Algorithmen und Datenstrukturen 2
Univ.-Prof. Dr. Alois Ferscha
Klausur am 05.02.2003

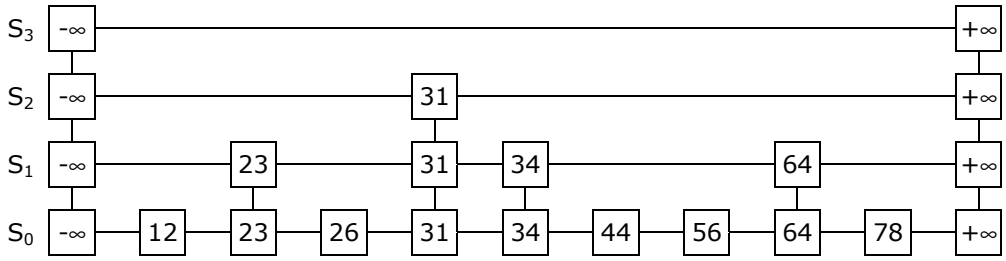
Institut für Praktische Informatik
Gruppe Software
Univ.-Prof. Dr. Alois Ferscha

Zuname: | Hörsaal:
Vorname: | Sitzplatz:
Matr. Nr.: | SKZ: | Punkte / Note:

☐ Ich habe die Übungen schon in einem früheren Semester absolviert: ☐ SS ☐ WS

1. Skip-Listen — 1 + 1 + 1 Punkte

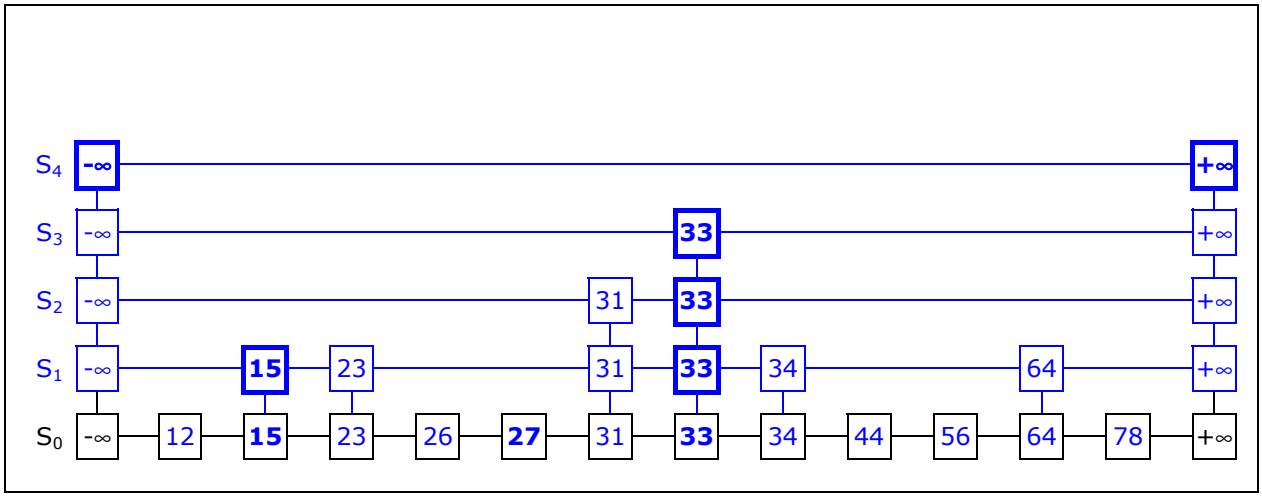
Gegeben sei folgende Skip-Liste:



a.) Wieviele Suchschritte (=Anzahl der Verweistraversierungen) sind notwendig, um folgende Elemente in der Skip-Liste zu finden?

Element:	**78**	**12**	**63**
Suchschritte:	7	4	8

b.) Fügen Sie in die Skip-Liste der Reihe nach die Werte **15**, **27** und **33** ein, wenn der Zufallszahlengenerator fortlaufend folgende Zahlenfolge liefert: 0,34 0,58 0,87 0,49 0,12 0,26 0,69



c.) Geben Sie die erwartete Anzahl von Knoten in der Skip-Liste zur Speicherung von *n* Knoten an:

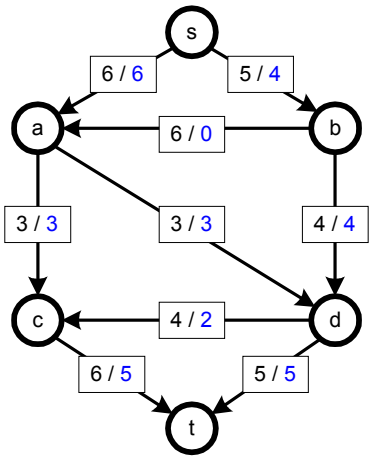
Anzahl der Knoten = (Summe der Knoten in den einzelnen Reihen) + (je 2 Knoten am Rand) =

