting***
Input: $a = (a_1, \ldots, a_n)$ und $b = (b_1, \ldots, b_n)$ aus Σ^n, mit $\Sigma = \{0, \ldots, u-1\}$.
Methode:
1 Wähle eine Primzahl $p > \max\{u, 2n\}$;
2 Wähle zufällig ein r aus $\{0, \ldots, p-1\}$;
3 $\mathrm{fp}_a := (a_1 + a_2 \cdot r + a_3 \cdot r^2 + \cdots + a_n \cdot r^{n-1}) \bmod p$;
4 $\mathrm{fp}_b := (b_1 + b_2 \cdot r + b_3 \cdot r^2 + \cdots + b_n \cdot r^{n-1}) \bmod p$;
5 **if** $\mathrm{fp}_a = \mathrm{fp}_b$ **then return** 0 **else return** 1.

Die Polynomauswertungen in Zeilen 3 und 4 bewerkstelligt man in linearer Zeit mit dem bekannten Horner-Schema. Insgesamt kostet dieser Algorithmus also Zeit $O(n)$. Wenn $a = b$ ist, wird auf jeden Fall 0 ausgegeben. Wenn $a \neq b$ ist, gilt

$$\mathbf{Pr}(\text{Ausgabe ist } 0) = \frac{\#(\text{Nullstellen von } h = f_a - f_b)}{p} \leq \frac{n-1}{p} < \frac{1}{2}.$$

Interessant ist noch die folgende Beobachtung: Der Algorithmus ist durchführbar, selbst wenn sich die Daten a und b nicht am selben Ort (im selben Computer) befinden. Es genügt, am jeweiligen Ort die Fingerprints fp_a und fp_b zu berechnen und einen der beiden zu übermitteln. Dabei ist für eine Fehlerwahrscheinlichkeit von $1/2^\ell$ die Übermittlung von nur $\ell \cdot 2\log(\max\{u, 2n\})$ Bits nötig – viel weniger als wenn man die $n \log u$ Bits des gesamten Textes schicken würde.

6.7.2 Textsuche (Pattern matching, Algorithmus von Rabin-Karp)

Im Fall der Textsuche hat man es mit folgender Problemstellung zu tun:

Gegeben sind:

- „Text“ $t = (t_1, \ldots, t_n) \in \Sigma^n$ und
- „Muster“ $a = (a_1, \ldots, a_m) \in \Sigma^m$.

Die Frage ist, ob a als Teilwort $(t_i, \ldots, t_{i+m-1})$ in t vorkommt, und falls ja, an welcher Stelle bzw. welchen Stellen i.

Beispiel: Das Muster `bra` kommt im Text `abrakadabra` an den Stellen $i = 2$ und $i = 9$ vor.

Es muss gesagt werden, dass es für dieses sehr wichtige Problem auch hocheffiziente deterministische Algorithmen gibt. Dennoch ist die Betrachtung randomisierter Algorithmen interessant. Diese können auch leicht adaptiert werden, wenn es um mehrdimensionale „Texte“ und „Muster“ geht oder wenn das Muster unbestimmte Buchstabenpositionen hat („wildcards“).

Um (für $1 \le i \le n - m + 1$) a mit $(t_i, \ldots, t_{i+m-1})$ zu vergleichen, wollen wir die „Fingerabdruckspolynome“

$$f_i = f_i(X) = t_i X^{m-1} + t_{i+1} X^{m-2} + \cdots + t_{i+m-2} X + t_{i+m-1}$$

und

$$g = g(X) = a_1 X^{m-1} + a_2 X^{m-2} + \cdots + a_{m-1} X + a_m$$

über $\mathbb{Z}_p$ benutzen. Wie in Abschnitt 6.7.1 sieht man: Wenn $a \neq (t_i, \ldots, t_{i+m-1})$ ist, sind die Polynome f_i und g nicht identisch, also ist das Differenzpolynom $h_i = f_i - g$ nicht das Nullpolynom. Weil der Grad von h_i nicht größer als $m - 1$ ist, hat h_i nach Lemma 6.1 höchstens $m - 1$ Nullstellen in $\mathbb{Z}_p$. Wir folgern:

$$|\{r \mid 0 \le r < p \wedge h_i(r) = 0\}| \le m - 1. \tag{6}$$

Wir wählen hier $p > knm$, für ein beliebig festzusetzendes k (das auch von n, m abhängen darf). Wenn wir nun r aus $\{0, \ldots, p-1\}$ zufällig wählen, folgt aus (6):

$$\mathbf{Pr}_r(\exists i \leq n-m+1 \colon a \neq (t_i, \ldots, t_{i+m-1}) \wedge f_i(r) = g(r)) \leq \frac{(n-m)m}{p} < \frac{1}{k}. \quad (7)$$

Was soll nun der Vorteil dieses Verfahrens sein? Wir müssen $n-m+2$ Polynomauswertungen vornehmen. Auf naive Weise implementiert kostet dies Zeit $O(nm)$, und das ist ebenso langsam wie der naive Textsuchalgorithmus.

