Julius-Maximilians-Universität Würzburg
Fakultät für Mathematik und Informatik

SS12, Dr. Spoerhase

Exakte Algorithmen

Nils Wisiol

12. Juni 2012

Inhaltsverzeichnis

1 Einführung

Hier fehlt noch etwas.

2 Dynamisches Programmieren

Hier fehlt noch etwas

15.5.12

Hier fehlt noch etwas

Lemma 1. *Sei $\alpha \leq 1/2$. Dann gilt*

$$\sum_{i=0}^{\alpha \cdot n} \binom{n}{i} = O^*(2^{h(\alpha)\cdot n}),$$

wobei $h(\alpha) = -\alpha \log_2 \alpha - (1-\alpha)\log_2(1-\alpha)$.

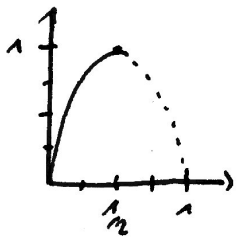


Abbildung 2.1: Graph des Binomialkoeffizienten

Beweis. Es ist $\sum_{i=0}^{\alpha n} \leq n\binom{n}{\alpha n} = O^*(\binom{n}{\alpha n})$, denn die Binomialkoeffizienten $\binom{a}{b}$ steigen für $b \leq n/2$ monoton an. Per Definition gilt

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}.$$

Die Fakultät kann abgeschätzt werden durch $\sqrt{2\pi n}(n/e)^n \leq n! \leq 2\sqrt{2\pi n}(n/e)^n$, also ist $n!$ proportional zu $(n/e)^n$. Daraus folgt, dass

$$\begin{aligned}\binom{n}{\alpha n} &= O^*\left(\frac{(n/e)^n}{(\alpha n/e)^{\alpha n}((1-\alpha)n/e)^{(1-\alpha)n}}\right) = O^*\left(\alpha^{-\alpha n}(1-\alpha)^{-(1-\alpha)n}\right) \\ &= O^*\left(2^{-\alpha \log_2 \alpha n} \cdot 2^{-(1-\alpha)\log_2(1-\alpha)n}\right),\end{aligned}$$

woraus die Behauptung folgt. □

Satz 2. *Eine kleinste dominierende Mänge lässt sich in $O(1,7088^n)$ ermitteln.*

Beweis. Zunächst bestimmen wir eine nicht-erweiterbare unabhänge Menge I. Falls $|I| \leq \alpha n$, testen wir in $O^*(2^{h(\alpha)n})$ alle Teilmengen $D \subseteq V$ mit $|D| \leq |I|$. Falls $|I| > \alpha n$, wende Satz **??** an und berechne kleinste dominierende Menge in $O^*(2^{(1-\alpha)n})$.

Aus der Skizze ergibt sich die Laufzeit als das Maximum der beiden dargestellten Funktionen bei $\alpha^* \leq 0,22711$ bzw. $O(2^{0,7729n}) = O(1,7088^n)$. □

Der schnellste derzeit bekannte Algorithmus für dieses Problem benötigt ca. $O(1,5^n)$ und stammt aus 2010.

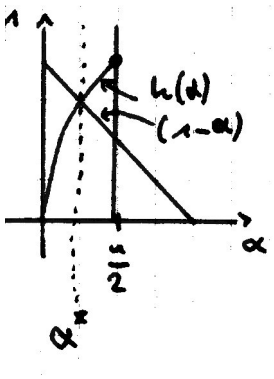
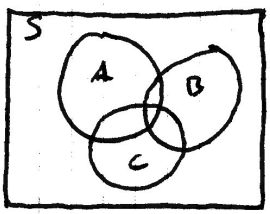


Abbildung 2.2: Bestimmung von α^* als Maximum der zwei möglichen Funktionen

3 Inklusion-Exklusion



Gehe von einem Problem aus, bei es leicht ist zu zählen, welche Elemente aus dem Universum S die Eigenschaft A oder B, A und B, ... erfüllen; es aber schwer ist zu zählen, wie viele Elemente diese Eigenschaften nicht besitzen. Es ergibt sich jedoch der Zusammenhang

$$|\overline{A \cup B \cup C}| = |S| - (|A| + |B| + |C|) + (|A \cap B| + |A \cap C| + |B \cap C|) + (|A \cap B \cap C|).$$