= (n + n/2 + n/4 + n/8 + ... + $n/2^{ld\,n}$) + (2.ld(n)) $= n.\sum_{i=1}^{ld(n)} \frac{1}{2^i} + 2.ld(n)$ = 2n + 2.ld(n)

2. Maximaler Fluß in Netzwerken — 2 + 1 Punkte

a.) Gegeben ist folgendes Netzwerk *N*:

(Die Angaben bei den Kanten bedeuten: **Kapazität** / **Fluß**)

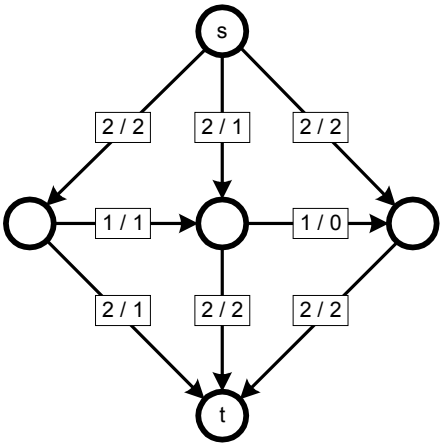


Berechnen Sie nach dem Algorithmus von Ford-Fulkerson den maximalen Fluß (*max flow*) innerhalb dieses Netzwerks von der Quelle *s* zur Senke *t*.

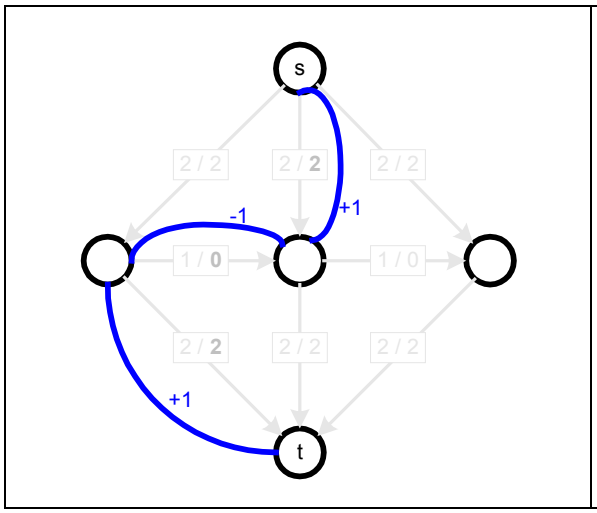
Fügen Sie dazu in die freien Felder der Abbildung rechts für jede Kante den aktuellen Fluß ein.

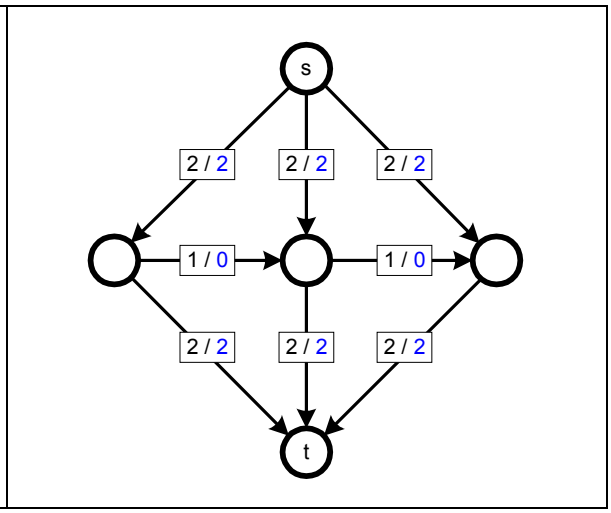
max flow: 10

b.) Gegeben ist folgendes Netzwerk *N* in der *i*-ten Iteration des Ford-Fulkerson Algorithmus zur Bestimmung des maximalen Flusses von der Quelle *s* zur Senke *t*.



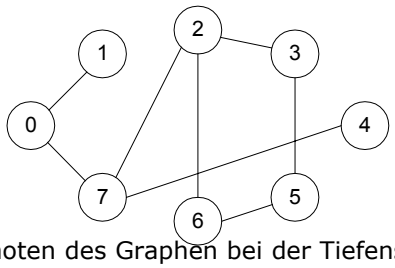
Zeichnen Sie in die Abblidung links unten den Restgraphen ein und führen Sie rechts daneben den nächsten Iterationsschritt aus.





