Karlsruher Institut für Technologie
Institut für Theoretische Informatik

Prof. Dr. Peter Sanders
Dr. Thomas Worsch, Dr. Simon Gog
Demian Hespe, Yaroslav Akhremstev

4. Übungsblatt zu Algorithmen II im WS 2017/2018

http://algo2.iti.kit.edu/AlgorithmenII_WS17.php
{hespe,sanders,simon.gog,worsch,yaroslav.akhremtsev}@kit.edu

Aufgabe 1 (*Analyse: ADAC Mitgliedschaft*)

Der "**A**utomobil **D**urch **A**lgorithmiker **C**lub" (ADAC) leistet auf Autobahnen Pannenhilfe. Ein Autofahrer hat in seiner Zeit als Verkehrsteilnehmer n Pannen, $n \in \mathbb{N}_{\geq 0}$, für die er die Hilfe des ADAC in Anspruch nehmen muss. Für jede geleistete Pannenhilfe verlangt der Club eine Aufwandsentschädigung abhängig von der Schwere der Panne. Mitglieder beim ADAC müssen lediglich ein Viertel dieser Kosten bezahlen. Eine lebenslange Mitgliedschaft kann man sich durch eine Einmalzahlung in Höhe von 1000 DM (**D**ijkstra **M**ark) sichern.
Da nicht schon mit Erwerb des Führerscheins klar ist, wie viele Pannen man in seinem Leben haben wird und wie schwerwiegend diese sein werden, stellt sich die Frage, ab wann es sich lohnt eine Mitgliedschaft beim ADAC zu erwerben.

a) Geben Sie eine Strategie an, die einen kompetitiven Faktor (competitive ratio) von ∞ erreicht. Begründen Sie kurz.

b) Wie gut ist die Strategie, sich nie eine Mitgliedschaft beim ADAC zu sichern? Begründen Sie.

c) Zeigen Sie, dass folgende Strategie einen kompetitiven Faktor $c = 3$ hat. Die Strategie ist, sich beim Pannenhelfer eine Mitgliedschaft zu kaufen, wenn die momentan von ihm bearbeitete Panne die Gesamtausgaben für Pannen (ohne Mitgliedschaft) auf über 500 DM anheben würde.

Hinweis: Verwenden Sie die summierten Gesamtkosten K über alle Pannen (ohne Mitgliederrabatt).

Aufgabe 2 (*Analyse: Online-gaming-Algorithmen*)

Angestellte in einem Rechenzentrum haben einen recht eintönigen Job. Ihnen stehen zwar die größten Rechner zur Verfügung, Sie dürfen diese aber nicht selbst verwenden. Stattdessen heisst es nur, die Maschinen möglichst gut auszulasten und Rechnungen zu schreiben. Ein ziemlich langweiliger Job möchte man meinen – zum Glück gibt es noch Computerspiele.
Ein Mitarbeiter hat zu Weihnachten ein neues Computerspiel erhalten, dass er liebsten andauernd spielen würde. Diesem Wunsch steht leider seine Arbeit im Wege. Eine neue Dienstanweisung besagt, dass der teure Großrechner zu mindestens 50% ausgelastet sein muss. Da das Scheduling in diesem Rechenzentren noch von Hand durchgeführt wird, muss der spielfreudige Mitarbeiter die laufenden Jobs überwachen und schnell neue Jobs auf leerlaufende Maschinen verteilen. So bleibt leider nur wenig Zeit zum Spielen am Arbeitsplatz.
Jeder Job hat eine Mindestlaufzeit von 5 Minuten. Die Zeit zum Verteilen der Jobs kann für die Bestimmung der Auslastung vernachlässigt werden. Der Mitarbeiter kann in dieser Zeit aber nicht spielen. Ein Job hat während seiner Ausführung die Maschine exklusiv. Es gibt keine automatischen Benachrichtigungen über das Ende eines Jobs. Der Mitarbeiter muss dies selbst periodisch überprüfen.

a) Entwerfen Sie einen *online scheduling* Algorithmus, der die Anzahl an Spielunterbrechungen des Mitarbeiters minimiert.

b) Zeigen Sie, dass Ihr Algorithmus optimal bzgl. der Anzahl an Unterbrechungen ist.

