Algorithmische Mathematik I

Wintersemester 2009/2010
Prof. Dr. Mario Bebendorf
Dr. Jan Hamaekers

universitätbonn

Übungsblatt 10.

Abgabe am **Mittwoch, 20.1.2010 (vor der Vorlesung)**.

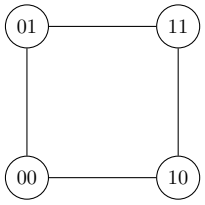
Aufgabe 1. (Hyperwürfel)

Der *n-dimensionale Hyperwürfel* $H(n) = (V, E)$ ist ein wie folgt definierter ungerichteter Graph:

- Die Knoten sind Bitfolgen der Länge n,

$$V := \{\, b_1 \cdots b_n \,|b_i \in \{0, 1\}\}.$$

- Für $p, q \in V$ gilt $(p, q) \in E$ genau dann, wenn p und q sich an genau einer Stelle unterscheiden.



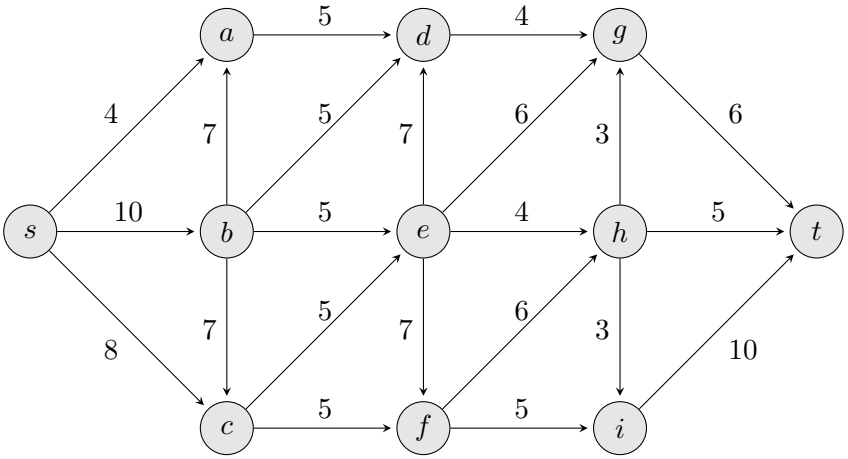
Rechts ist beispielhaft der $H(2)$ dargestellt.

a) Ein *Hamiltonkreis* in einem ungerichteten Graphen ist ein Kreis, der jeden Knoten genau einmal enthält. Ein Graph mit einem Hamiltonkreis heißt hamiltonscher Graph. Zeichnen Sie den dreidimensionalen Hyperwürfel $H(3)$ und geben Sie einen hamiltonschen Kreis für $H(3)$ an.

b) Zeigen Sie mittels vollständiger Induktion: für $n \geq 2$ ist $H(n)$ hamiltonsch.

(10 Punkte)

Aufgabe 2. (Flussnetzwerke)



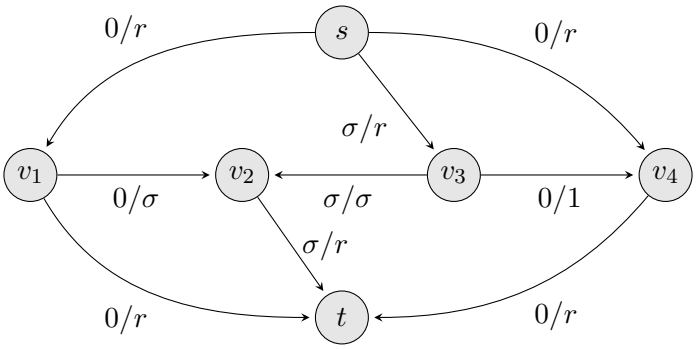
a) Berechnen Sie für das oben stehende Flussnetzwerk einen Maximalfluss f. Die angegebenen Zahlen geben die Kapazität der jeweiligen Kante an. Begründen Sie den Maximalfluß. Zu jedem Schritt sollen die augmentierten Wege angegeben werden.

b) Geben sie einen Schnitt minimaler Kapazität an.

