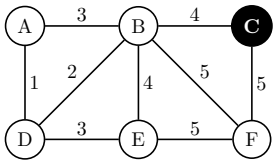
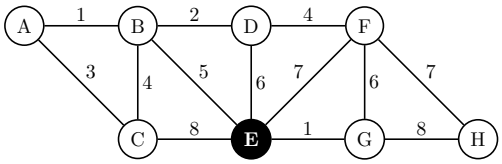
# Übungsblatt 7

Algorithmen und Datenstrukturen (WS 2013, Ulrike von Luxburg)

**Präsenzaufgabe 1 (Minimaler Spannbaum)** Bestimmen Sie für die folgenden beiden Graphen einen minimalen Spannbaum. Nutzen Sie dabei jeweils einmal den Algorithmus von Kruskal und den Algorithmus von Prim (gestartet am schwarzen Knoten).




**Präsenzaufgabe 2 (Dynamische Programmierung)** Dynamische Programmierung basiert darauf, für die gesuchte Gesamtlösung eine *rekursive* Zerlegung in stets kleinere Teilprobleme zu identifizieren. Zum Beispiel entsprechen Divide & Conquer Algorithmen diesem allgemeinen Ansatz, wobei deren Teilprobleme meist so wenige und so klein sind (z.B. nur zwei Teilprobleme halber Größe), dass sich das Gesamtproblem insgesamt effizient lösen lässt.
Leider liefern viele rekursive Zerlegungen nur unwesentlich kleinere Teilprobleme (z.B. Größe nur um 1 verringert). Glücklicherweise kommt es dabei aber häufig vor, dass viele Teilprobleme letztlich auf Lösungen *derselben* kleineren Teilprobleme basieren. Das Gesamtproblem besteht sozusagen aus "überlappenden Teilproblemen". In diesem Fall ist eine naive rekursive Berechnung extrem verschwenderisch, da immer wieder über verschiedene Wege zu stets denselben Teilproblemen hinabgestiegen wird, und diese immer wieder erneut komplett berechnet werden.
Genau an dieser Stelle setzt dynamisches Programmieren an: "*Vermeide das erneute Lösen von bereits gelösten Teilproblemen*!". Dafür gibt es im Wesentlichen zwei Varianten:

**Bottom-Up**: Finde eine Reihenfolge aller Teilprobleme (üblicherweise von klein nach groß), so dass zum Zeitpunkt der Berechnung eines Teilproblems alle dafür benötigten Teilprobleme bereits zuvor gelöst wurden. Berechne alle Teilprobleme in genau dieser Reihenfolge und speichere dabei ihr jeweiliges Ergebnis ab (z.B. in einem Array). Jedes später zu lösende Teilproblem kann dann in konstanter Zeit im Array auf die von ihm verwendeten Teilergebnisse zugreifen, ohne diese erneut berechnen zu müssen.

**Top-Down (durch "Memoization")**: Führe die Rekursion ohne Umschweife auf dem Gesamtproblem aus, z.B. sei dies eine rekursive Funktion mit dem Eingabeparametersatz $P$. Zu Beginn der Funktion überprüfe nun zunächst, ob sie mit exakt derselben Eingabe früher schonmal aufgerufen und das Ergebnis gespeichert wurde (je nach Datentyp von $P$ in einem Array oder Hashtable). Wenn ja, dann liefere sofort das gespeicherte Ergebnis für Eingabe $P$ zurück. Wenn nein, dann berechne das Ergebnis vollständig (rekursiv), aber speichere es zum Ende des Funktionsaufrufs als Rückgabewert für die Eingabe $P$, so dass es bei jedem späteren Aufruf der Funktion mit derselben Eingabe $P$ nicht erneut berechnet werden muss. Auf diese Weise wird das Ergebnis der Funktion für jede mögliche Eingabe höchstens einmal berechnet.

Betrachten Sie nun erneut die rekursive Struktur der Fibonacci-Zahlen ($F_0 := 0, F_1 := 1, F_n := F_{n-1} + F_{n-2}$), und deren direkte Umsetzung als rekursiver Programmcode:

```
function FIB(n)
    if n ≤ 1 then
        return n
    else
        return FIB(n−1) + FIB(n−2)
    end if
end function
```

(a) Warum ist diese Umsetzung so extrem ineffizient? ($\Omega(2^n)$ viele Additionen)

(b) Berechnen Sie dieselbe Rekursion mittels dynamischer Programmierung ("Bottom-Up"). Wieviele Additionen werden nun benötigt?

(c) Wie (b), nun aber mittels "Top-Down".

