Name:

Vorname:

Matrikelnummer:

Lösungsvorschlag

Karlsruher Institut für Technologie
Institut für Theoretische Informatik

Prof. Dr. P. Sanders ID 12.10.2012

Klausur Algorithmen II

Aufgabe 1.	Kleinaufgaben	16 Punkte
Aufgabe 2.	Fortgeschrittene Datenstrukturen: Fibonacci Heaps	11 Punkte
Aufgabe 3.	Starke Zusammenhangskomponenten	10 Punkte
Aufgabe 4.	Vertex Cover: Rechnerraum	7 Punkte
Aufgabe 5.	Externe Algorithmen: Summer School II	9 Punkte
Aufgabe 6.	Online Algorithmen: Kuhweg	7 Punkte

Bitte beachten Sie:
– Als Hilfsmittel ist nur **ein** DIN-A4 Blatt mit Ihren **handschriftlichen** Notizen zugelassen.
– **Schreiben** Sie auf **alle** Blätter Ihren Namen und Ihre Matrikelnummer.
– Merken Sie sich Ihre Klausur-ID für den Notenaushang.
– Die Klausur enthält **19 Blätter**.
– Zum Bestehen der Klausur sind 20 Punkte hinreichend.

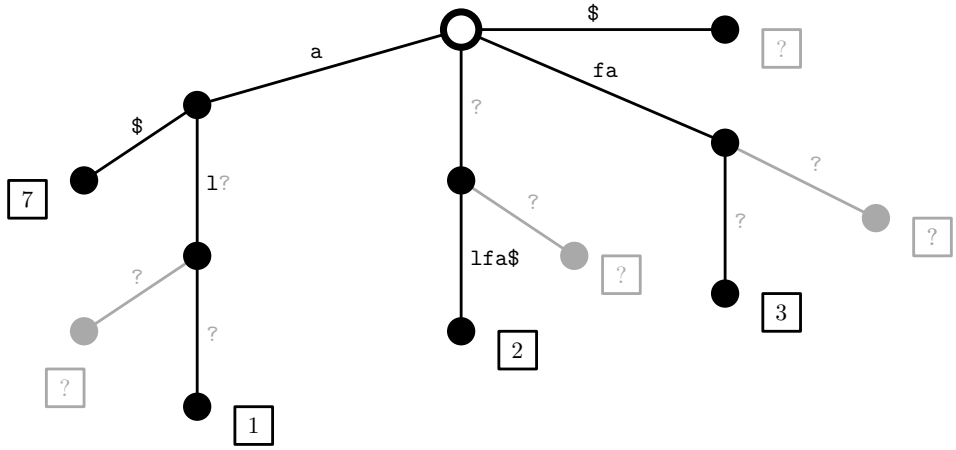
Lösungsvorschlag

Aufgabe 1. Kleinaufgaben [16 Punkte]

a. Durch einen Festplattencrash sind viele Daten verlorengegangen. Darunter befinden sich auch Teile eines Suffixbaumes, der ein wichtiges Codewort repräsentiert. Die Überreste des Baumes sind unten abgebildet. Graue Äste und Fragezeichen repräsentieren verlorengegangene Daten *unbekannter* Größe. Rekonstruieren Sie das gespeicherte Codewort.

Variablen: ***?****: verlorene Information unbestimmter Größe*

Falls Sie beide Kästchen benutzen, markieren Sie bitte, welches gewertet werden soll.



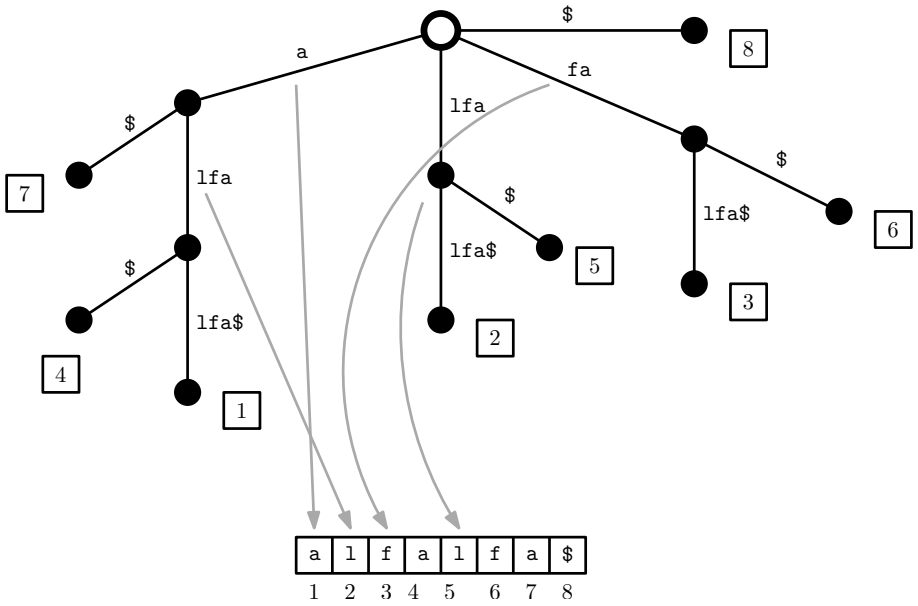
							$
1	2	3	4	5	6	7	8

							$
1	2	3	4	5	6	7	8

[3 Punkte]

Lösung

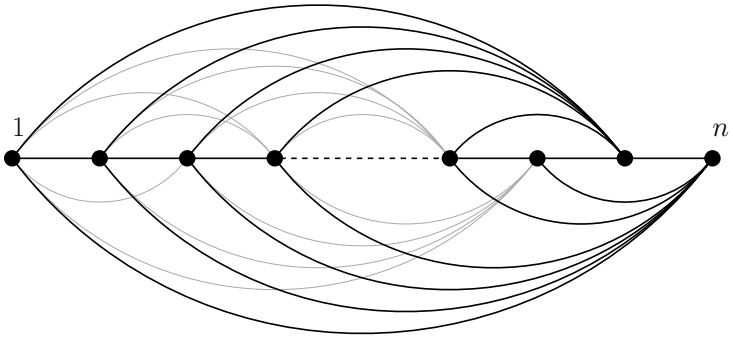
Das Bild zeigt den vollständigen Suffixbaum sowie die Stellen aus denen die Informationen extrahiert werden konnten.



b. Geben Sie eine Klasse an gerichteten, gewichteten Graphen mit n Knoten und $\Theta(n^2)$ Kanten an, auf denen Dijkstras Algorithmus –beginnend bei einem festen Knoten x– für jede Kante ein `insert` oder `decreaseKey` auslöst (2 Punkte). Markieren Sie x und begründen Sie die Korrektheit Ihrer Lösung (1 Punkt).

