9.2 Zufallsgesteuerte Algorithmen

- ❑ Einordnung
  - deterministischer Algorithmus, der als zusätzliche Elementaroperation Zufallsexperimente durchführen kann
  - Ziel: Beschleunigung von Algorithmen
  - oftmals: Trade-off zwischen Korrektheit und Effizienz
- ❑ Komplexität
  - Laufzeit kann bei gleicher Eingabe variieren
  - expected-case Komplexität
- ❑ Klassen von zufallsgesteuerten Algorithmen
  - Monte-Carlo-Algorithmen
    - ⇨ liefern korrektes Ergebnis nur mit einer gewissen Fehlerwahrscheinlichkeit, sind aber in jedem Fall effizient
  - Las-Vegas-Algorithmen
    - ⇨ liefern stets ein korrektes Ergebnis, terminieren aber nur mit einer bestimmten Wahrscheinlichkeit

Entwurf zufallsgesteuerter Algorithmen

- ❑ Auswahl
  - Aus einer Familie von deterministischen Algorithmen wird einer ausgewählt
- ❑ Stichprobe
  - Anhand einer zufällig gewählten Stichprobe wird auf Eigenschaften aller Daten geschlossen
- ❑ Zeuge
  - Durch n-maligen Ablauf kann die Fehlerwahrscheinlichkeit gesenkt werden.
- ❑ Unrodnung
  - Eingabe wird vor der Bearbeitung zufällig umgeordnet; danach geht die Abarbeitung evtl. schneller
- ❑ Komplexität
  - mittlere erwartete Komplexität
    - ⇨ $T_E(n) = \sum_{x \in U_n} p(x) E[T(x)]$
  - maximal erwartete Komplexität

$$T_{E\max} = \max_{x \in U_n} E[T(x)]$$

Monte-Carlo-Algorithmen

- Korrektheit
  - n-maliges Durchlaufen und dann Bestimmung des häufigsten Ergebnisses der Aufrufe (Abstimmungsverfahren); dies ist dann mit Wahrscheinlichkeit (1-p)^n richtig.
- Beispiel: Monte-Carlo-Integration
  - Der Erwartungswert einer Funktion f: R --> R im Intervall [a, b] ist
    $$E[f(x)] = \frac{1}{b-a}\int_a^b f(x)dx$$
  - Erwartungswert kann durch den Wert von n Funktionswerten approximiert werden
    $$E[f(x)] = \lim_{x\to\infty}\frac{1}{n}\sum_{i=1}^{n} f(x_i)$$
  - In der Praxis muß man oft $n > 10^6$ wählen
  - Monte-Carlo-Integration kann dann sinnvoll sein, wenn man kein gutes numerisches Verfahren hat

Probabilistische Primzahltests

- Große Primzahlen spielen in kryptographischen Protokollen eine entscheidende Rolle
- Ausgangspunkt
  - Für eine Zahl $n > 2$ soll entschieden werden, ob es sich um eine Primzahl handelt
- Deterministischer Algorithmus
  - Klassische Divisionsmethode: testet, ob n durch 2 oder eine ungerade Zahl aus dem Bereich $\{3, ..., \sqrt{n}\}$ teilbar ist.
  - Hat exponentiellen Aufwand
  - bis heute kein deterministischer Algorithmus mit polynomialem Aufwand bekannt
- Gesucht: effizientes Testkriterium
- Grundlagen
  - Multiplikative Gruppe modulo n: $Z_n^* = \{a \in \{1,...,n-1\} \mid ggT(a,n) = 1\}$
    - ⇨ teilerfremden Zahlen zu n
  - Eulersche Funktion $|Z_n^*| = \Phi(n) = n\prod_{p/n}\left(1-\frac{1}{p}\right)$
    - ⇨ beschreibt Mächtigkeit der multiplikativen Gruppe
  - Satz von Euler
    - ⇨ Für alle $a \in Z_n^*$ $gilt$ $a^{\Phi(n)} \equiv 1 \mod n$
    - ⇨ Ist n eine Primzahl, dann gilt $\Phi(n) = n-1$
  - Satz von Fermat
    $$a^{n-1} \equiv 1 \mod n$$

Probabilistische Primzahltests

- Satz von Fermat ist nützliches Kriterium, falls Exponentation effizient durchzuführen ist.
  - Verwende beispielsweise a=2
  - Welche Aussage kann mit solch einem Test gefunden werden?