**Präsenzaufgabe 3 (Dynamisches Programmieren II)** Sei $\mathbf{s} = (s_1, \ldots, s_n) \in \mathbb{R}^n$ eine beliebige Zahlenfolge. Eine *Teilfolge* von $\mathbf{s}$ besteht aus einer beliebigen Auswahl der Elemente von $\mathbf{s}$ unter Beibehaltung ihrer Reihenfolge. Wir interessieren uns für die Länge $L(\mathbf{s})$ einer längsten monoton wachsenden Teilfolge in $\mathbf{s}$. Zum Beispiel ist dies in $\mathbf{s} = (5, 3, 4, 1, 7, 6, 7)$ die Länge $L(\mathbf{s}) = 4$, aufgrund der Teilfolge $(3, 4, 7, 7)$.

(a) Finden Sie eine rekursive Berechnung von $L(\mathbf{s})$.

(b) Leiten Sie aus (a) mittels dynamischer Programmierung einen Algorithmus ab, der Ihnen $L(\mathbf{s})$ in Laufzeit $\mathcal{O}(n^2)$ liefert.

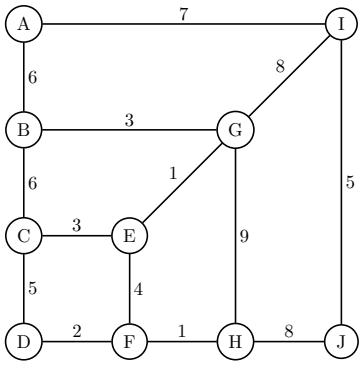
(c) Erweitern Sie Ihren Algorithmus so, dass er zudem die zugehörige Teilfolge ausgibt.

---

**Hausaufgaben zum 29. Januar, 9:00 (Vorlesungsbeginn).**

---

**Aufgabe 1 (Minimale Spannbäume (2+2+2+2))** Gegeben sei folgender Graph $G$:




(a) Führen Sie Kruskal's Algorithmus aus, um einen minimalen Spannbaum für $G$ zu bestimmen. Geben Sie dessen Kanten in der Reihenfolge ihrer Hinzufügung an.

(b) Nutzen Sie in (a) die Union-Find-Datenstruktur (genauer: Union-By-Rank, wobei im Falle gleichen Rangs der zwei Wurzeln die alphabtisch kleinere der Elternknoten wird), und geben Sie deren Endzustand an. Es genügt, den Elternknoten jedes Knotens zu nennen.

(c) Führen Sie Prim's Algorithmus gestartet an $\{\mathrm{A}\}$ aus, um einen minimalen Spannbaum für $G$ zu bestimmen. Geben Sie die Kanten in der Reihenfolge ihrer Hinzufügung an.

(d) Begründen Sie am Pseudocode, warum Prim's Algorithmus Laufzeit $\mathcal{O}(|E| \cdot \log |V|)$ hat.

**Aufgabe 2 (Dynamisches Programmieren (4))** Sei $\Sigma^+ := \bigcup_{k \geq 1} \Sigma^k$ die Menge aller Strings endlicher positiver Länge über einem Alphabet $\Sigma$. Gegeben sei ein String $\mathbf{s} = s_1 \ldots s_n \in \Sigma^+$ der Länge $n$. Es wird vermutet, dass $\mathbf{s}$ einen Text darstellt, aus dem alle Leerzeichen und Interpunktionen entfernt wurden. Zum Beispiel könnte die Eingabe so aussehen: `diesistsinnvoll`. Sie haben Zugang zu einer Wörterbuch-Funktion $dict : \Sigma^+ \to \{0, 1\}$, welche für eine beliebige Eingabe $w$ bestimmt, ob $w$ ein gültiges einzelnes Wort beschreibt ($dict(w) = 1$), oder nicht ($dict(w) = 0$).
Nutzen Sie dynamische Programmierung, um effizient zu ermitteln, ob der Eingabestring $\mathbf{s}$ in eine Folge von gültigen Worten zerlegt werden kann (Ausgabe 1), oder nicht (Ausgabe 0). Die Laufzeit darf $\mathcal{O}(n^2)$ betragen, wobei jeder Aufruf von $dict$ Zeit $\mathcal{O}(1)$ benötigt.