Variablen: ***n****: Anzahl Knoten,* ***x****: ein fester Knoten* [3 Punkte]

Lösung



Eine mögliche Lösung ist ein Graph, bei dem jeder Knoten i eine Kante zu den Knoten $1 \ldots i-1$ besitzt (siehe Bild). Dieser Graph hat insgesamt $\sum_{i=1}^{n} i-1 = n \cdot (n-1)/2 - n = \Theta(n^2)$ Kanten. Die Kantengewichte sind wie folgt festgelegt:

$$c((u,v)) = \begin{cases} 1 & v = u-1 \\ 2*u & v \neq u-1 \end{cases}$$

Sucht man von Knoten n ausgehend, so bekommt zunächst jeder Knoten Gewicht $2n$. Nur Knoten $n-1$ erhält Gewicht 1. Induktiv lässt sich erkennen, dass weitere Knoten Gewicht $2(n-1)+1 = 2n-1$ erhalten, nur Knoten $n-2$ Gewicht 2. Somit bewirkt nach den ersten n `insert` jede Kante ein `decreaseKey`.

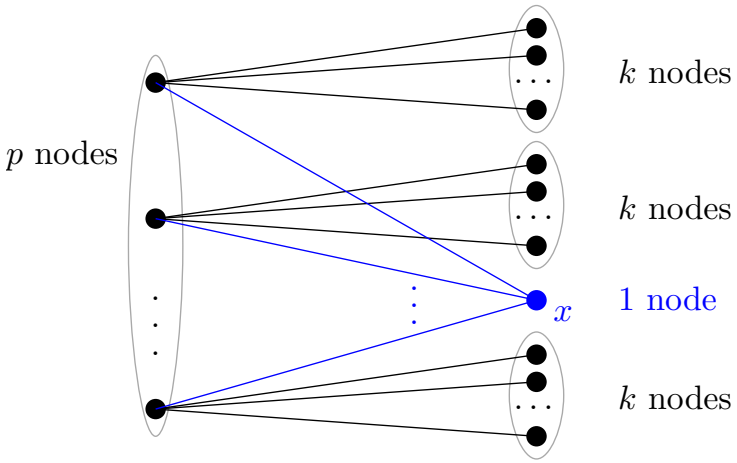
c. Folgender Algorithmus bestimmt ein *Dominating Set* auf einem Graph $G = (V, E)$:

1. Wähle Knoten v mit maximalem Grad.
2. Füge v zum *Dominating Set* hinzu.
3. Entferne v und alle seine Nachbarn aus der Menge der zu betrachtenden Knoten.
4. Wiederhole solange noch Knoten zu betrachten sind.

Widerlegen Sie die folgende Aussage durch ein Gegenbeispiel: Der angegebene Algorithmus berechnet eine 2-Approximation für das *Minimum Dominating Set* Problem. [2 Punkte]

Lösung

Der angegebene Algorithmus berechnet keine 2-Approximation. Gegenbeispiel:



Für $p > k+1$ wird zuerst Knoten x in das *Dominating Set* aufgenommen und zusammen mit allen Knoten auf der linken Seite entfernt. Anschließend sind nur noch die Knoten auf der rechten Seite vorhanden, die alle in das *Dominating Set* aufgenommen werden. Insgesamt hat das berechnete *Dominating* $p \cdot k + 1$ Elemente. Optimal wären p. Es ergibt sich ein Approximationsfaktor von $\frac{p \cdot k+1}{p} = k + \frac{1}{p}$. Für $k \geq 2$ ist dies keine 2-Approximation.

d. Ein Mobilfunkmast (m_i) kann alle Gebäude in einem kreisförmigen Gebiet (Radius r_i) erreichen. Die dafür benötigte Sendeleistung wächst mit der Größe des Gebiets. Momentan werden zwei Mobilfunkmasten eingesetzt, um alle Gebäude am Campus zu erreichen. Um Wartungskosten zu sparen, sollen die beiden Masten durch einen einzelnen ersetzt werden.
Entwerfen Sie einen Algorithmus, der eine optimale Position für den neuen Mobilfunkmasten bestimmt. Es müssen weiterhin alle Gebäude am Campus erreicht werden. Außerdem muss die dafür benötigte Sendeleistung minimal sein. Ihr Algorithmus soll erwartet linear in der Anzahl Gebäude ausgeführt werden können. Begründen Sie die Laufzeit Ihres Algorithmus. Gebäude können als punktförmig angenommen werden. Sie liegen in Form einer Koordinatenliste der Länge n vor.

Variablen: $\mathbf{m_i}$*: Sendemast,* $i \in \{0,1\}$, $\mathbf{r_i}$*: Senderadius von Mast* m_i, $\boldsymbol{n}$*: Anzahl Gebäude* [2 Punkte]

Lösung

Auf dem Campus befinden sich n Gebäude. Zur Lösung bestimmt man den *smallest enclosing ball* für diese Gebäude in $O(n)$. Der Mittelpunkt ist die optimale Position für den neuen Mobilfunkmasten.

e. Gegeben seien die Laufzeiten von drei Algorithmen. Welche davon sind *fixed parameter tractable* bezüglich k, welche bezüglich l?