(10 Punkte)

Aufgabe 3. (Ford–Fulkerson)

Gegeben sei folgendes Netzwerk mit Fluss f:



Dabei sei $\sigma = \frac{\sqrt{5}-1}{2}$ und $r = \frac{1}{1-\sigma}$.[1] Der gegebene Fluss resultiert beispielsweise aus dem Nullfluss durch Augmentierung entlang des Wegs $\pi = s, v_3, v_2, t$. Zeigen Sie, dass es eine Folge von augmentierten Wegen $\pi_1, \pi_2, \pi_1, \pi_3, \pi_1, \pi_2, \pi_1, \pi_3, \ldots$ gibt, so dass der Ford–Fulkerson–Algorithmus nicht terminiert. Dazu beachte man, dass $\sigma^n = \sigma^{n+1} + \sigma^{n+2}$ gilt.

(10 Punkte)

Programmieraufgabe 5. (Verkettete Listen/Maximales Matching)

a) Gegeben sei folgender `C`-Programmcode.

```
#include <stdlib.h>
#include <stdio.h>

struct list {
    int value;
    struct list* next;
};
struct list* push_front( struct list* first, int value );
struct list* pop_front( struct list* first );

int main(int argc, char** argv) {
    struct list* a = NULL;
    struct list* tmp = NULL;

    a = push_front( a, 3 );
    a = push_front( a, 5 );
    a = push_front( a, 10 );
    a = push_front( a, -1 );
    a = push_front( a, 20 );
    a = push_front( a, 25 );
    a = push_front( a, 40 );

    for( tmp = a; tmp != NULL; tmp = tmp->next )
        printf( "%d->␣", tmp->value );
    printf("\n");

    while( a != NULL ) {
        a = pop_front( a );
```

[1] Anmerkung: $r = \sum_{k=0}^{\infty} \sigma^k = \frac{1}{1-\sigma}$ ist der Wert der geometrischen Reihe der σ.

```
        for( tmp = a; tmp != NULL; tmp = tmp->next )
            printf( "%d->␣", tmp->value );
        printf("\n");
    }
    return 0;
}
```

Implementieren Sie die Funktionen `push_front()` und `pop_front()` derart, dass

- `push_front(a,i)` ein neues Listenelement `i` *vorne* an die Liste `a` anfügt und
- `pop_front(a)` das erste Element der Liste `a` entfernt.

Der Rückgabewert soll jeweils ein Zeiger auf das (dann neue) erste Element der Liste sein. Beachten Sie, dass Speicher für Listenelemente `struct list` mit `calloc()` alloziiert bzw. mit `free()` freigegeben werden muss.

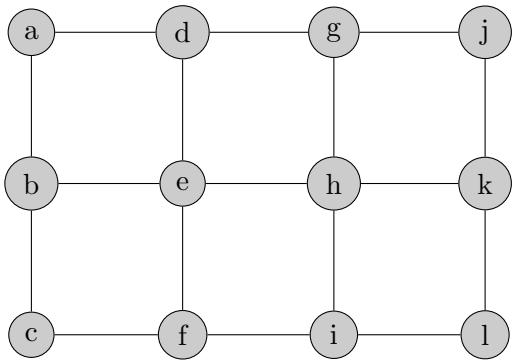
Hinweise zur Verwendung von Zeigern, Funktionen und `struct`s finden Sie in der Literatur oder auf dem Hinweisblatt „Programmierhinweise: `struct`s, Funktionen und Zeiger“ auf der Vorlesungswebseite.

b) Implementieren Sie Algorithmus 5.61 (*Maximales Matching*) aus der Vorlesung. Folgende Anforderungen sollte Ihr Programm erfüllen:

- `Input:` Ein ungerichteter Graph $G = (V, E)$.
- `Output:` Maximales Matching M.
- Die Laufzeitanforderungen dürfen nicht größer als $O(|E| + |V|)$ sein.

Hinweis: Entsprechend Beispiel 5.23 in Kapitel 5.3 (*Speicherung von Graphen*) der Vorlesung können Graphen mit Hilfe eines Arrays von Adjazenzlisten gespeichert werden.

Wenden Sie Ihre Implementation auf folgenden Graphen an:



(10 Punkte)

Achtung: Bezüglich des konkreten Termins der Abgabe der Programmieraufgabe hängen in der Woche vom 18.1.-22.1.2010 im CIP-Pool Listen aus.

Abgabe innerhalb der Woche 25.1.–29.1.2010 im CIP-Pool