Lösungsskizzen zur Prüfung

(1) Siehe Skriptum Seite 20-25.

(2) Eine dafür geeignete Datenstruktur ist z.B. der Stack (Stapel). Durch Vorgehen nach dem im Skriptum beschriebenen Algorithmus zur Umwandlung in einen Postfix-Ausdruck ergibt sich für den Stack im Verlauf folgendes Bild:

Infix-Ausdruck: $3 * (9 - 4) - (14/(7 - 5) + (2 + 2) * 2)$

Die Zwischenschritte schauen für die Umwandlung und für die Berechnung des Ergebnisses beispielsweise wie folgt aus:

Input	Stack	Output
3		3
*	*	
(	*(	
9	*(	9
-	*(-	
4	*(-	4
)	*	-
-	-	*
(	-(	
14	-(	14
/	-(/	
(	-(/(	
7	-(/(	7
-	-(/(-	
5	-(/(-	5
)	-(/	-
+	-(+	/
(	-(+(	
2	-(+(	2
+	-(+(+	
2	-(+(+	2
)	-(+	+
*	-(+*	
2	-(+*	2
)	-	* +
EOF		-

Input	Stack
3	3
9	3 9
4	3 9 4
-	3 5
*	15
14	15 14
7	15 14 7
5	15 14 7 5
-	15 14 2
/	15 7
2	15 7 2
2	15 7 2 2
+	15 7 4
2	15 7 4 2
*	15 7 8
+	15 15
-	0

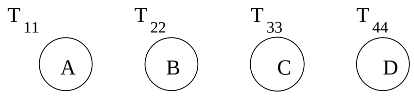
(3)

(a) Siehe Skriptum Seiten 88-89.

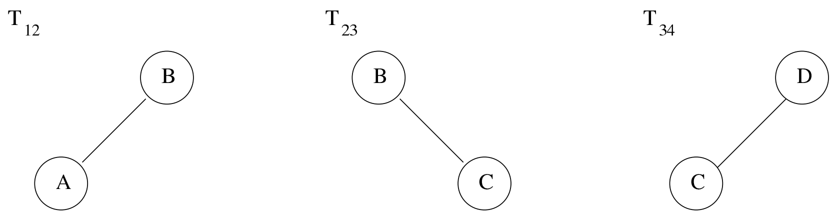
(b) Dynamisches Programmieren löst Probleme durch Kombination der Lösungen von Teilproblemen. Im Gegensatz zu Divide-and-Conquer sind die Teilprobleme dabei im Allgemeinen nicht unabhängig. Weiters wird im Gegensatz zum rekursiven D&C-Ansatz die Lösung für jedes Teilproblem gespeichert. Tritt also ein Teilproblem in mehreren "übergeordneten" Problemen auf, so wird es nur einmal berechnet und später lediglich wieder auf die schon berechnete Lösung zugegriffen. Bei D&C würde das Teilproblem dabei beliebig oft erneut berechnet werden!

(c) Berechnung siehe Skriptum Seite 89. Die Ergebnisse lauten:

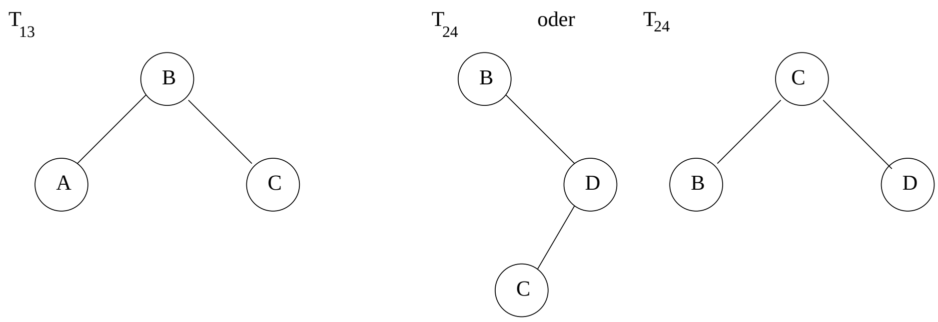
1 Knoten: $c_{11} = 1$, $c_{22} = 4$, $c_{33} = 2$, $c_{44} = 3$



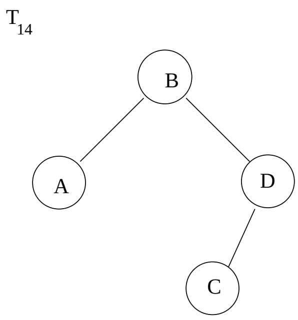
2 Knoten: $c_{12} = 6$, $c_{23} = 8$, $c_{34} = 7$



3 Knoten: $c_{13} = 10$, $c_{24} = 16$

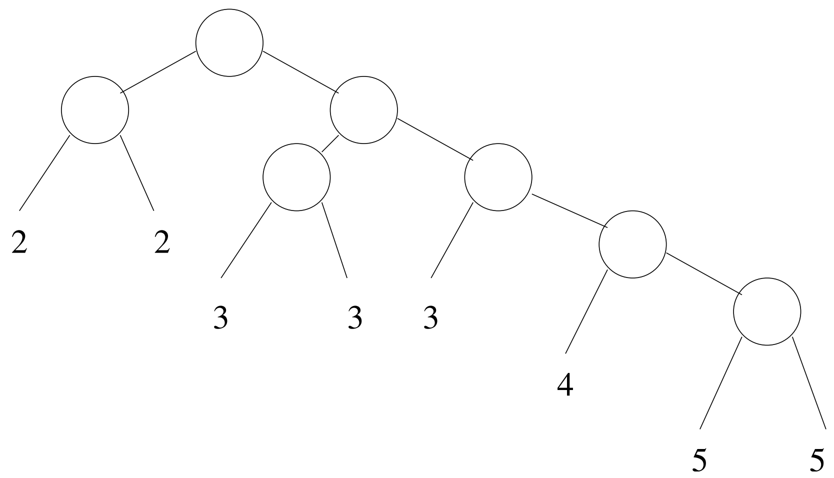


4 Knoten: $c_{14} = 18$



(4)

(a) Ja, ein solcher Code existiert (an den Astenden stehen die jeweiligen Codelängen):



(b) Es existiert zwar ein präfix-freier Code mit diesen Codelängen (siehe Beispiel 4b vom 29.1.2001), aber kein wie in diesem Fall geforderter optimaler präfix-freier Code! Der Codebaum eines optimalen binären Codes ist immer ein voller Binärbaum. Daraus folgt auch, daß es immer $2*i$, $i > 0$ Codewörter geben muß, die am meisten Bits benötigen. Anders gesagt: der oder die tiefsten Knoten eines Codebaums müssen immer 2 Blätter haben (siehe auch voriges Beispiel).