1. $f(n,k,l) = n^{3l} + 2k^2l^3$
2. $g(n,k,l) = n^{2\cdot\sqrt{k+l}}$
3. $h(n,k,l) = n^2 \cdot \sqrt{k+l}$

Variablen: ***n**: Eingabegröße,* ***k**: Laufzeitparameter,* ***l**: Laufzeitparameter* [3 Punkte]

Lösung

Ein Algorithmus ist *fixed parameter tractable* bzgl. einer Variable k, wenn seine Laufzeit in $O(p(n)f(k))$ liegt. Dabei ist $p(n)$ ein Polynom, das nicht von k abhängt und $f(k)$ eine berechenbare Funktion, die nicht von n abhängt.

Damit ergibt sich:

1. $f(\cdot)$: Bzgl. k *fixed parameter tractable*.
2. $g(\cdot)$: Weder bzgl. k noch l *fixed parameter tractable*.
3. $h(\cdot)$: Sowohl bzgl. k als auch l *fixed parameter tractable*.

f. Gegeben sei ein Cache, der 4 Hauptspeicherseiten halten kann, sowie eine Folge an Seitenzugriffen σ. Nachfolgend ist die Belegung des Caches nach jedem Seitenzugriff abgebildet. Die Platzierung der Seiten im Cache ist dabei unerheblich. Welches Ihnen bekannte *Online Paging* Verfahren erzeugt diese Abfolge (1 Punkt)? Nennen Sie ein Verfahren, das weniger Seitenfehler erzeugt und wenden Sie es an (2 Punkte).

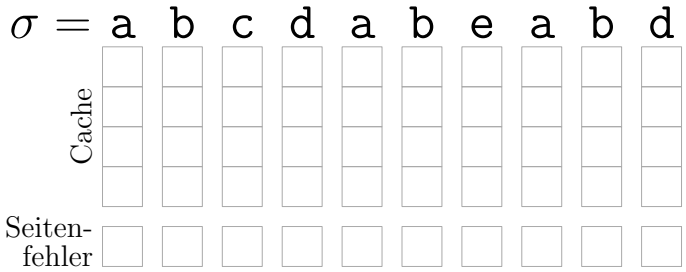
$\sigma =$	a	b	c	d	a	b	e	a	b	d
Cache	a	a	a	a	a	a	e	e	e	e
		b	b	b	b	b	b	a	a	a
			c	c	c	c	c	c	b	b
				d	d	d	d	d	d	d
Seiten-fehler	✓	✓	✓	✓	—	—	✓	✓	✓	—

Legende:
— kein Seitenfehler
✓ Seitenfehler

Variablen: ***a, b, c, d, e**: Speicherseiten*

Tragen Sie Ihre Lösung in eine der Vorlagen ein:

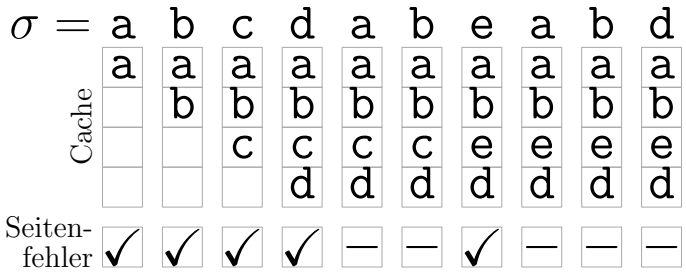
$\sigma =$	a	b	c	d	a	b	e	a	b	d
Cache										
Seiten-fehler										



Falls Sie beide Vorlagen verwenden, markieren Sie bitte, welche zu werten ist. [3 Punkte]

Lösung

Es handelt sich um das *FIFO Verfahren*. Das *LRU Verfahren* erzeugt für diese Eingabefolge weniger Seitenfehler.



Lösungsvorschlag

Aufgabe 2. Fortgeschrittene Datenstrukturen: Fibonacci Heaps [11 Punkte]

a. Geben Sie eine Klasse an Zuständen eines *Fibonacci Heaps* an, für die eine `decreaseKey` Operation auf einem ausgewählten Knoten x eine Laufzeit von $\Theta(\log n)$ besitzt; geben Sie den Knoten x explizit an (2 Punkte). Beschreiben Sie, wie der von Ihnen ausgewählte Zustand erzeugt werden (2 Punkte).

Variablen: $\mathbf{B_k}$*: Binomialbaum,* $\boldsymbol{k}$*: Höhe des Baums,* $\boldsymbol{n}$*: Anzahl Knoten im Baum* [4 Punkte]

Lösung

Eine `decreaseKey` Operation auf einem Blattelement x in maximaler Tiefe bedingt $\Theta(\log n)$ kaskadierende Schnitte, wenn alle seine Vorgänger bis zur Wurzel markiert sind und nur ein Baum existiert.
Ein solcher Baum kann wie folgt erzeugt werden: Generiere einen Binomialbaum $B_{\log n}$ durch $n+1$ Einfügeoperationen und ein Entfernen. Anschließend lösche alle Blätter des orginalen Binomialbaums mit Ausnahme des Blatts mit maximaler Tiefe. Dies ist Blattelement x.

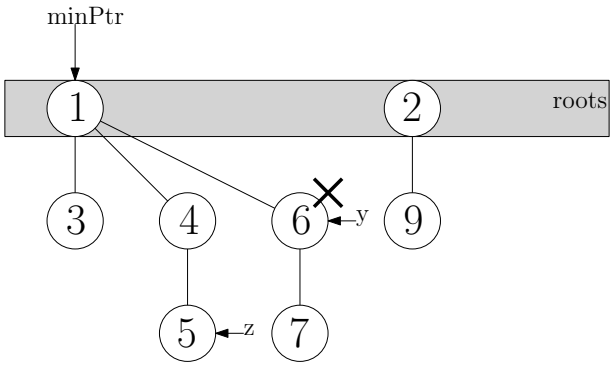
b. Gegeben sei ein Fibonacci Heap. Der Heap habe den unten eingezeichneten Zustand. Kreuze symbolisieren markierte Elemente, y und z sind *Handles* auf die markierten Elemente.