Algorithmus von Miller-Rabin

- Monte-Carlo Algorithmus
- basiert auf den bisherigen Überlegungen zu Primzahltests
  - In $s \in N$ aufeinanderfolgenden Iterationen wird eine Basis a zufällig und gleichverteilt aus den Zahlen {1, ..., n-1} ausgewählt und dem Test
    $$a^{n-1} \equiv 1 \mod n$$
    unterzogen
    - ⇨ Das zugrundeliegende Konstruktionsparadigma nennt man Random Search
  - Verwendet Test: Zeuge (a, n) für die Überprüfung anstelle $a^{n-1} \neq 1 \mod n$
    - ⇨
```
d = 1;
for i = k-1 downto 0 {
      d = (d * d) % n;
      if b[i] == 1 {d = (d * a) % n;}
}
return not d;
```
    - ⇨ $b_k, b_{k-1}, ..., b_0$ ist Binärdarstellung von n-1

- Wieso handelt es sich hierbei um einen Monte-Carlo Algorithmus?
- Aufwand ?

Fehleranalyse des Miller-Rabin Algorithmus

- Eine Abschätzung der Fehlerwahrscheinlichkeit ist für Monte-Carlo Algorithmen wichtig
- Satz
  - Für $n > 2$, ungerade und prim, gibt es mindestens $(n-1)/2$ Zeugen für die Zusammengesetztheit Z_n^*
- Beweis ("Outline")
  - Beweis zeigt, daß nicht mehr als $(n-1)/2$ Zeugen existieren
  - Jeder Nichtzeuge a muß zu Z_n^* gehören
  - Wäre ggT$(a, n) > 1$, dann hat $a*x = 1 \bmod n$ keine Lösung
  - Untergruppe B von
  - Satz von Lagrange
- Fehlerwahrscheinlichkeit
  - Random-Search liefert in jeder Iteration mit einer Wahrscheinlichkeit von mindestens 1/2 einen Zeugen für die Zusammengesetztheit, falls n keine Primzahl ist
  - Fehlerwahrscheinlichkeit ist höchstens $1/2^s$

Las-Vegas Algorithmen

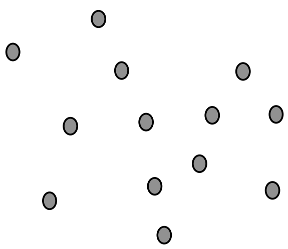
- Liefern immer das richtige Ergebnis, wenn sie terminieren.
  - allgemeiner Las Vegas Algorithmus: terminiert mit $p > 1/2$
  - Macao Algorithmus: terminiert mit Sicherheit
- Randomisierte Binärbäume
  - Beispiel für einen Macao Algorithmus
  - Menge S (Schlüssel) von natürlichen Zahlen n liegt vor. Der nächste Schlüssel wird zufällig und gleichverteilt aus den noch nicht eingefügten Schlüsseln gezogen und in den bisherigen Suchbaum eingefügt.
    - ⇨ wann bringt das beispielsweise Vorteile?
- Analyse der Laufzeit
  - Erwartete Anzahl von Schlüsselvergleichen?

Geometrische Algorithmen

- ❏ fundamental für graphische Datenverarbeitung
- ❏ Beispiele
  - Bestimme alle Punkte, die innerhalb eines Rechtecks liegen
  - Bestimme alle Paare sich schneidender Rechtecke einer Rechteckmenge
  - Bestimme den Umriß einer Punkt- oder Rechteckmenge (konvexe Hülle)
  - ...
- ❏ Beispiel:
  - Eine Fläche sei durch einen geschlossenen Polygonzug gegeben. Liegt ein gegebener Punkt p innerhalb einer Fläche?
    - ⇨ sende von p einen Radarstrahl aus (Winkel beliebig)
    - ⇨ zähle, wieviele Liniensegmente des Polygonzugs er schneidet
      - ⇨ ungerade Zahl: p liegt innerhalb des Polygonzugs
      - ⇨ gerade Zahl: p liegt außerhalb des Polygonzugs
    - ⇨ Implementierung benötigt Funktion zur Bestimmung des Schnittpunktes einer Halbgeraden mit einem Liniensegment

Konvexe Hülle

- ❏ Konvexes Gebiet $G \subseteq R^2$
  - zu je zwei Punkten liegt die Verbindungslinie komplett in G
- ❏ Konvexe Hülle H einer endlichen Punktmenge G
  - Teilmenge der Punkte, so daß die Verbindungslinien ein konvexes Gebiet bilden, in dem alle Punkte liegen
- ❏ Beispiel

