Karlsruher Institut für Technologie
Fakultät für Informatik
ITI Wagner

Algorithmische Geometrie
Sommersemester 2012
Martin Nöllenburg/Andreas Gemsa

Übungsblatt 4 - Lineare Programmierung

1 Korrektheitsbeweis

In der Vorlesung wurde der Algorithmus RandomPermutation(A) vorgestellt.

a) Beweise seine Korrektheit, indem du zeigst, dass jede mögliche Permutation von A gleich wahrscheinlich ist.

b) Zeige außerdem, dass die Aussage aus a) nicht mehr stimmt, wenn wir $r \leftarrow \text{Random}(k)$ durch $r \leftarrow \text{Random}(n)$ ersetzen.

Lösung:
Zu a): Zu zeigen ist, dass jeder beliebige Permutation mit Wahrscheinlichkeit $1/n!$ von RandomPermutation erzeugt wird. Das ist leicht einzusehen, denn die Wahrscheinlichkeit im ersten Durchlauf der Schleife ein bestimmtes Element aus der Menge A zu wählen ist $1/n$. Im zweiten Durchlauf verringert sich die Anzahl der möglichen Elemente um 1, also ist die Wahrscheinlichkeit nun ein bestimmtes auszuwählen $1/(n-1)$, usw. Also ist die Wahrscheinlichkeit in der k-ten Iteration ein bestimmtes Element auszuwählen $1/(n-k+1)$. Damit ist die Wahrscheinlichkeit eine bestimmte Permutation zu erzeugen:

$$\frac{1}{n} \cdot \frac{1}{n-1} \cdot \ldots \cdot 1 = \frac{1}{n!}$$

Zu b): Angenommen A besteht aus $a_1, \ldots, a_n$. Erzeuge einen Baum mit Wurzel $a_1, \ldots, a_n$. Die Wurzel hat n Kinder, jedes Kind ist dabei eine Permutation von der Wurzel. Diese Knoten haben jeweils wieder n Kinder usw. Am Schluss hat dieser Baum n^n Blätter. Jedes Blatt entspricht einer möglichen Permutation die mit gleicher Wahrscheinlichkeit ausgegeben werden kann. Da $n^n/n!$ im Allgemeinen nicht ganzzahlig ist folgt die Behauptung.

2 Paranoia

Algorithmus 1: ParanoidMax

Eingabe : Endliche Menge $A \subset \mathbb{R}$
Ausgabe : Maximum $\max_{a \in A} a$ der Menge
if $|A| = 1$ **then**
  return einziges Element $a \in A$
else
  a = zufällig gewähltes Element aus A
  b = ParanoidMax($A \setminus \{a\}$)
  if $b \geq a$ **then**
    return b
  else
    prüfe unnötigerweise jedes Element aus $A \setminus \{a\}$, um sicherzugehen, dass a wirklich größer ist
    return a

Betrachte Algorithmus 1, der das Maximum einer Menge von Zahlen berechnet. Schätze die asymptotische Worst-Case-Laufzeit des Algorithmus scharf ab. Beachte die zufällige Auswahl des Elementes a und zeige, dass die erwartete Laufzeit echt besser ist.

Lösung:
Worst-Case Fall: Bei jedem Durchlauf von ParanoidMax wird für a immer das Maximum gewählt. Die Rekursiongleichung für die Laufzeit lautet dann:

$$T(n) = \mathcal{O}(n) + T(n-1)$$

Damit gilt $T(n) \in \mathcal{O}(n^2)$. Diese Schranke ist scharf.

Aber erwartete Laufzeit: Die Wahrscheinlichkeit, dass in einem Schleifendurchlauf tatsächlich das Maximum gewählt wird ist $1/i$ wenn die Menge aus i Elementen besteht. Nur dann ist der Aufwand $O(i)$. Das heißt insgesamt ergibt sich für die erwartete Laufzeit: Sei X_i eine Zufallsvariable die genau dann 1 ist, wenn im i-ten Schleifendurchlauf das Maximum gewählt ist und 0 sonst. Für die Laufzeit folgt:

$$\sum_{i=1}^{n} O(i) \cdot X_i$$

Für die erwartete Laufzeit also:

$$E[\sum_{i=1}^{n} O(i) \cdot X_i] = \sum_{i=1}^{n} O(i) \cdot E[X_i]$$

Es bleibt noch das $E[X_i]$ zu begrenzen. Die Wahrscheinlichkeit, dass X_i im i-ten Durchlauf 1 ist, ist genau $1/i$. Es folgt:

$$E[\sum_{i=1}^{n} O(i) \cdot X_i] = \sum_{i=1}^{n} O(i) \cdot E[X_i] = \sum_{i=1}^{n} O(i) \cdot 1/i = O(n)$$