Geben Sie zunächst eine Folge von Operationen an, die diesen Zustand erzeugt (3 Punkte). Führen Sie anschließend folgende Operationen auf dem Heap aus und zeichnen Sie den Zustand des Heaps nach jeder Operation in die Vorlagen auf dem nächsten Blatt ein (4 Punkte):

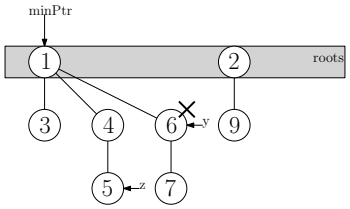
- `decreaseKey(y,5)`
- `remove(z)`
- `deleteMin()`

Hinweis: Vergessen Sie nicht, die Knoten korrekt zu markieren.

Gegebener Zustand des *Fibonacci Heap*:



Falls Sie mehrere Vorlagen beschriften, markieren Sie bitte, welche zu werten ist.



`decreaseKey(y,5)`:

roots

roots

`remove(z)`:

roots

roots

`deleteMin()`:

roots

roots

[7 Punkte]

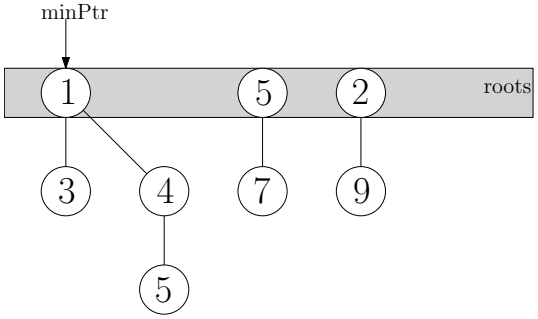
Lösung

Die folgenden Operationen erzeugen den gegebenen Zustand:

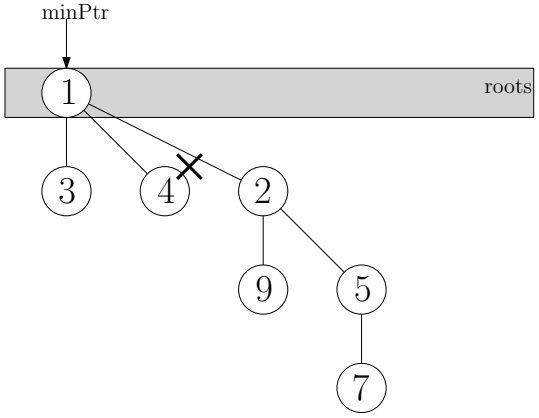
```
insert(0),insert(1),insert(3),insert(4),z:=insert(5),
y:=insert(6),insert(7),b:=insert(8),insert(9),deleteMin(),
decreaseKey(b,2)
```

Beachten Sie, dass die Definition des *Fibonacci Heap* keine Aussage darüber macht, in welcher Reihenfolge Wurzeln mit gleichem Rang vereinigt werden. Für diese Lösung und die der nächsten Teilaufgabe, nehmen wir an, es wird in Reihenfolge der Einfügungen vereinigt. Andere Annahmen sind aber auch zulässig.

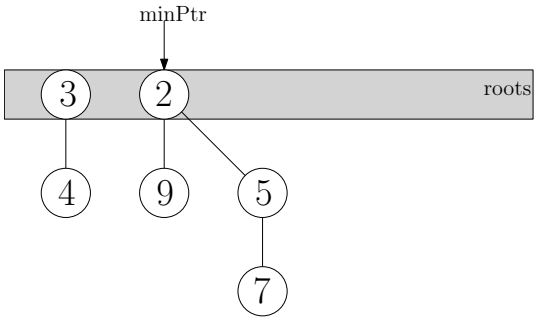
1. `decreaseKey(y,5)`:



2. `remove(z)`:



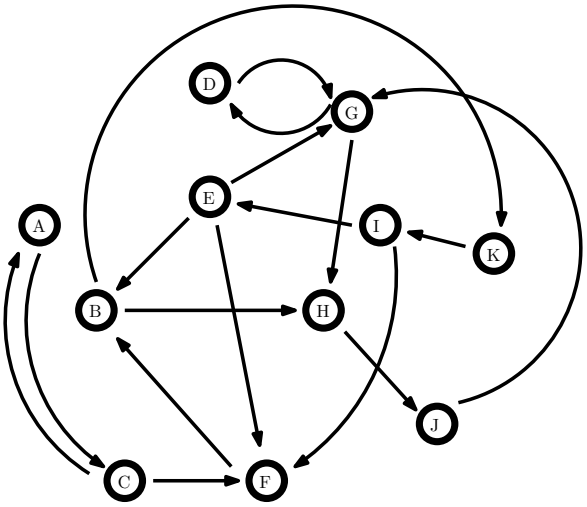
3. `deleteMin()`:



Lösungsvorschlag

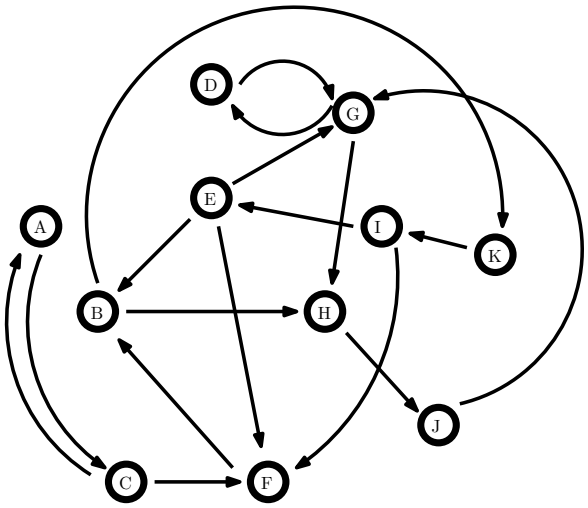
Aufgabe 3. Starke Zusammenhangskomponenten [10 Punkte]