**Die folgenden Aufgaben sind Bonusaufgaben, die Sie insbesondere dann lösen sollten, wenn Ihre bislang erreichte Gesamtpunktzahl die 50 %-Hürde noch nicht oder nur knapp überschreitet.**
**Außerdem enthält die allerletzte Aufgabe als besonderes Schmankerl eine Schatzsuche!**

---

**Bonusaufgabe 3 (Windpark-Planung (3 Bonuspunkte))** Das Energieunternehmen E.Off hat sich im Landkreis Dithmarschen auf 7 Standorte für neu zu errichtende Windräder $W_1, \ldots, W_7$ festgelegt. Nur $W_1$ kann jedoch direkt an das internationale Stromnetz angeschlossen werden. Daher müssen die Windräder untereinander mit zusätzlichen Leitungen zu einem Verbund zusammengeschlossen werden. Lagebedingt stehen dafür nur ausgewählte paarweise Verbindungen zur Disposition. Die Kosten $k(i,j)$ für eine Direktverbindung von $W_i$ und $W_j$ betragen in Tsd. EUR:

$k(1,2) = 25$, $k(1,3) = 14$, $k(1,4) = 13$, $k(2,3) = 2$, $k(2,5) = 6$, $k(3,4) = 11$, $k(3,5) = 9$,
$k(3,6) = 13$, $k(4,5) = 16$, $k(4,6) = 9$, $k(4,7) = 10$, $k(5,6) = 8$, $k(6,7) = 2$.

E.Off möchte den Windpark natürlich kostenoptimal anschließen. Führen Sie diese Problemstellung auf ein Ihnen aus der Vorlesung bekanntes Problem zurück und ermitteln Sie eine Lösung.

**Bonusaufgabe 4 (Minimaler Spannbaum (2+2 Bonuspunkte))** Sei $G = (V, E)$ ein zusammenhängender, ungerichteter Graph mit positiven Kantengewichten, und $w(G) = \sum_{e \in E} w_e$ das Gesamtgewicht von $G$. Beweisen Sie folgende Aussagen:

(a) Jeder zusammenhängende Teilgraph $G' = (V, E')$ mit $E' \subseteq E$ und minimal möglichem $w(G')$ unter all solchen Teilgraphen ist ein minimaler Spannbaum.

(b) Sei $V = A \dot\cup B$ eine beliebige nicht-triviale Partition von $V$ (d.h., $V = A \cup B$, sowie $A \cap B = \emptyset$ und $A, B \neq \emptyset$). Der Schnitt $cut(A,B) := \{e_{ab} \in E \mid a \in A, b \in B\}$ heiße *Z-Schnitt*, wenn der durch $A$ implizierte Teilgraph $G_{|A} := (A, E_{|A \times A})$ und der durch $B$ implizierte Teilgraph $G_{|B} := (B, E_{|B \times B})$ jeweils zusammenhängend ist. Zeigen Sie, dass $G$ genau dann ein Baum ist, wenn jeder Z-Schnitt genau eine Kante enthält.

**Bonusaufgabe 5 (Bernoullische Ungleichung (2 Bonuspunkte))** Beweisen Sie per Induktion für alle $n \in \mathbb{N}_{\geq 0}$ und $x \in \mathbb{R}_{\geq -1}$:

$$(1+x)^n \geq 1 + nx$$

**Bonusaufgabe 6 (Sortierverfahren (2 Bonuspunkte))** Jede Spalte in folgender Tabelle gibt die wesentlichen (nicht alle!) Zwischenschritte eines Sortieralgorithmus an. Dabei handelt es sich um In-Place-Varianten von BubbleSort, MergeSort, InsertionSort und QuickSort, wobei letzterer stets das ganz links im aktuellen (Teil-)Array stehende Element als Pivotelement wählt. Alle Algorithmen werden auf dieselbe Eingabe

| 6 | 3 | 5 | 1 | 8 | 7 | 4 | 2 |
|---|---|---|---|---|---|---|---|

angewandt.

Welche Spalte gehört zu welchem Sortieralgorithmus?

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 6 | 3 | 5 | 1 | 8 | 7 | 4 | 2 |
| 3 | 6 | 5 | 1 | 8 | 7 | 4 | 2 |
| 3 | 5 | 6 | 1 | 8 | 7 | 4 | 2 |
| 1 | 3 | 5 | 6 | 8 | 7 | 4 | 2 |
| 1 | 3 | 5 | 6 | 7 | 8 | 4 | 2 |
| 1 | 3 | 4 | 5 | 6 | 7 | 8 | 2 |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 6 | 3 | 5 | 1 | 8 | 7 | 4 | 2 |
| 1 | 6 | 3 | 5 | 2 | 8 | 7 | 4 |
| 1 | 2 | 6 | 3 | 5 | 4 | 8 | 7 |
| 1 | 2 | 3 | 6 | 4 | 5 | 7 | 8 |
| 1 | 2 | 3 | 4 | 6 | 5 | 7 | 8 |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 6 | 3 | 5 | 1 | 8 | 7 | 4 | 2 |
| 3 | 5 | 1 | 4 | 2 | 6 | 8 | 7 |
| 1 | 2 | 3 | 5 | 4 | 6 | 7 | 8 |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 6 | 3 | 5 | 1 | 8 | 7 | 4 | 2 |
| 3 | 6 | 1 | 5 | 7 | 8 | 2 | 4 |
| 1 | 3 | 5 | 6 | 2 | 4 | 7 | 8 |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |

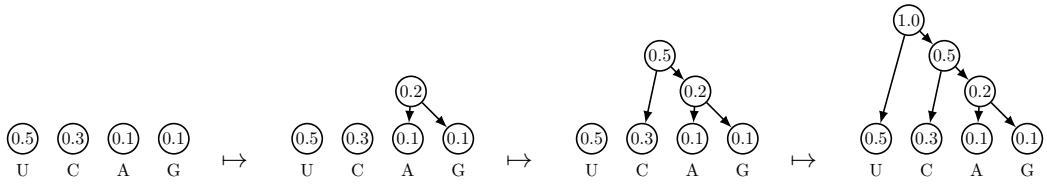
**Bonusaufgabe 7 (Huffman Coding (1+1+1+3+1 Bonuspunkte + eine Schatzsuche!))**
Folgender Algorithmus konstruiert "greedy" einen DAG $G = (V, E)$, der zur optimal komprimierten Darstellung beliebiger Zeichenketten über einem endlichen Alphabet $\Sigma$ genutzt werden kann, sofern die Häufigkeitsverteilung der Buchstaben bekannt ist (*Huffman Coding*):

*(1) Erzeuge für jedes $s \in \Sigma$ einen Knoten, gelabelt mit $f_s$, der "Häufigkeit" von $s$. Ordne diese horizontal anhand absteigender Häufigkeiten an. Die Menge der Knoten heiße $V$.*

*(2) Sei $T := \{v \in V \mid \text{in-degree}(v) = 0\}$. Solange $|T| > 1$, wähle "greedy" zwei verschiedene Knoten $i, j \in T$ mit minimaler Summe $f_i + f_j$, und füge dem DAG einen neuen Knoten $k$ hinzu, der mit $f_k := f_i + f_j$ gelabelt ist und mit je einer Kante nach $i$ und $j$ verbunden.*

*(3) Wenn nun $|T| = 1$, also $T = \{w\}$, dann ordne jedem Blattknoten $s \in \Sigma$ den eindeutigen Pfad von $w$ nach $s$ zu, beschrieben durch eine Sequenz aus '0' für 'links' und '1' für 'rechts'.*

Als Beispiel betrachten wir $\Sigma = \{U, C, A, G\}$ mit $(f_U, f_C, f_A, f_G) = (0.5, 0.3, 0.1, 0.1)$:




Wir erhalten somit die "Codierungen" $c(U) = 0$, $c(C) = 10$, $c(A) = 110$ und $c(G) = 111$. Jedes Wort $w = s_1 \dots s_k \in \Sigma^k$ kann nun eindeutig durch Konkatenation dieser Codierungen beschrieben werden. Zum Beispiel wird $w = UUCUAU$ beschrieben durch $c(w) = 001001100$. Andersherum beschreibt 110100 das Wort $ACU$. Jeder Buchstabe in $\Sigma$ ließe sich alternativ durch einfaches Durchnummerieren mit $b = \lceil \log_2 |\Sigma| \rceil$ Bits eindeutig beschreiben, jedes Wort $w \in \Sigma^+$ also durch $|w| \cdot b$ Bits. Obige Codierung nutzt stattdessen $|c(w)|$ Bits. Das Verhältnis $|w| \cdot b / |c(w)|$ wird als "Kompressionsfaktor" des Wortes $w$ bezeichnet.

(a) Bestimmen Sie den Kompressionsfaktor des Wortes $w = UACUUCUG$.

(b) Bestimmen Sie ein Wort der Länge 10 mit bestmöglicher Kompression.

(c) Bestimmen Sie ein Wort der Länge 10 mit schlechtestmöglicher Kompression.

(d) Die Matrikelnummern der AD-Studenten enthalten die Ziffern 0-9 mit folgenden Häufigkeiten:

| Ziffer $i$ : | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| $f_i$ : | 86 | 92 | 170 | 301 | 297 | 87 | 592 | 68 | 79 | 69 |

Konstruieren Sie daraus den Codierungs-Baum wie oben beschrieben und geben Sie die resultierenden Codierungen für alle Ziffern an. Codieren Sie dann die Matrikelnummer jedes Mitglieds Ihrer Gruppenabgabe, und geben Sie jeweils dessen Codierung und den jeweiligen Kompressionsfaktor an.

(e) Decodieren Sie in (d) die Folge $c(w) = 11011011011111111001110000101001010$ und geben Sie $w$ auf folgender Internetseite ein:

`http://www2.informatik.uni-hamburg.de/ML/AD-2013/treasure/`

Dort finden Sie eine Schatzkarte!