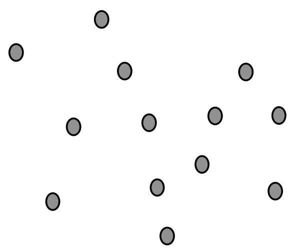


- ❏ klassischer Algorithmus ("Einwickeln")
  - wähle Punkt mit der kleinsten y-Koordinate.
  - nehme einen horizontalen, in die positive Richtung verlaufenden Strahl und "schwenke" ihn nach oben, bis er auf einen weiteren Punkt trifft.
  - neuer Punkt wird als Drehpunkt benutzt usw. bis Anfangspunkt wieder erreicht ist
  - Aufwand $O(n^2)$, falls alle Punkte der konvexen Hülle angehören

Konvexe Hülle

- Divide-and-Conquer Verfahren
  - für <= 3 Punkte ist die konvexe Hülle trivial
  - sonst bestimme die 4 Punkte mit minimaler bzw. maximaler *x*- bzw. *y*-Koordinate.
  - Alle punkte innerhalb des durch diese Pakete definierten Rechtecks liegen innerhalb und brauchen nicht weiter betrachtet werden
  - die 4 Gebiete außerhalb des Rechtecks werden rekursiv behandelt
  - die Gesamthülle wird aus den Zwischenergebnissen zusammengesetzt
  - wesentlich schneller: O(*n* ln *n*)

- Beispiel



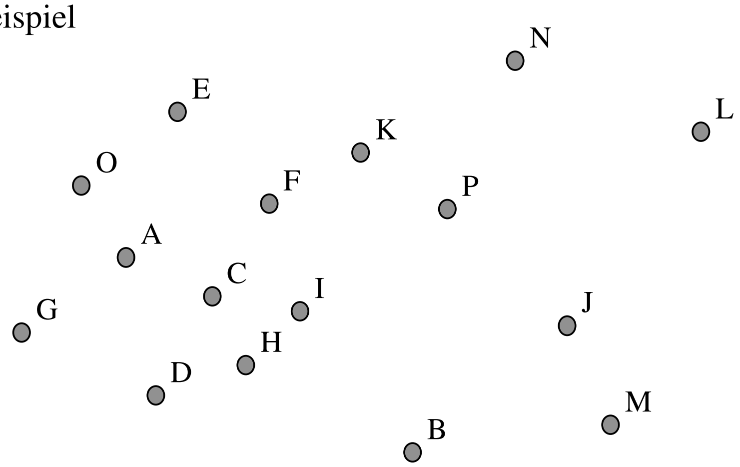
Bereichssuche

- Gegeben
  - Menge von Punkten
- Gesucht
  - Welche Punkte fallen in einen bestimmten Bereich
  - z.B. alle Städte im Umkreis von 100 km von Braunschweig

- Eindimensionaler Fall
  - Aufbau eines balancierten binären Suchbaums (Intervallsuche)
  -
```
s := {};
if p <> null {
 if p < p.key {s = intervallsuche(p.leftson, x, y);}
 if p > p.key {s = s + intervallsuche(p.rightson, x, y);}
 if ((x <= p.key) && (p.key <= y)) {s = s + {p.key};}
}
return s;
```

Mehrdimensionale Bereichssuche

- Gesucht
  - Welche Punkte von $P \subseteq R^2$ liegen in einem gegebenen Rechteck
- Zweidimensionale Suchbäume
  - erlauben Intervallsuche sowohl für die x-Koordinate als auch für die y-Koordinate
  - binäre Suchbäume mit Punkten in den Knoten
    - ⇨ x- und y-Koordinaten werden in alternierender Folge als Schlüssel verwendet
    - ⇨ als Wurzel wird y-Koordinate verwendet
  - Beispiel



    - ⇨ Achtung: den resultierenden Baum kann man nicht einfach balancieren, da dann die Ebenen durcheinander kommen

Mehrdimensionale Bereichssuche

- Aufteilung der Ebene entsprechend der Konstruktion des Baums

- Suche
  - Vorverarbeitung: Erstellung des 2D-Baums
  - Suchrechteck befindet sich vollständig oberhalb von A und rechts von E

Scan-line Prinzip

- typischer Anwendungsbereich: Entwurf höchstintegrierter Schaltungen
- auch als Plane-Sweep bezeichnet

- Grundprinzip
  - Ermitteln einer bestimmten Eigenschaft von Objekten; hierdurch werden die Objekte durchwandert

- Beispiel
  - Maximum einer Menge von Objekten
    - ⇨ jedes Objekt wird einmal betrachtet und getestet, ob es größer als das größte bisher betrachtete Objekt ist
    - ⇨ Problem ist räumlich eindimensional. Verfahren macht daraus eine zeitliche Folge von räumlich nulldimensionalen Objekten.
    - ⇨ hier: Reihenfolge nicht wesentlich. Im allgemeinen allerdings wesentlich für diese Verfahren.