3. Graphen – DFS und BFS — 2 Punkte

Gegeben ist der folgende ungerichtete Graph:



DFS:	2	3	5	6	7	0	1	4
BFS:	2	3	6	7	5	0	4	1

Geben Sie jeweils an, in welcher Reihenfolge die Knoten des Graphen bei der Tiefensuche (DFS) und bei der Breitensuche (BFS) besucht werden (niedriger Index jeweils zuerst). Ausgangsknoten ist in beiden Fällen der Knoten 2.

4. Graphen – BFS Implementierung — 5 Punkte

Implementieren Sie die Breitensuche BFS mit Hilfe der unten angegebenen Klasse *Graph* und unter Verwendung einer Adjazenzmatrix. Zwei adjazente Knoten sind in der Matrix mit 1 gekennzeichnet; der Rest mit 0. Verwenden Sie zum Markieren der besuchten Knoten die Hauptdiagonale der Adjazenzmatrix.

Sie können weiters eine Klasse *Queue* mit folgender Schnittstelle voraussetzen:

```
public class Queue {
    public Queue();                      // legt eine neue, leere Queue an
    public void enqueue(int i);          // fügt das Element i am Ende der Queue ein
    public boolean contains(int i);      // prüft, ob das Element i in der Queue vorhanden ist
    public int dequeue();                // entfernt das älteste Element der Queue.
}                                        // Gibt –1 zurück, wenn die Queue leer ist
```

Gehen Sie davon aus, daß vor dem Aufruf von DFS alle Felder der Hauptdiagonale mit 0 initialisiert wurden und eine leere Queue als protected Member Variable zur Verfügung steht.

```
public class Graph {
    protected int matrix[][] ;        // Adjazenzmatrix
    protected int n ;                 // Anzahl der Knoten im Graphen
    protected Queue queue;

    protected void BFS(int index) {
        this.matrix[index][index] = 1              // Knoten index als besucht markieren
        int i;
        for (i = 0; i < this.n; i++)               // alle erreichbaren Knoten besuchen
            if (this.matrix[index][i] == 1) {      // Knoten i ist erreichbar
                if (this.matrix[i][i] == 0) {      // Knoten i wurde bisher noch nicht besucht
                    if (!this.queue.contains(i))
                        this.queue.enqueue(i);     // Knoten i in Queue einfügen
                }
            }
         }
        i = this.queue.dequeue();
        if (i > 0) {                               // Queue ist noch nicht leer
            this.BFS(i);                           // Besuche Knoten i;



    }
}
```

5. Optimierung – Simplex — 1 + 2 + 1 Punkte

Gegeben ist die folgende Optimierungsaufgabe:

Minimieren Sie den Ausdruck: $3\mathbf{x_1 - 2x_2 + 6x_3 + 18}$
unter den Nebenbedingungen:

$$\begin{aligned} x_1 + x_2 &= 4 - x_4 \\ 2x_1 - 2x_2 + 5x_3 &= 1 - x_5 \\ 2x_2 - 2x_3 &= 4 - x_6 \end{aligned}$$

wobei $x_i \geq 0$ für $i = 1 .. 6$

Stellen Sie das Simplex-Tableau auf und führen Sie darauf den ersten Optimierungsschritt durch.

Tableau: Pivot-Element

		x_1	x_2	x_3
x_4	4	1	1	0
x_5	1	2	-2	5
x_6	4	0	**2**	-2
18		3	-2	6

1. Optimierungsschritt:

		x_1	x_6	x_3
x_4	2	1	-1/2	1
x_5	5	2	1	3
x_2	2	0	1/2	-1
14		3	1	4

Wie interpretieren Sie das Ergebnis nach dem ersten Optimierungsschritt?

Nach dem ersten Optimierungsschritt terminiert das Verfahren bereits, und wir erhalten die Parameter für die optimale Lösung:

$x_1 = 0$
$x_2 = 2$
$x_3 = 0$
$x_4 = 2$
$x_5 = 5$
$x_6 = 0$

6. Rekurrenzen – Master Theorem — 2 Punkte

Ein rekursiver Algorithmus hat folgende Laufzeit (in Abhängigkeit von der Problemgröße *n*):

$$T(n) = 8T\left(\frac{n}{2}\right) + 2n^3 + 4n^2$$

Lösen Sie die Rekurrenz mit Hilfe des Master-Theorems und geben Sie die asymptotische Laufzeitkomplexität für den Algorithmus an.