Sind N Objekte und eine Menge $P = \{P_1, ...P_n\}$ von Eigenschaften gegeben, bezeichnen wir für jedes $S \subseteq P$ mit $N(S)$ die Anzahl der Objekte, die (mindestens) die Eigenschaften in S erfüllen. Mit $N(0)$ bezeichnen wir die Anzahl der Objekte, die keine der Eigenschaften erfüllen. Wir können oben skizzierte Formel dann verallgemeinern, es ergibt sich

Satz 3. $N(0) = \sum_{S \subseteq P} (-1)^{|S|} N(S) = N(\emptyset) + \sum_i N(P_i) + \sum_{i \leq j} N(P_i, P_j) + ...$

Beweis. Elemente des Universums, die keine der Eigenschaften besitzen, werden auf beiden Seiten der Gleichung einmal gezählt. Betrachte nun die Elemente, die genau die Eigenschaften $S = \{P_{i_1}, ..., P_{i_s}\}$, $|S| = s$. Diese werden genau in den $N(S')$ mit $S' \subseteq S$ gezählt. Daraus folgt, dass diese Objekte zur rechten Seite der Gleichung jeweils $\sum_{S' \subseteq S} (-1)^{|S'|} = \sum_{i=0}^{s} \binom{s}{i} \cdot (-1)^i = (-1+1)^s = 0$ beitragen. Objekte, die mindestens eine Eigenschaft besitzen, werden also auf der rechten Seite nicht gezählt. □

22.5.12

Korollar 4. *$N(P) = \sum_{S \subseteq P} (-1)^{|S|} \overline{N}(S)$, wobei $\overline{N}(S)$ die Anzahl der Objekte mit keiner der Eigenschaften in S ist.*

Beweis. Es sei P_i' die Eigenschaft, dass ein Objekt die Eigenschaft P_i nicht besitzt. Es ist dann $N'(0) = N(P)$ und $N'(P_1', ...P_k') = \overline{N}(P_1, ..., P_k)$. Es folgt die Behauptung. □

Hamiltonpfad. Wir betrachten nun das gerichtete Hamiltonpfadproblem. Gegeben sei ein gerichteter Graph $G = (V, E)$ sowie Knoten $s, t \in V$. Das gerichtete Hamiltonpfadproblem bezeichnet das Problem, ob ein einfacher s-t-Pfad existiert, der alle Knoten in V besucht. Dieses Problem ist NP-schwer. Brute force benötigt $O^*(n!)$ Zeit. Wir geben einen Algorithmus an, der $O^*(2^n)$ Zeit und polynomiellen Speicherplatz benötigt.

Satz 5. *Die Anzahl der s-t-Hamiltonpfade kann in $O^*(2^n)$ Zeit und mit polynomiellen Speicherplatzverbrauch ermittelt werden.*

Beweis. Wir verwenden das Inklusion-Exklusion-Prinzip. Als Objekte sehen wir alle s-t-Pfade der Länge $n-1$ an – auch solche, die nicht einfach sind. Die Eigenschaften $v \in V$ sind definiert als „Pfad besucht den Knoten v“. Also ergibt sich $N(V)$ als die Anzahl der s-t-Hamiltonpfade. Zur Berechnung von $N(V)$ bestimmen wir zunächst für $W \subseteq V$ jeweils $\overline{N}(W)$, die Anzahl der s-t-Pfade der Länge $n-1$, die W vermeiden.

Dies erreichen wir durch dynamische Programmierung. Das Programm berechnet für ein festes $W \subseteq V \setminus \{s,t\}$, ein $k = 0, ..., n-1$ und ein $u \in V \setminus W$ die Anzahl der s-u-Pfade der Länge k, die W vermeiden. Diese Zahl nennen wir $\overline{N}(W,u,k)$. Es folgt $\overline{N}(W,t,n-1) = \overline{N}(w)$. Die Berechnung erfolgt für $k = 0$ durch

$$\overline{N}(W,u,0) = \begin{cases} 1 & (u = s) \\ 0 & (\text{sonst}) \end{cases}$$

und für $k > 0$ durch

$$\overline{N}(W,u,k) = \sum_{vu \in E, v \notin W} \overline{N}(W,v,k-1)$$

Die Laufzeit des dynamischen Programms für ein festes W ist $O(n \cdot m)$ mit $m = |E|$. Durch Wiederverwendung von Speicherplatz erreicht man einen Speicherverbrauch von $O(n \cdot \log 2^m) = O(n \cdot m)$.