Der Trick ist, dass sich die Berechnung der Fingerabdrücke beschleunigen lässt. Aus

$$\begin{aligned} f_i(r) &= t_i \cdot r^{m-1} + & t_{i+1} \cdot r^{m-2} + \cdots + t_{i+m-2} \cdot r & & + t_{i+m-1} \quad \text{und} \quad (8) \\ f_{i+1}(r) &= & t_{i+1} \cdot r^{m-1} + \cdots + t_{i+m-2} \cdot r^2 & & + t_{i+m-1} \cdot r + t_{i+m} \end{aligned}$$

erhalten wir

$$f_{i+1}(r) = (f_i(r) - t_i \cdot r^{m-1}) \cdot r + t_{i+m}.$$

Das Körperelement r^{m-1} können wir in Zeit $O(\log m)$ vorab berechnen. Dann kann man aus $f_i(r)$ in Zeit $O(1)$ den nächsten Wert $f_{i+1}(r)$ berechnen. Als Algorithmus ergibt sich zusammengefasst folgendes:

Algorithmus 6.6 ***Textsuche mit Fingerprinting, Rabin-Karp***
Input: $a = (a_1, \ldots, a_m) \in \Sigma^m$ und $t = (t_1, \ldots, t_n) \in \Sigma^n$, $n \geq m$; Zahl $k \geq 2$.
Aufgabe: Finde die Positionen $i \in \{1, \ldots, n-m+1\}$ mit $a = (t_i, \ldots, t_{i+m-1})$.
Methode:

```
1    Wähle eine Primzahl p > nmk;
2    Wähle zufällig ein r aus {0,...,p-1};
3    f ← (t_1 · r^{m-1} + t_2 · r^{m-2} + ··· + t_{m-1} · r + t_m) mod p;   // Horner-Schema
4    g ← (a_1 · r^{m-1} + a_2 · r^{m-2} + ··· + a_{m-1} · r + a_m) mod p;   // Horner-Schema
5    i ← 1;
6    j ← m;
7    z ← r^{m-1} mod p;
8    if f = g then print(i);
9    while j < n do
10     j ← j + 1;
11     f ← ((f − t_i · z) · r + t_j) mod p
12     i ← i + 1;
13     if f = g then print(i);
14   enddo
```

Wir benennen die Eigenschaften dieses Algorithmus.

- Der Rechenaufwand ist $O(m)$ am Anfang und $O(1)$ in jedem der $n-m$ Schleifendurchläufe, insgesamt also $O(n)$.
- Mit (8) zeigt man Folgendes (durch Induktion über $i = 2, \ldots, n-m+1$): In Schleifendurchlauf für $(t_i, \ldots, t_{i+m-1})$ (Inhalt von `i` zu Beginn: $i-1$, nach Zeile 12: i) erhält die Variable `f` in Zeile 11 den Wert $f_i(r)$; dieser wird dann in Zeile 13 auf Gleichheit mit `g` getestet, das den Fingerprint g von a enthält. Wenn $a = (t_i, \ldots, t_{i+m-1})$ ist, muss sich hier Gleichheit ergeben und i wird ausgegeben; wenn $a \neq (t_i, \ldots, t_{i+m-1})$ ist, ist die Wahrscheinlichkeit, dass i ausgegeben wird, höchstens $(m-1)/(knm) < 1/(kn)$.
- Die Wahrscheinlichkeit, dass es ein i mit $a \neq (t_i, \ldots, t_{i+m-1})$ gibt, so dass i ausgegeben wird, ist höchstens $1/k$.

Wenn man nur das erste Vorkommen von a in t finden möchte, kann man den Algorithmus folgendermaßen variieren: Wenn i ausgegeben wird, wird die Schleife unterbrochen und es wird direkt geprüft, ob $a = (t_i, \ldots, t_{i+m-1})$ ist. Falls dies so ist, ist man fertig, falls nicht, wird die Bearbeitung der Schleife wieder aufgenommen. Der modifizierte Algorithmus kann niemals ein falsches Resultat liefern (es handelt sich also um einen Las-Vegas-Algorithmus), der erwartete zusätzliche Aufwand für diese Tests ist $m \cdot (1 + \frac{1}{k})$ Vergleiche zwischen Buchstaben. – Der direkte Kontrollvergleich ist problematisch, wenn man *alle* Vorkommen des Musters finden will. Das Muster könnte $\omega(n/m)$-mal in t vorkommen; dann wäre der Vergleichsaufwand $\omega(n)$, also nicht linear.

Bemerkung: (a) Die beschriebene Technik lässt sich auf höhere Dimensionen verallgemeinern. Für $d = 2$ wäre zum Beispiel der „Text“ eine Bitmap aus $m_1 \times m_2$ Pixeln, das Muster eine Bitmap aus $n_1 \times n_2$ Pixeln. Man soll herausfinden, ob das Muster im Text irgendwo auftaucht. Dies lässt sich mit Aufwand $O(m_1 \cdot m_2)$ bewerkstelligen. (Dies ist eine interessante Übungsaufgabe. Wenn man das Muster horizontal verschiebt, kommen ja etwa n_1 neue Bildpunkte hinzu und n_1 viele verschwinden. Wie kann man die Aktualisierung der entsprechenden Polynomwerte in $O(1)$ Zeit realisieren?)

(b) Wenn das Muster „wildcards“ oder „don't care“-Buchstaben, also unbestimmte Positionen, enthält, kann man eine Variante von Algorithmus 6.6 benutzen. Dabei ist für jede unbestimmte Buchstabenposition pro Runde eine konstante Anzahl von arithmetischen Operationen nötig.