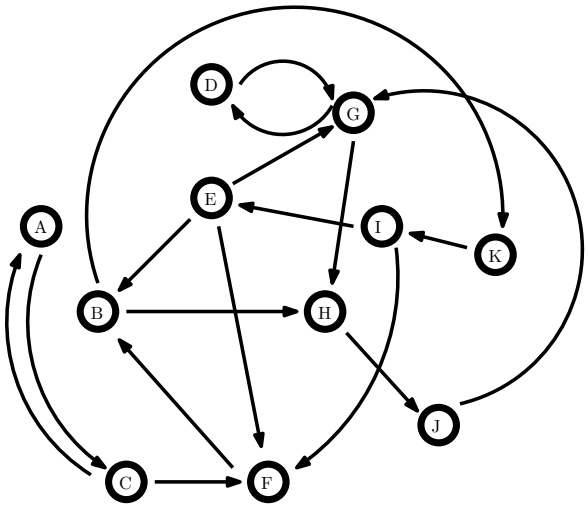
Gegeben sei folgender Graph:



a. Zeichnen sie **alle** *starken Zusammenhangskomponenten* in den Graphen ein.

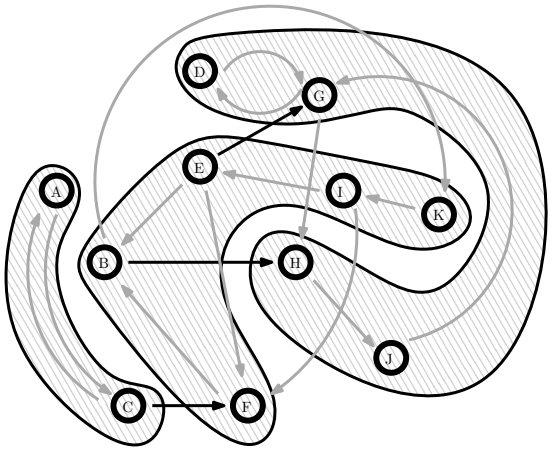
Falls Sie mehrere Graphen beschriften, markieren Sie bitte, welcher gewertet werden soll.





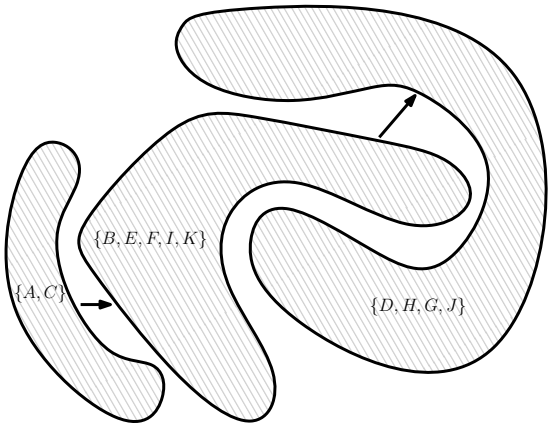
[2 Punkte]

Lösung



b. Geben Sie den Schrumpfgraphen des gegebenen Graphen an. [1 Punkt]

Lösung



c. Beweisen Sie die Aussage: Der Schrumpfgraph eines Graphen ist azyklisch. [4 Punkte]

Lösung

Beweis durch Widerspruch:
Annahme der Schrumpfgraph enthält einen Zyklus $n_1,\ n_2,\ \ldots\ n_n = n_1$. Per Definition repräsentiert jeder Knoten des Schrumpfgraphen eine maximal große SCC im Originalgraph. Da die Knoten nach Annahme einen Zyklus bilden, existieren in aufeinanderfolgenden Komponenten n_k und n_{k+1} Knoten $w_{o,k}$ bzw. $w_{i,k+1}$, die durch eine Kante verbunden sind. Ebenso existiert eine Verbindung innerhalb jeder Komponente von $w_{i,k}$ zu $w_{o,k}$ für alle k. Sei nun u ein Knoten aus der Komponenten n_i, und v ein Knoten aus der Komponente n_j (obda $i < j$). Somit existiert insgesamt ein Weg von u über $w_{o,i}$ und $w_{i,j}$ zu v. Nach dem selben Argument existiert ein solcher Weg von v zu u. Somit liegen $u,\ v$ in einer SCC, und alle Komponenten des Zyklus bilden eine einzige SCC im Originalgraphen. Dies ist ein Widerspruch dazu, dass die Knoten des Schrumpfgraphen jeweils maximal große SCCs aus dem Originalgraphen repräsentieren.

d. Gegeben sei ein gerichteter, nicht notwendigerweise zusammenhängender Graph $G = (V, E)$. Entwerfen Sie einen Algorithmus, der in linearer Zeit alle Knoten ausgibt, von denen man alle Knoten des Graphen erreichen kann (2 Punkte). Begründen Sie Ihre Laufzeit (1 Punkt).

Variablen: $\mathbf{n} = |\mathbf{V}|$*: Anzahl Knoten,* $\mathbf{m} = |\mathbf{E}|$*: Anzahl Kanten* [3 Punkte]

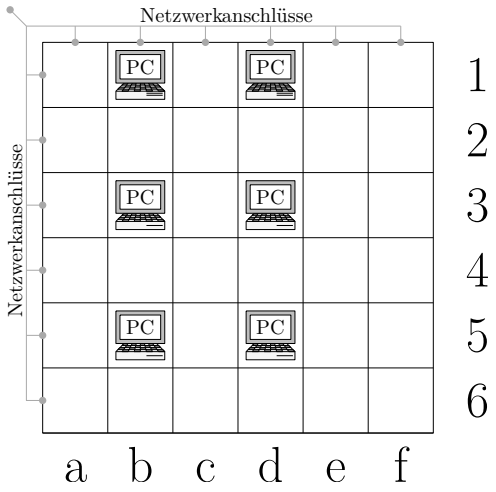
Lösung

Bestimme alle starken Zusammenhangskomponenten (SCCs) sowie den Schrumpfgraph G^S von G in $O(n+m)$. Enthält der Schrumpfgraph nur einen Knoten K^S ohne eingehende Kanten, so erreichen alle Knoten in G, die durch K^S repräsentiert werden, alle anderen Knoten von G.