Zur Berechnung des Endergebnisses iterieren wir über alle $W \subseteq V \setminus \{s,t\}$ und berechnen jeweils $\overline{N}(w)$ mit dem oben angegebenen dynamischen Programm. Wir addieren den Wert des Ausdrucks $(-1)^{|W|} \cdot \overline{N}(w)$ und erhalten $N(V)$ als Wert dieser Summe. Die Gesamtlaufzeit ergibt sich damit als $O(m \cdot n \cdot 2^n)$, der Gesamtspeicherbedarf als $O(m \cdot n)$, da der Speicher wiederverwendet werden kann. □

Graph-Färbung. Wir betrachten nun das Problem der Graph-Färbung. Als Eingabe sei ein Graph $G = (V,E)$ gegeben. Eine *zulässige Färbung* weißt jedem Knoten so einer Farbe zu, dass keine zwei benachbarten Knoten die selbe Farbe haben. Gesucht ist die minimale Anzahl an Farben, mit denen dies möglich ist. Brute force benötigt die Zeit $O^*(n^n)$. Wir geben einen Algorithmus an, der $O^*(2^n)$ Zeit und Speicherplatz benötigt.

Wir nennen die Menge aller Knoten einer bestimmten Farbe eine *Farbklasse*. Für korrekte Färbungen ist eine Farbklasse eine unabhängige Knotenmenge, daher können wir das Graphfärbungsproblem auch als Problem, die kleine Anzahl an unabhängigen Knotenmengen $I_1, ..., I_k$, die V partitionieren, formulieren. Dies ist verwand zum Set-Cover-Problem $(\mathcal{I}, V)$, wobei $\mathcal{I}$ die Menge aller unabhängigen Mengen ist. Verallgemeinert betrachten wir daher Set-Cover-Instanzen (S,U), bei denen U explizit gegeben ist und S in $O^*(2^n)$ mit $n = |U|$ aufgezählt werden kann.

Wir nennen ein k-Tupel $(S_1, ..., S_k) \in S^k$ ein *geordnetes k-Cover*, falls $\bigcup_{i=1}^k S_i = U$ und geben einen Algorithmus an, der die Anzahl der geordnetes k-Cover ermittelt. Für $W \subseteq U$ sei $S[W]$ definiert als $\{S_i \in S : S_i \cap W = \emptyset\}$ sowie $s[W] = |S[W]|$.

Lemma 6. *Die Anzahl der geordneten k-Cover beträgt $c_k = \sum_{W \subseteq U} (-1)^{|W|} s[W]^k$.*

5.6.12

Beweis. Wir betrachten die k-Tupel $(S_1, ..., S_k)$ als Objekte und für $v \in U$ den Ausdruck $v \in \bigcup_{i=1}^k S_i$ als Eigenschaft, und wenden darauf die Inklusions- / Exklusionsformel an. Die Eigenschaft $v \in U$ für ein k-Tupel gilt also nicht, wenn $v \notin \cup_{i=1}^k S_i$. Ist ein $W \subseteq U$ gegeben, so gelten alle Eigenschaften aus W nicht, wenn für alle $v \in W$ stets $v \notin \bigcup_i S_i$ gilt. Anders formuliert, muss für alle i der Schnitt $S_i \cap W = \emptyset$ sein. Da es genau $s[W]$ Mengen gibt, die W vermeinden, gibt es $s[W]^k$

Möglichkeiten, ein k-Tupel aus Mengen zu bilden, die W vermeiden. Es ist damit $\overline{N}[W] = s[W]^k$. Satz 3 liefert dann

$$c_k = N[U] = \sum_{W \subseteq U} (-1)^{|W|} s[W]^k$$

als Anzahl der Tupel, für die alle Eigenschaften $v \in U$ gelten. Dies ist die Anzahl der Tupel, in denen für jedes $v \in U$ gilt dass $v \in \bigcup_i S_i$ ist. □