a = 8

b = 2

f(n) = $2n^3 + 4n^2$

$n^{\log_b a} = n^{\log_2 8} = n^3$

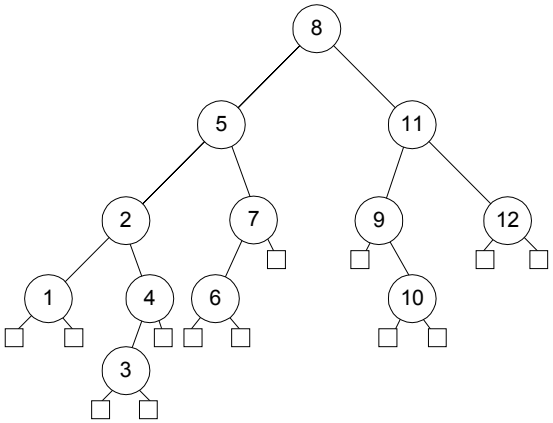
f(n) verglichen mit $n^{\log_b a}$ ergibt Fall 2:

f(n) = $\Theta\left(n^{\log_b a}\right) = \Theta\left(n^3\right)$

$\Rightarrow T(n) = \Theta\left(n^{\log_b a}.\log n\right) = \Theta\left(n^3.\log n\right)$

7. AVL-Baum – Löschen eines Knotens — 3 Punkte

Gegeben ist folgender AVL-Baum:

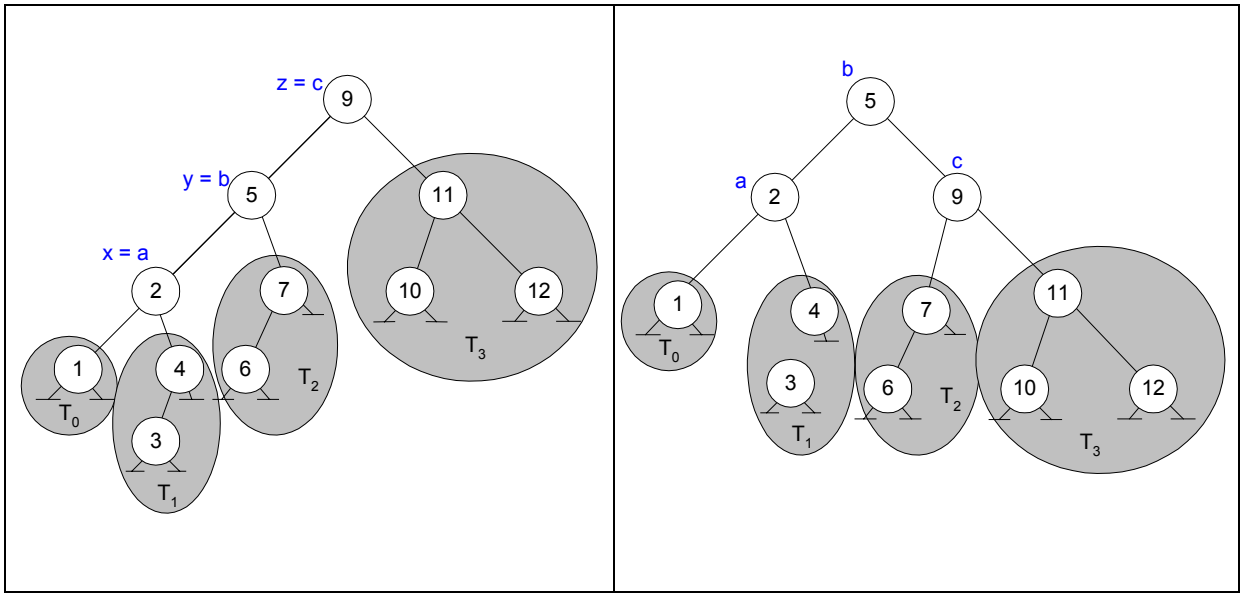


Löschen Sie aus diesem Baum den Wurzelknoten mit dem Schlüssel 8 (Zeichnung im Feld unten links) und zeigen Sie, wie der AVL-Baum nach einer eventuell notwendigen Rotation aussieht (rechts daneben).

Kennzeichnen Sie die Teilbäume T_0 bis T_3 und zeigen Sie, wo diese nach der Rotation eingehängt sind.

Geben Sie auch an, um welche Art der Rotation es sich dabei handelt (Single oder Double Rotation?).

- ☒ Single Rotation
- ❑ Double Rotation



8. MaxHeap - HeapSort — 2 Punkte

Gegeben ist folgender MaxHeap mit *n*=7 Knoten. Die Knoten des MaxHeaps werden — nach dem gleichen Verfahren, wie in der Vorlesung vorgestellt — flach in einem Array gespeichert:

7	6	4	5	1	2	3

Sortieren Sie dieses Array mit Hilfe des *HeapSort*, indem Sie die ersten drei Sortierschritte (= 3 x removeMax + downHeap) ausführen:

1. Sortierschritt:

6	5	4	3	1	2	7

2. Sortierschritt:

5	3	4	2	1	6	7

3. Sortierschritt:

4	3	1	2	5	6	7