Lösungsvorschlag

Aufgabe 4. Vertex Cover: Rechnerraum [7 Punkte]

Sie wurden mit dem Bau eines Verbindungsnetzes für ein Großraumbüro beauftragt. Die einzelnen Parzellen, in denen m Computer verteilt sind, bilden zusammen eine $w \times h$ Raummatrix (siehe Bild). Für die Verkabelung stehen Ihnen Kabelkanäle zur Verfügung. Diese können entlang von Zeilen oder Spalten der Raummatrix verlegt werden und erstrecken sich über die komplette Breite bzw. Länge des Büros. Um Kosten zu sparen, sind Sie an der minimalen Anzahl an Kanälen interessiert, die Sie benötigen, um alle Computer zu vernetzen (an allen Wänden stehen Netzwerkanschlüsse zur Verfügung).



Ihr Vorgesetzter modelliert das Problem wie folgt: Jede Zeile und jede Spalte der Matrix wird als je ein Knoten betrachtet. Diese $n = w + h$ Knoten bilden die Knotenmenge V eines Graphen. Zwei Knoten i, j werden durch eine Kante verbunden, wenn sich an Position i (Zeile), j (Spalte) der Matrix ein Computer befindet (die Kantenanzahl $|E|$ entspricht der Anzahl Computer m).

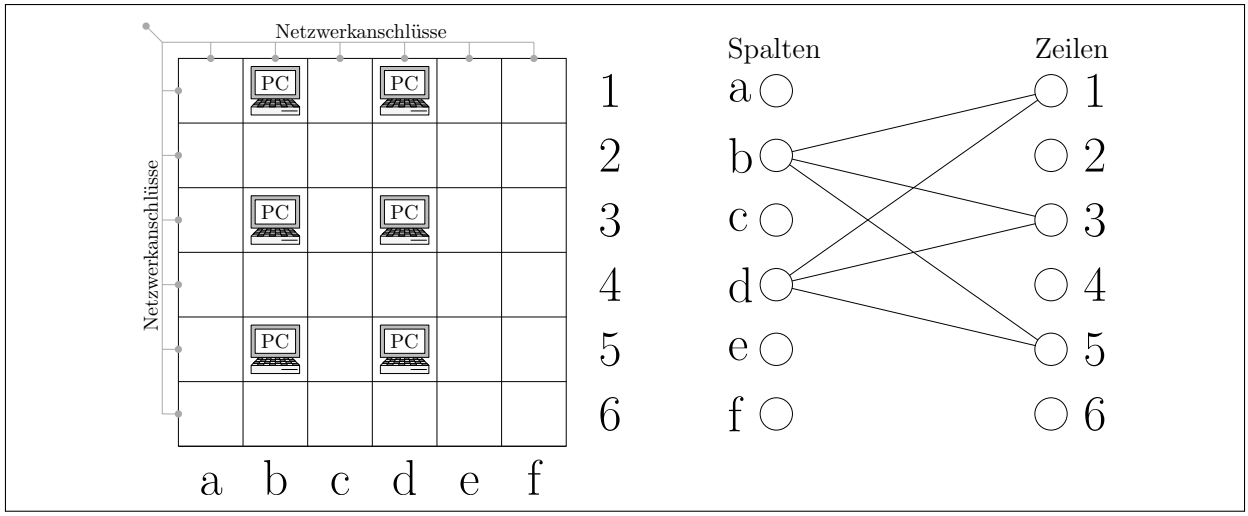
Variablen: ***n = w+h****: Anzahl Knoten,* ***m****: Anzahl Kanten (Computer),* ***i / j****: Zeilen-/Spaltenindex*

a. Zeichnen Sie den so modellierten Graphen, der dem angegebenen Rechnerraum entspricht. [1 Punkt]

Falls Sie mehrere Graphen beschriften, markieren Sie bitte, welcher gewertet werden soll.

Spalten	Zeilen	Spalten	Zeilen
a ○	○ 1	a ○	○ 1
b ○	○ 2	b ○	○ 2
c ○	○ 3	c ○	○ 3
d ○	○ 4	d ○	○ 4
e ○	○ 5	e ○	○ 5
f ○	○ 6	f ○	○ 6

Lösung

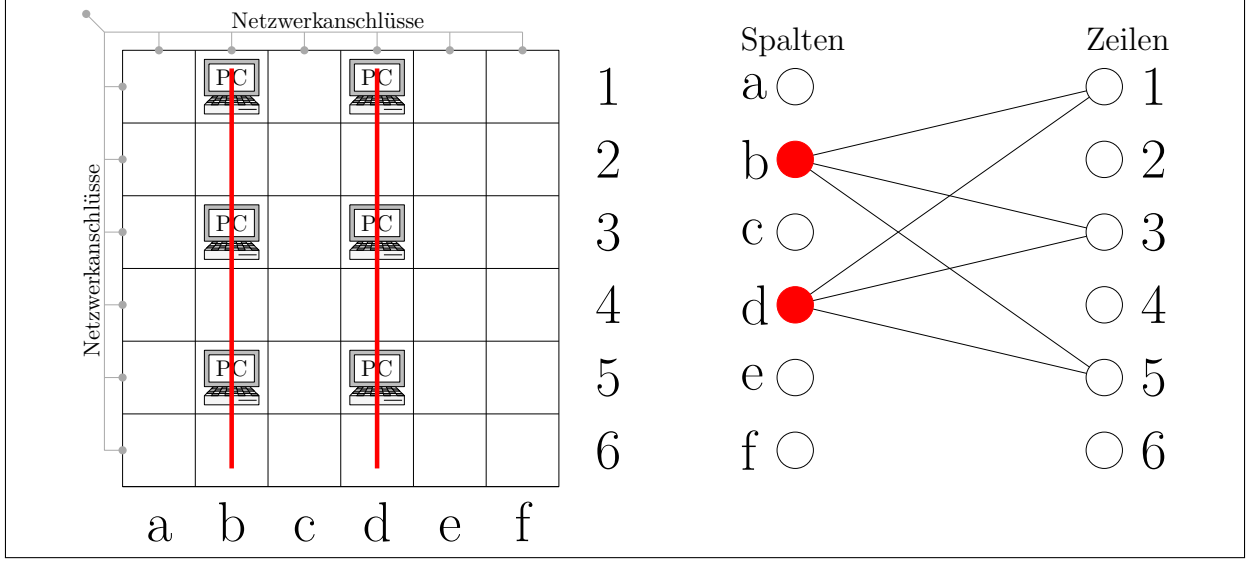


b. Begründen Sie, dass ein *Minimum Vertex Cover* auf einem so modellierten Graphen der Anzahl benötigter Kabelkanäle entspricht. [2 Punkte]