Satz 7. *Die Anzahl c_k aller k-Cover kann in $O^*(2^n)$ berechnet werden.*

Beweis. Es ist $c_k = \sum_{W \subseteq U} (-1)^{|W|} s[W]^k$. Zum Auswerten der Formel müssen alle Teilmengen $W \subseteq U$ aufgezählt und jeweils $s[W]$ berechnet werden. Dazu ordnen wir die Elemente von U durch $U = \{u_1, ..., u_n\}$ beliebig. Dann sei für alle $W \subseteq U$

$$g_i(W) = |\{S_i \in S[W] : \{u_1, ..., u_i\} \setminus W \subseteq S_i\}|.$$

Dies ist die Anzahl der Mengen, die W vermeiden, und alle Elemente $\{u_1, ..., u_i\} \setminus W$ enthält. Mit dieser Definition gilt $s[W] = g_0(W)$, denn $g_0(W) = |\{S_i \in S[W] : \emptyset \subseteq S\}| = |S[W]|$. Wir verwenden ein dynamisches Programm, das über $i = n, ..., 0$ iteriert und zunächst

$$g_n(W) = \begin{cases} 1 & (U \setminus W \in S) \\ 0 & (\text{sonst}) \end{cases}$$

berechnet. Für $0 \leq i < n$ ist dann

$$g_{i-1}(W) = \begin{cases} g_i(W) & (u_i \in W), \\ g_i(W \cup \{u_i\}) + g_i(W) & (u_i \notin W). \end{cases}$$

Im Fall $u_i \in W$ ändert sich durch den Ausschluss von W in der Definition von g_i nichts an der Bedingung, daher bleibt die Anzahl gleich. Im anderen Fall können zwei Fälle eintreten, deren Möglichkeiten addiert werden. $g_i(W \cup \{u_i\})$ erfasst die Elemente, die u_i vermeiden, während $g_i(W)$ die Möglichkeiten enthält, die u_i enthalten. Für jede Menge $W \subseteq S$ wird der Ausdruck $g_i(W)$ n-mal ausgewertet, daher beträgt die Laufzeit $O^*(2^n)$. Der Speicherverbrauch liegt bei $O(2^n)$, wenn man zu jeder Zeit immer nur die vorhergehende Zeile in der dynamischen Tabelle speichert. Somit kann die Formel mit Hilfe von diesem Algorithmus in $O(2^n)$ Speicher und Zeit ausgewertet werden. □

4 Measure & Conquer

Measure and Conquer bezeichnet ein Verfahren, um Rekursonsgleichungen wie beispielsweise $T(n) \leq T(n-1) + T(n-5)$ zu lösen. Ein Algorithmus für unabhängige Mengen ist

Name: *MIS*
Data: Graph G
Result: Größte Anzahl von unabhängigen Mengen
if *ex.* v *mit* $d(v) = 0$ **then**
  return $1 + \text{MIS}(G - v)$
else if *ex.* v *mit* $d(v) = 1$ **then**
  return $1 + \text{MIS}(G - N[v])$
else if $\Delta(G) \geq 3$ **then**
  wähle v mit $d(v) \geq 3$
  return $\max\{1 + \text{MIS}(G - N[v]), \text{MIS}(G - v)\}$
else if $\Delta(G) \leq 2$ **then**
  löse diesen Spezialfall effizient
end

Algorithm 1: Algorithmus zur Berechnung der größten Anzahl unabhängiger Mengen

In Fall 3 handelt es sich um Branching, die Laufzeit wird hier durch die Rekursonsgleichung $T(n) \leq T(n-1) + T(n-4)$ beschrieben. Die Standardanalyse liefert eine Abschätzung von $O(1,3803^n)$. Diese Analyse kann durch die Methode Measure & Conquer verbessert werden.

Dominating Set. Wir betrachten nun einen Algorithmus für Dominating Set, der bei konventioneller Analyse Laufzeit $O(1,9052^n)$, bei Analyse mit Measure & Conquer jedoch $O(1,5259^n)$ ergibt. Dazu modellieren wir Dominating Set als Set Cover-Instanz, wobei $U = V$ und $S = \{N[v] : v \in V\}$ ist. Die Größe dieser Set Cover-Instanz ist dann $|U| + |S| = n + n = 2n$. Wir entwickeln daher einen Algorithmus für Set Cover mit $O^*(c^{|U|+|S|})$ und $c \ll 2$.

Wir nehmen ohne Einschränkung an, dass $U = \bigcup_{S_i \in S} S_i$. Damit ist eine Set Cover-Instanz durch S spezifiziert. Die *Häufigkeit* eines Elements $u \in U$ sei definiert als die Anzahl der Mengen aus S,

die u enthalten.

