Julius-Maximilians-Universität Würzburg
Fakultät für Mathematik und Informatik

SS12, Dr. Spoerhase

Exakte Algorithmen

Nils Wisiol

22. Mai 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 $h(\alpha) = -\alpha \log_2 \alpha - (1-\alpha)\log_2(1-\alpha)$.

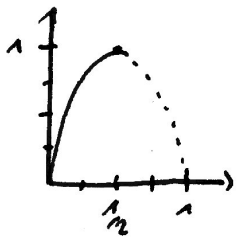


Abbildung 2.1: Graph des Binomialkoeffizienten

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.

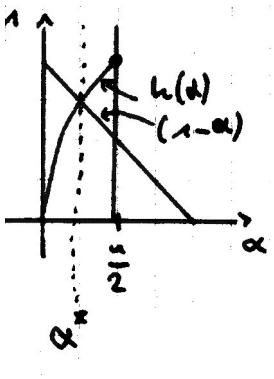
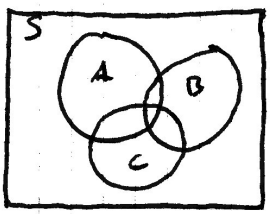


Abbildung 2.2: Bestimmung von α^* als Maximum der zwei möglichen Funktionen

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

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. Es ist damit $\overline{N}[W] = s[W]^k$. Satz 3 liefert dann

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

□