Lösung

Jeder Knoten entspricht einer Zeile oder Spalte im Gitter, also einem Kabelkanal. Jede Kante entspricht einem Computer. Per Definition ist das *Minimum Vertex Cover* die kleinste Menge an Knoten (=Kabelkanäle), so dass alle Kanten (=Computer) zu mindestens einem dieser Knoten inzident sind (=über den Kabelkanal angeschlossen sind).

Minimum Vertex Cover und Kabelkanäle im Beispiel:



c. Entwerfen Sie einen Algorithmus, der ein solches *Vertex Cover* in *maximal* $O(n^4 + m^4)$ Zeit berechnet (3 Punkte). Begründen Sie die Laufzeit (1 Punkt).

Ein Algorithmus mit größerer polynomieller Laufzeit erhält maximal 3 Punkte (2+1 Punkte), ein Algorithmus mit exponentieller Laufzeit maximal 2 Punkte (1+1 Punkte).

Hinweis: Die angegebene Laufzeitschranke ist eine sehr grobe Abschätzung. [4 Punkte]

Lösung

Nach *Königs Theorem* (siehe Übung) besteht in jedem bipartiten Graph eine Äquivalenz zwischen *Matchings maximaler Kardinalität* und *minimalen Vertex Covern*. Der Graph ist offensichtlich bipartit: Kanten laufen nur zwischen den Knoten die Zeilen repräsentieren und Knoten die Spalten repräsentieren. Ein *maximales Matching* kann über einen Flussalgorithmus berechnet werden. Dazu ergänzt man den Graphen um einen Quellknoten s und Kanten von s zu allen Knoten die Zeilen repräsentieren, sowie um einen Zielknoten t und Kanten zu ihm von jedem Knoten der eine Spalte repräsentiert. Für das Matching müssen außerdem alle Kantengewichte gleich 1 gewählt werden.
Beispielsweise Ford Fulkerson löst das Matchingproblem in der gewünschten Laufzeit. Seine Laufzeit ist $O(\text{Anzahl Kanten} \cdot \text{Anzahl Knoten} \cdot \text{maximales Kantengewicht})$. Es werden $m+2n$ Kanten benötigt, $n+2$ Knoten und das maximale Kantengewicht ist 1. Dies ergibt $O(n^2 m)$. Da es maximal $m \leq n^2$ Computer geben kann, folgt $O(n^4)$.

Lösungsvorschlag

Aufgabe 5. Externe Algorithmen: Summer School II [9 Punkte]

Ein Freund hatte Ihnen von einer faszinierenden *Summer School* auf Sizilien im letzten Jahr berichtet. Seine Erzählungen klangen so gut, dass Sie beschlossen, sich für dieses Jahr auch zu bewerben. Doch als Sie nun einen Brief der Schule in Händen halten, finden Sie darin weder eine Zu- noch Absage, sondern einen verzweifelten Hilferuf der Schule an Sie als erfahrenen Algorithmiker.
Ihr Freund hatte Ihnen bereits berichtet, dass es schon im letzten Jahr durch die begrenzte Teilnehmeranzahl zu Problemen gekommen sei. Er hatte den Organisatoren damals geholfen, ein möglichst gemischtes Teilnehmerfeld auszuwählen. Die Teilnehmer sollten dabei aus einer Menge an n Bewerbern ausgewählt werden, so dass keine zwei Teilnehmer schon zuvor miteinander befreundet waren. Diese Auswahl sollte mit Hilfe von m bekannten Freundschaften getroffen werden. Für diesen Zweck hatte Ihr Freund einen Algorithmus zur Berechnung eines *Maximal Independent Set* entworfen.
Doch genau hier scheint das Problem zu liegen. Die Ergebnisse dieses Algorithmus haben den Veranstaltern so gut gefallen, dass sie ihn auch dieses Jahr wieder einsetzen wollten. Allerdings wurde die Teilnehmerzahl so weit erhöht, dass die sizilianischen Rechner nicht mehr damit zurechtkamen. Die anfallenden Daten passen einfach nicht mehr in den Arbeitsspeicher und für neue Computer ist nicht genug Geld vorhanden.
Die Freundschaftsbeziehungen sind in Form einer sortierten Liste auf der Festplatte gespeichert. Gegenseitige Freundschaften sind in beiden Richtungen gespeichert.

