ETH Zürich
Institut für Theoretische Informatik
Prof. Dr. A. Steger
Ueli Peter

Algorithmen und Komplexität
Lösungsvorschlag Musterklausur 1

Lösungsvorschlag zu Aufgabe 1

a) Nein. Zum Beispiel bei einer bereits sortierten Sequenz benötigt Quicksort $\Omega(n^2)$ Vergleiche (wenn man das erste Element als Pivot benutzt).

b) Nein. Der Speicheraufwand ist $O(|V| + |E|)$ und somit optimal.

c) Ja. Sei w das kleinste Gewicht im Graphen. Wie man leicht sieht, wählt der Algorithmus genau den gleichen Spannbaum falls wir zu allen Gewichten $|w|$ addieren. Da unter dieser Transformation auch der minimale Spannbaum der selbe ist funktioniert Prim auch mit negativen Kantengewichten.

d) Nein. Prim hat Laufzeit $O(n \log n + m)$ und Kruskal $O(m \log n)$. Prim ist also für alle n und $m \geq n - 1$ asymptotisch mindestens gleich schnell wie Kruskal.

e) Ja. Aus $A \in P$ wissen wir, dass es einen Algorithmus gibt, welcher A in polynomieller Zeit entscheidet. Dieser Algorithmus kann jede Eingabe effizient als Ja- Instanz prüfen.

f) Nein. Beide Probleme sind NP-vollständig.

g) Ja. $\log^3(n^2) = \Theta(\log^3 n)$ und $\log n^3 = \Theta(\log n)$.

h) Ja. Weil $10^5 n = O(n^2)$

i) Nein. Wenn z.B. $f(n) = n$ gilt, dann gilt $f(n) + g(n) = O(n)$.

j) Nein. $n^2 + 2n = \frac{1}{2}n^2 + \frac{1}{2}n^2 + 2n$ und $\frac{1}{2}n^2 + 2n = \Omega(n^2)$.

Lösungsvorschlag zu Aufgabe 2

a) Wir benutzen Kruskal. Zuerst werden die Kanten nach Gewicht sortiert: $1, 2, 3, 4, 5, 6, 7, 8, 9$.

Dann werden die Kanten der Reihe nach betrachtet. Die Kanten $4, 7, 8, 9$ werden nicht gewählt, weil sie einen Kreis schliessen.

b) Zwecks Widerspruch nehmen wir an, dass es einen Graphen G mit paarweise unterschiedlichen Kantengewichten und zwei unterschiedlichen minimalen Spannbäumen T und T' gibt. Folglich gibt es eine Kante $e \in T \setminus T'$ und einen Kreis C in $T' \cup \{e\}$. Wenn es eine Kante $e' \in C$ gibt mit $w(e') > w(e)$ dann hat der Spannbaum $T' \cup \{e\} \setminus \{e'\}$ kleineres Gewicht als T'. Folglich hat e grösstes Gewicht in C. Es muss auch eine Kante $\bar{e} \in C \setminus T$ geben, so dass $\bar{T} := T \setminus \{e\} \cup \{\bar{e}\}$ ein Spannbaum von G ist (mindestens eine Kante von $C \setminus \{e\}$ muss zwischen den zwei Komponenten von $T \setminus \{e\}$ liegen). Weil $w(\bar{e}) < w(e)$ gilt, hat $\bar{T}$ kleineres Gewicht als T, was ein Widerspruch zur Minimalität von T ist.

c) Wir berechnen zuerst mit Prim einen MST T von G. Dann berechnen wir für jede Kante $e \in T$ mit Prim einen MST von $G \setminus \{e\}$ und geben den kleinsten dieser $n - 1$ Bäume aus.

Da der zweitkleinste Spannbaum mindestens eine Kante von T nicht enthält, finden wir ihn mit oben beschriebenem Algorithmus.

Weil die Laufzeit eines Aufrufs von Prim $O(|E| + |V| \log |V|)$ ist und wir Prim exakt $|V|$ mal aufrufen, hat unser Algorithmus die geforderte Laufzeit.

Lösungsvorschlag zu Aufgabe 3

Wir benutzen das Prinzip der Dynamischen Programmierung. Zuerst wollen wir eine Rekursion für die Funktion

$k(s,i,j) :=$ min #Operationen um die Zeichenkette $s[i], s[i+1], \ldots, s[j]$ in ein Palindrom zu verwandeln

aufstellen.

Wir beobachten, dass wir falls $s[i]$ gleich $s[j]$ ist immer $s[i]$ und $s[j]$ stehen lassen können und das Teilproblem $s[i+1], \ldots, s[j-1]$ in ein Palindrom verwandeln können. Angenommen dies wäre nicht der Fall, dann wäre $k(s,i,j)$ echt kleiner ist als $k(s,i+1,j-1)$. Dies würde bedeuten, dass entweder $s[i]$ oder $s[j]$ gelöscht wird, weil beide zu ersetzen oder beide zu löschen nicht zu einer optimalen Sequenz von Operationen führen kann. Nehmen wir also ohne Verlust der Allgemeinheit an, dass $s[i]$ gelöscht wird. Folglich gilt $k(s,i+1,j) \leq k(s,i+1,j-1) - 2$. Dies ist offensichtlich nicht möglich.

Falls $s[i] \neq s[j]$ können wir entweder in einer Operation die beiden Zeichen angleichen, oder eines der beiden Zeichen löschen.