Das dichteste Punktepaar in der Ebene

- Gesucht
  - Kleinster Abstand zweier Punkte in der Ebene: $\min_{1 \le i < j \le n} |p_i p_j|$
- Vorgehensweise
  - Bilden eines Streifens mit der Breite des bisherigen Minimums
    - ⇨ Verzeichnis mit den Punkten innerhalb dieses Streifens. Aktualisierung erforderlich, falls
      - ⇨
  - Bestimmen des Minimums bei Erreichen eines neuen Punktes
    - ⇨ Bereichssuche
    - ⇨ nur y-Koordinaten sind wichtig (balancierter Baum)

Segmentschnitte

- Plane-Sweep
  - eine senkrechte Gerade durchwandert die Ebene von links nach rechts
- Segmentschnitt-Problem
  - gegeben sei eine Menge orthogonaler Liniensegmente
- Beispiel
  - Reihenfolge C E D G I B F C H B A I E D H F
- Erstellung eines *x*-Baums
  - Beim Finden eines Anfangspunktes eines vertikalen Segments wird das Segment in den binären Suchbaum eingetragen. Beim Finden des Endpunktes wird es gelöscht.
  - Schlüssel: *x*-Koordinate
  - bei horizontalen Segmenten wird eine Intervallsuche durchgeführt
  - Linien werden anhand ihrer *y*-Koordinaten sortiert und danach in dieser Reihenfolge verarbeitet
- Aufwand
  - $O(n \ln n)$

Segmentschnitte

- Situation
  - Geradenstücke mit beliebiger Steigung
- Algorithmus
  - Sortierung nach der *y*-Koordinate
  - Ähnlich wie eben: Suchbaum; beim Antreffen des unteren Punkts einer Linie erfolgt der Eintrag in den Baum, Löschen beim Antreffen des oberen Punks
  - Schlüssel
    - ➩ allgemeine Ordnungsbeziehung

Intervallbäume

- Effiziente Datenstruktur für eine Menge von Intervallen
- Eigenschaften
  - an den Blättern stehen überschneidungsfreie Intervall-Teilstücke
  - Blattreihenfolge von links nach rechts entspricht aufsteigender Ordnung der Intervalle nach *x*-Koordinate
  - ein innerer Knoten repräsentiert das Intervall, das alle darunterliegenden überdeckt
  - an jedem Knoten werden die Namen der zughörigen Intervalle gespeichert
  - einIntervall im Baum wird ruch mehrere Pfade dargestellt

Überlappende Rechtecke

- Gegeben
  - Menge von Rechtecken
- Gesucht
  - alle Paare von Rechtecken, die sich schneiden
- Teilprobleme
  - Ist ein Rechteck komplett in einem anderen enthalten?
  - falls nicht, schneiden sich die Rechtecke?

  - Auf das erste Teilproblem läßt sich mit dem Plane-Sweep Algorithmus lösen
    - ⇨ sortiere Anfangs-/Endpunkte der Rechteck-Liniensegmente nach der *y*-Koordinate
    - ⇨ behandle alle Punkte in *y*-Reihenfolge
      - ⇨ falls der nächste Punkt ein unterer Eckenpunkt ist, dann Eintragen in Intervallbaum
      - ⇨ falls es ein oberer Punkt ist: Löschen
      - ⇨ falls es ein Suchpunkt ist, Suche nach überdeckendem Intervall

Kontur einer Rechteckmenge

- ❑ Gegeben
  - Menge von Rechtecken
- ❑ Gesucht
  - Kontur aller überdeckten Flächen
- ❑ Algorithmus
  - Berechnung der Streifenmenge
    - ➩ horizontale Streifen, deren Dicke durch die Rechteck-Überlagerungen bestimmt ist
  - Bestimme für jeden Streifen die bedeckten Bereiche, dargestellt als Menge von disjunkten x-Intervallen
    - ➩ Plane-Sweep für jeden Streifen
  - Bestimmung der Konturstücke

Finito

Viel Erfolg
bei der Vordiplomsklausur !!!