Name: *SC*
Data: Set Cover-Instanz S
Result:
if *(1)* $|S| = 0$ **then**
  return 0
else if *(2) es existieren* $S_1, S_2 \in S$ *mit* $S_1 \subseteq S_2$ **then**
  return $\mathrm{SC}(S - S_1)$
else if *(3) es existiert* $u \in U(S)$ *mit Häufigkeit 1, d.h. liegt in genau einem* $S^* \in S$ **then**
  return $1 + \mathrm{SC}(\mathrm{del}(S^*, S))$
end
wähle Menge $S' \in S$ mit maximaler Kardinalität ;
if *(4)* $|S'| = 2$ **then**
  löse in Polynomialzeit (vgl. Übungsaufgabe) ;
else if $|S| \geq 3$ **then**
  $S_{in} = S - S^*$;
  $S_{out} = \mathrm{del}(S, S^*)$;
  return $\min\{\mathrm{SC}(S_{in}), 1 + \mathrm{SC}(S_{out})\}$
end

Algorithm 2: Algorithmus für Set Cover

Dabei ist $\mathrm{del}(A, S) = \{T : T = R \setminus A \neq \emptyset, R \in S\}$.

Für die konventionelle Analyse sei $k'(S) = |S| + |U(S)|$. Wir können aufgrund des Algorithmus die Rekursionsungleichung

$$T(k') \leq T(k' - 1) + T(k' - 4)$$

herleiten. Die übliche Lösungsmethode ergibt $T(k') = O(1,3803^{k'})$. Damit ergibt sich als Laufzeit für Dominating Set $O(1,3803^{2n}) = O(1,9052^n)$.

Für die Analyse mittels Measure & Konquer definiert man n_i als Anzahl der Mengen mit Kardinalität i und m_j als Anzahl der Elemente mit Häufigkeit j. Das Maß sei

$$k(S_i) = \underbrace{\sum_{i \geq 1} w_i n_i}_{k_w(S_i)} + \sum_{j \geq 1} v_j m_j$$

mit Gewichten $w_i, v_j \in [0, 1]$. Damit ist $k(S_i) \leq |U| + |S|$. Vereinfachend nehmen wir für die Gewichte an, dass

(a) $w_i \leq w_{i+1}, v_i \leq v_{i+1}$,

(b) $w_1 = v_1 = 0$,

(c) $w_1 = v_1 = 1$ für $i \geq 6$,

(d) $\Delta w_i \geq \Delta w_{i+1}$ mit $\Delta w_i := w_i - w_{i-1}$ und Δv_i analog.

Wir bestimmen den Branching-Vektor $(\Delta k_{out}, \Delta k_{in})$ in Abhängigkeit der w_i und v_i.

(1) Im Fall, dass S_i nicht gewählt wird, verändert sich Δk_{out} wie folgt.

  (i) Durch Wegfallen von S_i wird $k_w(S)$ umd $w_{|S_i|}$ reduziert.

(ii) Sei r_i definiert als Anzahl der Elemente von S_i mit Häufigkeit i. Die Reduktion von $k_v(S)$ ist dann kleiner oder gleich $\sum_{i=2}^{6} r_i \Delta v_i$.

(iii) Angenommen $r_2 > 0$. Seien $R_1, ..., R_h$, $h \leq r_2$ die Mengen ungleich S_i, die mindestens ein Element der Häufigkeit 2 mit S_i teilen. Die Mengen R_i werden nachfolgend durch Regel (3) reduziert. Sei dazu $v_{2,i}$ die Anzahl solcher Elemente in R_i. Dann ist $|R_i| \geq v_{2,i} + 1$ und $|S_i| \geq v_{2,i} + 1$. Die Reduktion von $k_w(S)$ durch Wählen von R_i ist dann mindestens $W_{v_{2,i}+1}$. Damit existiert mindestens ein Element, das nicht in S_i liegt und aus der Instanz entfernt wird. Also wird $k_v(S)$ mindestens um v_2 reduziert.

Wir erhalten eine Reduktions von mindestens

$$\Delta k' = \begin{cases} 0 & (r_2 = 0), \\ v_2 + w_2 & (r_2 = 1), \\ v_2 + w_2 + w_3 & (r_2 = 2), \\ v_2 + w_4 & (r_2 \geq 3, |S_i| = 3), \end{cases}$$