Variablen: ***n****: Anzahl Bewerber,* ***m****: Anzahl Freundschaftsbeziehungen;*
B*: Blockgröße,* ***M****: Größe des Arbeitsspeichers (siehe Teilaufgaben)*

a. Entwerfen Sie einen Algorithmus, der ein *Maximal Independent Set* in $O(m \log n)$ interner Arbeit und $O(m/B)$ I/O Operationen berechnet. Nehmen Sie an, dass die Knoten des *Independent Sets* in den Speicher passen, aber dass der komplette Graph nicht mehr im Speicher aufgebaut werden kann (3 Punkte). Begründen Sie die Laufzeit Ihres Algorithmus (1 Punkt). [4 Punkte]

Lösung

Jede Kante wird doppelt als die Paare (v,w) und (w,v) extern gespeichert. Man iteriert über die sortierten Paare und prüft für jeden Knoten, ob ein Nachbar bereits im *Independent Set* enthalten ist. Dies kann mit einem balancierten Baum getestet werden. Fällt die Überprüfung für alle Nachbarn negativ aus, wird der Knoten ins *Independent Set* aufgenommen.
Da im Set maximal n Knoten enthalten sind, kann obige Überprüfung in $O(\log n)$ pro Paar geschehen. Bei $O(m)$ Kanten hat der Algorithmus die geforderte Laufzeit $O(m \log n)$.
Jede Kante wird genau einmal gelesen. Da die Kanten in der gleichen Reihenfolge abgearbeitet werden, wie sie im externen Speicher liegen, müssen $O(m/B)$ Speicherzugriffe getätigt werden.

b. Entwerfen Sie einen I/O effizienten Algorithmus, der ein *Maximal Independent Set* berechnen kann, das selbst nicht mehr in den Speicher passt (4 Punkte). Geben Sie die I/O Komplexität Ihres Algorithmus an (1 Punkt). [5 Punkte]

Hinweis: Sie dürfen $n < M^2/B$ annehmen

Lösung

Der Basisansatz des Algorithmus funktioniert identisch zum Hauptspeicheralgorithmus. Wir betrachten die Kanten in sortierter Reihenfolge. Zusätzlich zu der sortierten Kantenfolge halten wir eine Ausschlussliste in Form einer externen Prioritätsliste. Wenn ein Knoten betrachtet wird, entfernen wir so lange Knoten aus der Prioritätsliste, bis sie entweder leer ist, oder ein Knoten mit einer ID größer oder gleich der des aktuellen Knoten das Minimum bildet. Ist die ID gleich der des aktuellen Knotens, so verwerfen wir ihn. Ansonsten ist er Teil des *MIS*. Alle Nachbarknoten mit höherer ID werden in diesem Fall in die Ausschlussliste übernommen. Die I/O Komplexität ist gegeben durch $O(\text{pq}(m)) = \frac{2m}{B}\left(1 + \lceil \log_{M/B} \frac{m}{M} \rceil\right)$.

Lösungsvorschlag

Aufgabe 6. Online Algorithmen: Kuhweg [7 Punkte]

Kühe haben es schwer im Leben, insbesondere kurzsichtige Kühe. Deshalb sind sie oft von dem Drang getrieben, in die Freiheit entkommen zu wollen. Erst vor einem Jahr erreichte die Milchkuh *Yvonne* größte Aufmerksamkeit, als es ihr gelang aus ihrem Gatter zu entkommen und sich drei Monate in einem Wald zu verstecken. Vor dieser Flucht stand allerdings ein schwieriges *Online Problem*:

Als Yvonne erfuhr, dass der Zaun der Weide ein Loch hatte, begab sie sich sofort zum Zaun, um an ihm entlang nach dem Loch zu suchen und zu entkommen. Allerdings wusste sie nicht, in welcher Richtung es näher zum Loch war – und die Zeit drängte, musste sie doch fliehen, bevor der Bauer etwas bemerkte. Zu allem Überfluss war Yvonne auch noch kurzsichtig und konnte das Loch nur in höchstens 1m Abstand erkennen.

Nehmen Sie an, dass Yvonne ihre Suche am Rand der Weide, direkt am Zaun, beginnt.

Variablen: l*: Umfang der Weide (siehe Teilaufgaben)*

a. Angenommen der Umfang der Weide sei $l < \infty$. Geben Sie den kompetetiven Faktor an für die Strategie, die Weide in **einer** Richtung zu umrunden. [2 Punkte]

Lösung

Der schlimmste Fall für diese Strategie liegt vor, wenn das Loch $1+\varepsilon$ hinter dem Startpunkt von Yvonne entgegen ihrer Laufrichtung liegt. Eine optimale Lösung legt also die Strecke $1+\varepsilon$ zurück, der *worst case* die Strecke $l-1-\varepsilon$. Somit ergibt sich ein kompetetiver Faktor von $\frac{l-1-\varepsilon}{1+\varepsilon} \to l-1$ für $\varepsilon \to 0$.

b. Sei $l = \infty$. Entwerfen Sie eine Strategie, mit der Yvonne höchstens 10mal so weit laufen muss, wie im optimalen Fall (3 Punkte). Beweisen Sie den kompetitiven Faktor (2 Punkte). [5 Punkte]

Lösung

Nutze *doubling* als Strategie: Laufe zuerst $d = 1$ in eine Richtung. Wird das Loch nicht gefunden, kehre zum Start zurück, verdopple d und starte in die Gegenrichtung. Wiederhole diesen Vorgang so oft, bis das Loch gefunden wurde.
(jeder alternierende Algorithmus mit multiplikativ wachsender Suchdistanz erhält 3 Punkte; bei additiv wachsender Suchdistanz gibt es noch 2 Punkte)

Der *worst case* für diesen Ansatz liegt vor, wenn wir direkt vor dem Loch umkehren. Das Loch liegt also bei einer Distanz $2^k + \varepsilon$, mit k gleich der Anzahl Richtungswechsel. Die bis dahin zurückgelegte Distanz von Yvonne ist $2 \cdot \sum_{i=0}^{k-1} 2^i + 2^k$. Zusätzlich muss Yvonne nun noch den Weg $2^k + 2 \cdot 2^{k+1} + (2^k + \varepsilon)$ zurücklegen, um beim Loch anzukommen. Damit ergibt sich für diese Strategie $ALG = 2 \cdot \sum_{i=0}^{k+1} 2^i + (2^k + \varepsilon) = 2 \cdot (2^{k+2} - 1) + (2^k + \varepsilon) < 8 \cdot 2^k + 2^k < 9 \cdot OPT$ mit $OPT = 2^k + \varepsilon$.
Die Strategie hat also den geforderten kompetitiven Faktor von $\frac{ALG}{OPT} = 9 < 10$.

Schematische Darstellung des *Kuhpfades*: