

Übung 2

Max Wisniewski, Alexander Steen

Aufgabe 1

Gesucht ist ein $O(M(n))$ -Algorithmus zur Bestimmung des Betrags der Determinanten, wobei $M(n)$ die Anzahl der arithmetischen Operationen für die Matrizenmultiplikation von $n \times n$ -Matrizen ist.

Lösung:

Sei A eine invertierbare $n \times n$ -Matrix. Wir zerteilen die Matrix wie in der VL in folgende Teile:

$$A = XYZ = \begin{pmatrix} I & 0 \\ A_{21}A_{11}^{-1} & I \end{pmatrix} \begin{pmatrix} A_{11} & 0 \\ 0 & D \end{pmatrix} \begin{pmatrix} I & A_{11}^{-1}A_{12} \\ 0 & I \end{pmatrix}$$

wobei

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}, \quad D = A_{22} - A_{21}A_{11}^{-1}A_{12}$$

Wir nehmen vorerst an, dass A_{11} ebenfalls invertierbar ist. Wir kommen später darauf zurück. Für die Determinante von A gilt nun:

$$\begin{aligned} \det A &= \det \begin{pmatrix} I & 0 \\ A_{21}A_{11}^{-1} & I \end{pmatrix} \det \begin{pmatrix} A_{11} & 0 \\ 0 & D \end{pmatrix} \det \begin{pmatrix} I & A_{11}^{-1}A_{12} \\ 0 & I \end{pmatrix} \\ &= \det \begin{pmatrix} A_{11} & 0 \\ 0 & D \end{pmatrix} = \det A_{11} \det D \end{aligned}$$

Stellen wir nun die Rekursionsgleichung für $D(n)$ (Anz. Operationen für die Berechnung der Determinanten) auf und lösen sie:

$$\begin{aligned} D(1) &= 1 \\ D(n) &= 2 \cdot D\left(\frac{n}{2}\right) + 3O\left(M\left(\frac{n}{2}\right)\right) + \left(\frac{n}{2}\right)^2 + 1 \\ &= 2 \cdot D\left(\frac{n}{2}\right) + cM\left(\frac{n}{2}\right) \\ &= \dots \\ &= 2^k D\left(\frac{n}{2^k}\right) + c\left(M\left(\frac{n}{2}\right) + 2M\left(\frac{n}{4}\right) + \dots + 2^{k-1}M\left(\frac{n}{2^k}\right)\right) \\ &\stackrel{(*)}{=} 2^k D\left(\frac{n}{2^k}\right) + M(n) \cdot c \sum_{i=0}^k (2a)^i \\ &\stackrel{k=\log n}{=} 2^{\log n} 2^{\log n} + M(n) \cdot c' \\ &= O(M(n)) \end{aligned}$$

Umformung (*) gilt, da $M(\frac{n}{2}) \leq a \cdot M(n)$ für ein $a < 1/2$.

Die Annahme, dass A_{11} invertierbar ist, können wir treffen, da wir im Zweifelsfall den Algorithmus auf $B := A \cdot A^t$ anwenden, und dort die linke obere Teilmatrix immer invertierbar ist.

Da dann $\det B = (\det A)^2$ gilt hier $|\det A| = \sqrt{\det B}$.

Aufgabe 2

(a)

Zeigen Sie, dass man die Lösung eines Booleschen linearen Gleichungssystems $Ax = b$, wobei $A \in B^{k \times n}$, $b \in B^k$ effizient berechnen kann, falls eine existiert. Dabei ist $B = \{0, 1\}$, der Spaltenvektor x besteht aus den Variablen x_1, \dots, x_n , die Addition ist das logische oder und die Multiplikation ist das logische und.

Lösung:

Wenn wir die Gleichung auflösen kommen wir auf die folgende Form.

$$\begin{pmatrix} a_{11} \wedge x_1 \vee \dots \vee a_{1n} \wedge x_n \\ \dots \\ a_{k1} \wedge x_1 \vee \dots \vee a_{kn} \wedge x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ \dots \\ b_k \end{pmatrix}$$

Nun gehen wir nach dem folgenden Algorithmus vor:

```

for i = 1 to k do
  if  $b_i = 0$  then
    for j = 1 to n do
      if  $a_{ji} = 1$  then
         $x_i := 0$ 
remaining  $x_i := 1$ 

```

Wenn wir als Ergebnis eine 0 erhalten wollen, müssen alle Einträge, die ein $a_{ij} = 1$ haben genullt werden, da sonst insgesamt eine 1 herauskommen würde.