Offensichtlich gilt $k(s,i,j) = 0$ für $i \geq j$ und wir kriegen folgende Rekursion:

$$k(s,i,j) = \begin{cases} 0, & \text{if } i \geq j, \\ k(s,i+1,j-1), & \text{if } s[i] = s[j], \\ \min(1+k(s,i+1,j),\ 1+k(s,i,j-1),\ 1+k(s,i+1,j-1)), & \text{else.} \end{cases}$$

Unser Program soll für eine Zeichenkette der Länge n die Funktion $k(s,1,n)$ berechnen. Dazu definieren wir für $1 \leq i,j \leq n$ die Tabelle $T[i,j] := s(s,i,j)$, welche wir mit folgendem Algorithmus iterativ ausfüllen können.

Algorithm 1 Pal(n)

```
for i = 1...n do
  T[i,i] := 0
  T[i,i-1] := 0
end for
for ℓ = 1...n do
  for i = 1...n − ℓ do
    if s[i] = s[i+ℓ] then
      T[i,i+ℓ] = T[i+1,i+ℓ−1]
    else
      T[i,i+ℓ] = 1 + min{T[i+1,i+ℓ], T[i,i+ℓ−1], T[i+1,i+ℓ−1]}
    end if
  end for
end for
return T[1,n]
```

Offensichtlich benutzen wir die beschriebene Rekursion und berechnen den Eintrag $T[1,n]$. Es bleibt zu zeigen, dass wir nie auf noch nicht ausgefüllte Elemente zugreifen. Wenn wir $T[i,i+\ell]$ berechnen greifen wir auf die Elemente $T[i+1,i+\ell-1], T[i+1,i+\ell]$ und $T[i,i+\ell-1]$ zu. Wie man leicht sieht, wurden diese Elemente alle bereits in einer früheren Iteration berechnet. Da die äussere Schleife von 1 bis n und die innere von 1 bis $n-\ell \leq n$ geht, haben wir eine totale Laufzeit von $O(n^2)$. Für die Tabelle benötigen wir $O(n^2)$ Speicher.

Lösungsvorschlag zu Aufgabe 4

a) Da wir zu jedem Zeitpunkt einen Zeiger auf das minimale Element haben, können wir dies einfach in $O(1)$ Operationen auslesen und ausgeben.

b) Um das zweitkleinste Element zu finden rufen wir zuerst *ExtractMin* aus und speichern das kleinste Element in der Variable *Min*. Dann lesen wir das zweitkleinste Element aus und retournieren es, bevor wir das kleinste Element wieder einfügen. Unser Aufwand ist also

$$\text{Aufwand ExtractMin } + O(1) + \text{ Aufwand Insert.}$$

Amortisiert gibt dies einen totalen Aufwand von $O(\log n)$. Im schlimmsten Fall (wenn n Elemente in der Wurzelliste hängen) kann der Aufwand aber auf $\Theta(n)$ ansteigen.

c) Wir müssen an jedem inneren Knoten für jedes Kind v speichern wie viele Blätter der Teilbaum mit Wurzel v hat. Da wir bei jeder Delete oder Insert Operation sowieso den Baum traversieren müssen, kann diese Information einfach und ohne asymptotische Kosten mitgeführt werden. Wenn wir nun das k-te Element suchen, dann können wir unsere Suche in der Wurzel beginnen und jeweils zum Kind weitergehen, in dessen Teilbaum das k-te Element liegt. Da unser Baum Tiefe $O(\log n)$ hat, benötigt unsere Funktion $O(\log n)$ Operationen.

Lösungsvorschlag zu Aufgabe 5

Wir beweisen zuerst, dass HALF-CLIQUE in NP liegt. Wie man leicht sieht, dient eine gegebene Clique der grösse $\lfloor |V|/2 \rfloor$ (also eine komplett verbundene Teilmenge $S \subset V$ mit $|S| = \lfloor |V|/2 \rfloor$) als Zertifikat, welches einfach in $O(n^2)$ überprüft werden kann.

Weiter zeigen wir $CLIQUE \leq_p$ HALF-CLIQUE. Sei $\phi = G = (V, E), k$ eine Eingabe für das Problem CLIQUE. Wir konstruieren in polynomieller Zeit eine Eingabe $f(\phi) = (V', E')$ für HALF-CLIQUE, so dass gilt

$$\phi \in CLIQUE \longleftrightarrow f(\phi) \in \text{HALF-CLIQUE}.$$

Die Instanz $f(\phi)$ konstruieren wir wie folgt:

Falls $2k > |V|$, fügen wir zu V noch $2k - |V|$ isolierte Knoten hinzu. Offensichtlich hat G genau dann eine Clique der Grösse k wenn $f(G, k) = G'$ eine Clique der Grösse $\lfloor |V'|/2 \rfloor = \lfloor (|V| + (2k - |V|))/2 \rfloor = k$ hat.

Falls $2k = |V|$ gilt sind die Probleme äquivalent.

Falls $2k < |V|$, fügen wir genau $|V| - 2k$ Knoten zu V hinzu und verbinden jeden dieser neuen Knoten mit allen Knoten. Nun hat G genau eine Clique der Grösse k wenn $f(G, k) = G'$ eine Clique der Grösse $\lfloor |V'|/2 \rfloor$ hat, weil genau $|V| - 2k$ der Knoten jeder maximalen Clique in G' die Knoten aus $V' \setminus V$ sind und folglich $\lfloor |V'|/2 \rfloor - (|V| - 2k) = (|V| + |V| - 2k)/2 - |V| + 2k = k$ Knoten einer Clique der Grösse $\lfloor |V'|/2 \rfloor$ aus V sind.

Die Transformation kann offensichtlich in $O(n^2)$ Operationen durchgeführt werden.