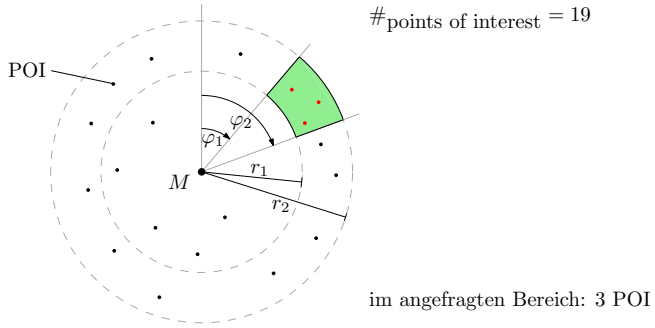
Aufgabe 3 (*Kleinaufgaben: Geometrie-Entwurf*)

Entwerfen Sie einen Algorithmus, der ...

a) in Zeit $O(n \log n)$ ein geschlossenes, kreuzungsfreies Polygon aus n Punkten $P \in \mathbb{R}^2$ konstruiert.

b) in Zeit $O(n)$ für eine Menge von n Filialen einen Standort für ein Zentrallager berechnet, so dass der maximale Abstand (in Luftlinie) zwischen Zentrallager und allen Filialen minimiert wird.

c) (*) in Zeit $O(\log n)$ die Anzahl an *points of interest* (POI) um einen fixen Mittelpunkt M in einem Winkelbereich $[\varphi_1, \varphi_2]$ und einem Entfernungsbereich $[r_1, r_2]$ bestimmt.



Bei n POI ist eine Vorverabeitungszeit von $O(n \log n)$ und ein Platzverbrauch von $O(n)$ erlaubt.

Aufgabe 4 (*Analyse+Entwurf: Überdeckungsproblem*)

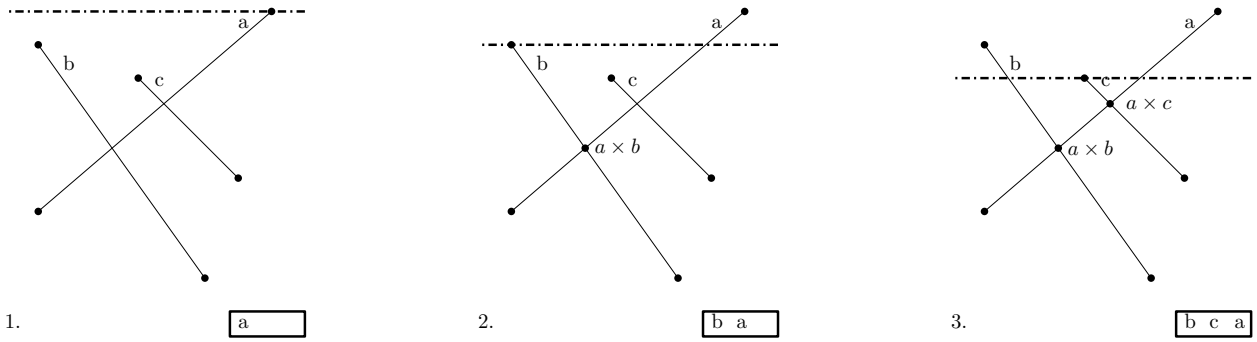
Zur Gebietsüberwachung wurde in einem weitläufigen Gelände ein Sensornetz aus mehreren Millionen Knoten ausgelegt. Die Positionsdaten der Knoten wurden per Funkübertragung an einer zentralen Stelle gesammelt. Jeder Sensorknoten überwacht ein kreisförmiges Gebiet mit Radius r. Durch Fehler in der Ausbringung können dabei starke Überlappungen der von den Knoten überwachten Gebiete entstanden sein.
Um diese Information zu einem späteren Zeitpunkt ggf. nutzen zu können, haben die Betreiber beschlossen, dass alle Knotenpaare bestimmt werden sollen, deren Gebiete sich teilweise überlappen.