Nun wissen wir, dass die Verbleibenden Ergebnisse alle 1 sein müssen und da wir nichts negieren können ist das Problem monoton. Dies bedeutet, falls wir eine gültige Lösung haben, kann das ändern einer 0 auf eine 1 das Ergebniss nicht falsch machen. Daher ist, falls eine Lösung existiert, das setzen von allen (verbleibenden) Variablen auf 1 auch eine Lösung.

Damit haben wir einen Algorithmus, der in $O(n^2)$ läuft.

(b)

Was hat es mit der Lösbarkeit von $Ax' = b$, wobei $A \in B^{2k \times n}$, $b \in B^k$, $x = (x_1, \dots, x_n, \overline{x_1}, \dots, \overline{x_n})$ auf sich?

Lösung:

Sei P dieses Problem.

Claim 1. P ist NP-vollständig. ┐

Proof 1.

$SAT \preceq P$:

Sei Φ eine Formel in KNF. Sei k die maximale Anzahl von Literalen pro Klausel und n

die Anzahl der Klauseln.

Dann ist $A \in B^{2k \times n}$ die Matrix mit

$$a_{ij} = \begin{cases} 1 & , \text{Variable } j \text{ kommt in der } i \text{ Klausel vor} \\ 0 & , \text{sonst} \end{cases}$$

Dabei ist es egal ob die Variable negiert oder nicht negiert ist.

Der Vektor $b \in B^k$ ist der Vektor in dem jeder Eintrag eins ist.

Die Reduktion läuft in poly Zeit, da wir nur einmal über die Formel iterieren müssen. (Wahlweise ein weiteres mal, falls wir vorher k und n bestimmen wollen).

Das die erkannten Sprachen gleich sind, folgt offensichtlich, wenn man sich die aus-geschriebene Variante in Aufgabe 2a) anguckt, da es sich um die Definition von SAT handelt, wenn es nur 1en sind.

$P \in NP$:

Dies ist offensichtlich, da wir auf dem letzten Zettel gezeigt haben, dass die Boolesche Ma-trizenmultiplikation in Polyzeit durchführbar ist. Falls wir uns den Vektor x als Zeugen geben, können wir ihn also in poly Zeit verifizieren.

Aufgabe 3

Der transitive Abschluss A^* einer Booleschen $n \times n$ Matrix A ist definiert als

$$A^* = \bigvee_{i=0}^{n-1} A^i, \text{ wobei } A^0 = I_n \text{ ist.}$$

(a)

Was ist die Bedeutung von A^* , wenn A die Adjazenzmatrix eines gerichteten Graphen ist?

Lösung:

Ein Eintrag a_{ij}^* der Matrix A^* ist 1, wenn es einen Pfad von Knoten i zu Knoten j gibt; sonst 0. Der transitive Abschluss der Adjazenzmatrix ist also die "Zusammenhangsma-trix" des zugrundeliegenden Graphen.

(b)

Zeigen Sie, dass man zur Berechnung von A^* $O(M(n))$ Boolesche Operationen benötigt, wenn sich Boolesche $n \times n$ - Matrizen mit $M(n)$ Operaitonen berechnen lassen.

Lösung:

Nach Aufgabenzettel gilt folgende Indentität für den transitiven Abschluss einer $n \times n$ - Matrix $A = \begin{pmatrix} B & C \\ D & E \end{pmatrix}$, mit n gerade:

$$A^* = \begin{pmatrix} F & FCE^* \\ E^*DF & E^* \vee E^*DFCE^* \end{pmatrix}, \quad F = (B \vee CE^*D)^*$$

Sei $T(n)$ die Anzahl der Operationen zur Berechnung des transitiven Abschlusses einer $n \times n$ -Matrix. Aufstellen der Rekursionsgleichung liefert

$$\begin{aligned} T(1) &= 0 \\ T(n) &= 2T\left(\frac{n}{2}\right) + 7M\left(\frac{n}{2}\right) + 2\left(\frac{n}{2}\right)^2 \\ &= 2T\left(\frac{n}{2}\right) + cM\left(\frac{n}{2}\right) \end{aligned}$$

Das ist die gleiche Rekursionsgleichung wie in Aufgabe 1, ergibt also als Lösung $T(n) = O(M(n))$.

Die Rekursionsgleichung ist korrekt. Betrachten wir das System nur dem Schema entsprechend, haben wir zwei Komponenten V_1 ist die erste Hälfte der Knoten und V_2 die zweite Hälfte. Dann beschreiben die verschiedenen Teilmatrizen Kanten wie in Figure 1 gezeigt. B beschreibt Kanten die in V_1 verlaufen, E beschreibt Kanten die in V_2 verlaufen und C, D beschreibt Kanten, die V_1 und V_2 wechseln.

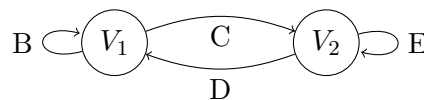


Tabelle 1: Rekursions Schema

Nun entsprechen die Teilmatrizen in etwa genau dem Kleene Algorithmus. Als neue Matrix B haben wir die Möglichkeit mit B in V_1 zu bleiben, oder wir gehen mit C in V_2 loopen dort mit E und kehren dann mit D nach V_1 zurück. Ähnlich gehen wir im neuen C vor. Dort können wir von und nach V_1 loopen und gehen dann einen Schritt nach V_2 und können auch dort loopen. Für den unteren Teil analog. Unserer Anschauung und dem Beweis des Kleene Algorithmus entsprechend sollten wir so alle möglichen Wege erhalten.

(c)

Wie lässt sich, unter der Berücksichtigung von (a), A^* mit Floyd-Warshall oder mit wiederholter Breitensuche berechnen? Vergleichen Sie die Laufzeiten mit der Laufzeit die Sie in Teil (b) ermittelt haben.

Lösung:

(1) Da der Floyd-Warshall-Algorithmus eine Matrix der kürzesten Wege (minimale Kosten) ausgibt, gibt es genau dann einen Weg wenn ein Matrixeintrag kleiner ∞ ist. Sei diese Matrix $\tilde{A}^* = (\tilde{a}_{ij}^*)$ eben diese Matrix. Dann erhält man die Matrix $A^* = (a_{ij}^*)$ durch

$$a_{ij}^* = \begin{cases} 1 & , \text{ falls } \tilde{a}_{ij}^* < \infty \\ 0 & \text{sonst} \end{cases}$$

Der Floyd-Warshall-Algorithmus hat eine Laufzeit von $O(n^3)$ und ist damit schlechter als die Matrizenmultiplikation, die mit $O(n^{2.3\text{blabla}})$ Operationen auskommt.

(2) Wiederholte Breitensuche: Man interpretiert die Eingabematrix als Adjazenzmatrix eines Graphen. Eine Breitensuche wird von jedem Knoten aus gestartet um alle Zusammenhangskomponenten zu finden. Damit kommt man auf eine Laufzeit von $n \cdot (|V| + |E|)$. Für Graphen mit wenigen Kanten ist diese Laufzeit nahe an n^2 (also besser), für Graphen mit vielen Kanten nah an n^3 (also schlechter).

(d)

Geben Sie einen effizienten Algorithmus zur Multiplikation von Matrizen an, die nur wenige Einträge $\neq 0$ haben. Analysieren Sie die Anzahl der Operationen als Funktion der Dimension n der Matrix und der Anzahl m der Nichtnullelemente.

Lösung:

Der folgende Algorithmus bekommt als Eingabe zwei $n \times n$ -Matrizen $A = (a_{ij})$, $B = (b_{ij})$.

```
1  for i = 1 to n do
2    for j = 1 to n do
3      tempsum := 0;
4      for k = 1 to n do
5        if  $a_{ik} \neq 0 \wedge b_{kj} \neq 0$  then
6          tempsum := tempsum +  $a_{ik} \cdot b_{kj}$ 
7        end
8      end
9       $c_{ij} := \text{tempsum}$ 
10   end
11 end
```

Wir sehen, dass wir nun nur noch für jedes nicht Nullelement multiplizieren und addieren müssen. Wenn wir es nach oben abschätzen, müssen wir für jedes nicht Nullelement nur n Multiplikationen und Addition ausführen. Da wir nur arithmetische Operationen zählen und die Vergleiche nichts kosten erhalten wir so einen Algorithmus mit der Laufzeit $O(m * n)$.

Dies erhält die ursprüngliche Laufzeit, da wir bei einer vollbesetzten Matrix $O(n^3)$ Operationen brauchen.