3 Züge

Gegeben seien n Züge, die auf parallelen Gleisen fahren. Jeder Zug z_i $(i = 1, \ldots, n)$ fahre dabei mit konstanter Geschwindigkeit v_i und nehme zum Startzeitpunkt $t = 0$ (hier startet nur die Zeit, die Züge haben schon die volle Geschwindigkeit) die Position k_i auf der Strecke ein. Gebe einen Algorithmus an, der in $O(n \log n)$ Zeit berechnet, welche Züge bis zu einem gegebenen Zeitpunkt $t_{\text{stop}} > 0$ mindestens einmal in Führung waren.

Hinweis: Der in der Vorlesung vorgestellte Algorithmus zu Berechnung vom Schnitt mehrerer Halbebenen könnte dabei nützlich sein.

Lösung:
Bilde für jeden Zug z_i eine Gerade die als y-Achsenabschnitt ihre Startposition k_i hat und als Steigung die Geschwindigkeit v_i. Trage diese Geraden in ein Koordinatensystem, bei der die x-Achse die Zeit und die y-Achse die Distanz ist, ein (für eine Illustration s. Übungsfolien). Wir sind an der oberen Kontur interessiert. Die können wir ganz leicht über den Algorithmus zum Schnitt mehrerer Halbebenen berechnen.

4 Rückblick: Polygon Partitionieren

In der dritten Vorlesung wurde ein Verfahren vorgestellt welches, mit Hilfe von einer doppelt verketteten Kantenliste (DCEL), ein einfaches Polygon in monotone Polygone auftrennt. Im Verlauf des Algorithmus werden neue Kanten zur DCEL hinzugefügt, nämlich Diagonalen um Split– oder Merge–Knoten zu entfernen. Dabei wurde angenommen, dass wir eine Kante in konstanter Zeit zur DCEL hinzufügen können.

Argumentiere, zunächst warum das Hinzufügen von Kanten in eine DCEL im Allgemeinen *nicht* in konstanter Zeit möglich ist. Zeigen dann, wie man im Fall des in der Vorlesung vorgestellten Algorithmus zur Polygon-Partitionierung Kanten trotzdem in $O(1)$ in die DCEL einfügen kann.

Lösung:
Zwei Probleme:

1.) Facetteninformationen müssen aktualisiert werden, sobald eine Diagonale eingefügt wird. Dieser Aufwand ist linear in der Facettengröße.

Lösung:
Facetteninformationen spielen für den Algorithmus keine Rolle. Diese Daten werden einfach nicht von der DCEL verwaltet und müssen demnach auch nicht aktualisiert werden.

2.) Wenn wir eine neue Diagonale einfügen: Wie wählen wir in $O(1)$ die richtigen Kanten für das Anpassen der next(e) bzw. prev(e) Einträge?

Lösung:
Wir können zeigen, dass jeder Knoten im Verlauf des Algorithmus nur $O(1)$ zusätzliche Inzidenzen bekommt. Dazu:

- Initial hat jeder Knoten genau Grad 2
- Jeder Knoten ist höchstens einmal helper Knoten: +1 neue Inzidenzen
- Jeder Knoten ist höchstens einmal Merge-/Split-/...-Knoten: +2 neue Inzidenzen

Insgesamt hat jeder Knoten maximal Grad 5 und damit liegt der Grad jedes Knotens in $O(1)$. Um die passenden Einträge zu bekommen müssen wir die Knoten radial sortieren und können dann für eine neue Diagonale in $O(1)$ ihre Nachbarn an dem Knoten bestimmen.