a) Zeigen Sie, dass ein *Sweepline*-Algorithmus, der einfach alle (im Rahmen der Algorithmenausführung) aktiven Sensorknoten auf Überlappung prüft, in quadratischer Laufzeit resultieren kann, auch wenn die Ausgabekomplexität (Anzahl überlappender Knotenpaare) linear ist.

b) Überlegen Sie, wie Sie dennoch einen *Sweepline*-Algorithmus verwenden können, um das Problem in Linearzeit zu lösen, falls die Ausgabekomplexität linear ist.

Aufgabe 5 (*Entwurf: Platzeffizienter Linienschnitt*)

In der Vorlesung wurde ein *Sweepline*-Algorithmus zur Bestimmung der Schnitte zwischen n Liniensegmenten behandelt. Der Algorithmus arbeitet eine Liste an Ereignispunkten (Schnittpunkte, Linienanfänge und Linienenden), in Form einer nach der y-Position sortierten *Queue*, ab. Gleichzeitig führt er eine Liste aktiver Kanten in sortierter Reihenfolge.
Das betrachtete Problem lässt sich in seiner Laufzeitkomplexität nie besser lösen als durch die Anzahl Schnittpunkte vorgegeben. Liegen z.B. $O(n^2)$ Schnittpunkte vor, so kann bestenfalls eine quadratische Laufzeitkomplexität erreicht werden.
Für den Algorithmus aus der Vorlesung gilt die gleiche Einschränkung auch für den Platzbedarf. Die *Queue* muss bis zu $O(n + k)$ Ereignispunkte gleichzeitig halten, bei insgesamt k Linienschnitten. Dies liegt unter anderem daran, dass einmal erkannte Schnittpunkte auch zwischen Liniensegmenten existieren können, die in der sortierten Liste nicht (mehr) benachbart sind. Dies sei durch folgendes Beispiel illustriert:



Im Beispiel wird zuerst der Schnittpunkt $a \times b$ zwischen den Linien a und b bestimmt. Nach der Aktivierung von Linie c ist der Schnittpunkt weiterhin in der *Queue*, die Linien a und b sind allerdings nicht mehr benachbart.

Modifizieren Sie den Algorithmus so, dass er nur noch $O(n)$ Platz für die Ereignispunkte benötigt.

Aufgabe 6 (*Analyse+Entwurf:* Graham's Scan)

Betrachten Sie den *Graham's Scan* Algorithmus zur Bestimmung der konvexen Hülle einer Punktemenge $P \in \mathbb{R}^2$ mit $|P| = n$. In der Vorlesung wurde eine lexikographische Sortierung der Punkte vorgeschlagen, mit der Folge dass die obere und untere Hülle getrennt berechnet werden mussten.

a) Geben Sie eine geeignetere Sortierung der Punkte an, so dass die gesamte konvexe Hülle in einem Durchlauf berechnet werden kann.

b) Zeigen Sie, dass der *Graham's Scan* Algorithmus nicht für jede beliebige Sortierung der Punkte eine korrekte konvexe Hülle liefert (Randfälle ausgenommen).

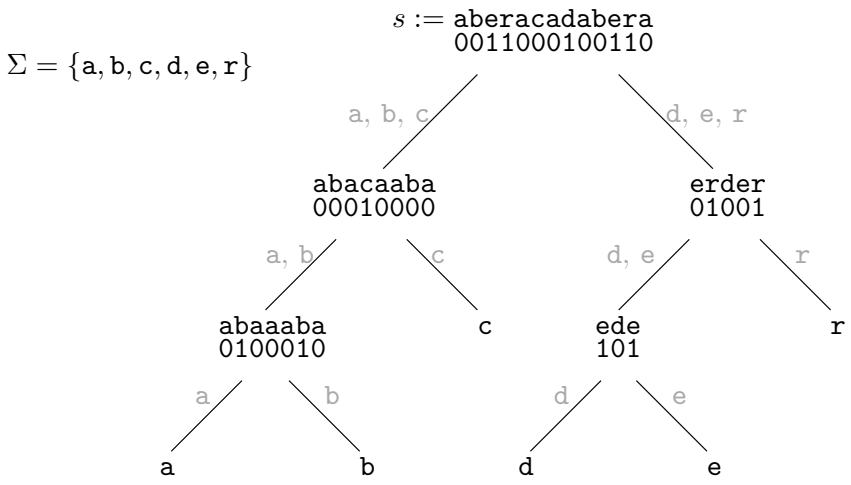
c) Zeigen Sie anhand eines Beispiels, dass es möglich ist, dass der *Graham's Scan* Algorithmus p schon zur Hülle hinzugefügte Punkte hintereinander verwirft für beliebig großes p.

d) Eine Schneidemaschine bringt Stoffe anhand eines Schnittmusters in die gewünschte Form. Ein Schnittmuster ist dabei durch ein Polygon aus n Ecken definiert. Die Schneidemaschine kann beliebige konvexe Stoffe bearbeiten.

   Entwerfen Sie einen Algorithmus, der den minimal möglichen Verschnitt für ein gegebenes Stoffmuster in Linearzeit berechnet. Sie können davon ausgehen, dass die Ecken des Polygons als geschlossener Kantenzug vorliegen.

Aufgabe 7 (*Entwurf:* Wavelet Trees *für Zeichenketten (*)*)

Aus der Vorlesung ist Ihnen eine schnelle `rank(i)` Operation auf Bitfolgen bekannt. Diese Operation berechnet die Anzahl Einsen in der Bitfolge bis zu Position i. Dies ist in konstanter Zeit möglich. Für schnelle Suchalgorithmen auf Zeichenketten benötigt man eine Erweiterung dieser Operation, `rank(c,i)`, die die Anzahl an Zeichen `c` bis Position i liefert.
Für diese (und weitere) Aufgaben sind *Wavelet Trees* für Zeichenketten ein sehr nützliches Werkzeug. Im folgenden soll diese Datenstruktur kurz vorgestellt werden: Für eine Zeichenkette s über dem Alphabet Σ wird ein *Wavelet Tree* wie in folgendem Bild aufgebaut.



Die Blätter des Baumes halten die verwendeten Zeichen des Alphabets, die inneren Knoten halten Bitvektoren, die die Menge der Zeichen partitionieren. Eine 0 bzw. 1 gibt an, ob das Zeichen an dieser Stelle zur unteren (aufgerundet) oder zur oberen (abgerundet) Hälfte des in diesem Knoten aktiven Teil des Alphabets gehört. In der Wurzel ist das gesamte Alphabet aktiv. Dieses wird in jedem Nachfolger halbiert (die Kantenbeschriftungen geben das aktive Alphabet an). Die angegebenen Zeichenketten über den Bitvektoren dienen nur der Anschauung, im *Wavelet Tree* werden sie nicht gespeichert.

Gehen Sie davon aus, dass die Bitvektoren eine $\texttt{rank}_{0/1}\texttt{(i)}$ Operation zur Bestimmung der Nullen bzw. Einsen bis Position i in $O(1)$ besitzen. Außerdem können Sie annehmen, dass *Wavelet Trees* balanciert sind. Entwerfen Sie unter diesen Voraussetzungen einen Algorithmus, der ...

a) `access(s,i)` berechnet, d.h. der das i-te Zeichen der Zeichenkette s zurückliefert.
(Bsp.: `access(s,4) = 'r'`)

b) `rank(s,c,i)` berechnet, d.h. der die Anzahl an Zeichen c in s bis Position i angibt.
(Bsp.: `rank(s,'a',7) = 3`)

Beide Algorithmen dürfen $O(\log u)$ Zeit benötigen, mit $u = \min(|s|, |\Sigma|)$.

Ausgabe: 05.12.2017
Abgabe: keine Abgabe, keine Korrektur