Gruppe **14**
Datenstrukturen und Algorithmen

Phil Pützstück, 377247
Benedikt Gerlach, 376944
Sebastian Hackenberg, 377550

Hausaufgabe 9

Aufgabe 1

Wir gehen von beschrifteten Knoten aus, also dass hier 2 isomorphe graphen nicht unbedingt als gleich gezählt werden. D.h. bspw. ein Graph mit Knoten A,B,C und Kante AC ist (isomorphisch zu, aber) nicht gleich dem Graphen mit Knoten A,B,C und Kante BC.

a)

Es gibt genau $4^{\binom{n}{2}}2^n = 2^{n^2}$ gerichtete Graphen mit n Knoten. Für jedes Paar von den $\binom{n}{2}$ Paaren von Knoten haben wir genau 4 Möglichkeiten: keine Kante, Kante von A nach B, Kante von B nach A, 2 Kanten. Dies entspricht dem Faktor $4^{\binom{n}{2}}$. Weiter haben wir für jeden der n Knoten entweder 1 oder keinen self-loop dies entspricht dem weiteren Faktor 2^n.

b)

Es gibt genau $2^{\binom{n}{2}} = 2^{\frac{n(n-1)}{2}}$ ungerichtete Graphen mit n Knoten. Die Idee ist wie in a). Für jedes von den $\binom{n}{2}$ möglichen Paaren von Knoten haben wir entweder 1 Kante oder keine, also 2 Möglichkeiten pro Paar. Ein ungerichteter Graph hat keine self-loops.

c)

Es gibt genau $\frac{n!}{(k-1)!} = n \cdot (n-1) \cdot (n-2) \cdots (n-k)$ mögliche einfache Pfade der Länge k in einem ungerichtetem vollständigem Graphen mit n Knoten. Dies liegt daran, dass wir n Möglichkeiten haben eine Startposition auszusuchen. Danach haben wir $n-1$ Möglichkeiten für den nächsten zu besuchenden Knoten, danach $n-2$ etc. weil der Graph vollständig ist und wir nur einfache Pfade betrachten. Wenn wir dann beim Faktor $(n-k)$ angekommen sind haben wir einen Pfad der Länge k.

d)

Da Zykel in einem ungerichtetem Graph mindestens Länge 3 haben müssen, folgt sofort, dass für Graphen mit $n < 3$ Knoten genau 0 der gesuchten Zykel existieren. Sei nun also $n \geq 3$. Dann haben wir genau $n(n-1)(n-2)$ Möglichkeiten für einen Zykel der Länge 3 in einem vollständigem ungerichtetem Graphen mit n Knoten. Dies ist analog zu c), da wir n mögliche Startknoten haben, und der letzte Knoten wieder der Startknoten, die restlichen aber unterschiedlich sein müssen. Analog haben wir $n(n-1)(n-2)(n-3)$ Zykel der Länge 4, da wir beim nicht die Möglichkeit haben vor dem letzten Knoten zum Startknoten wiederzukehren ohne die Eigenschaft der paarweisen Verschiedenheit zu verletzen. Insgesamt haben wir also
$n(n-1)(n-2) + n(n-1)(n-2)(n-3) = n(n-1)(n-2)^2$ Zykel der Länge höchstens 4.

e)

i. Dies ist wahr, denn für $(u, v) \in \hat{E}$ gilt $(u, v) \in E \ \vee \ (u, v) \in E'$ und dann $(v, u) \in E \ \vee \ (v, u) \in E'$ also insgesamt $(v, u) \in \hat{E}$.

ii. Dies ist falsch. Betrachte $V = \{1, 2\}$ und $E = \{(1, 2)\}, \ E' = \{(2, 1)\}, \ \hat{E} = \{(1, 2), (2, 1)\}$. Dann ist $\hat{G}$ stark zsmhängend, aber weder G noch G^T stark zsmhängend.

iii. Dies ist wahr. Sei o.B.d.A. G stark zsmhängend, dann ist offensichtlich $E \subseteq \hat{E}$, und da die Graphen alle die gleiche Knotenmenge V haben damit auch $\hat{G}$ stark zsmhängend.

iv. Dies ist wahr. Sei o.B.d.A. G schwach zsmhängend, d.h. wir haben $\forall u, v \in V : (u, v) \in E \ \vee \ (v, u) \in E$ und damit $\forall u, v \in V : (u, v) \in E' \ \vee \ (v, u) \in E'$. Für beide Knotenmengen folgt dann aber im entsprechenden ungerichtetem Graph mit Knotenmenge K, dass $\forall u, v \in V : (u, v) \in K$. Grundlegend entsprechen umgredrehte Kanten genau den gleichen ungerichteten wie ihr "Urbild".

f)

Dies ist falsch. Betrachte den DAG mit Knotenmenge $V = \{a, b, c\}$ und entsprechenden Gewichten $w_a = 0, w_b = 1, w_c = 1$. Weiter haben wir die Kantenmenge $E = \{(b, a), (c, a)\}$. Sowohl b als auch c hängen von a ab, jedoch nicht untereinander. Die einzig sinvollen Pfade wären (a, b, c) oder (a, c, b). Je nachdem wie man die Definition nimmt ist nun entweder keiner von beiden ein kritischer Pfad (v_i muss von v_{i-1} abhängig sein) oder beide und damit nicht eindeutig. In beiden Fällen ist die Behauptung widerlegt.

Aufgabe 2

a)

Die asymptotische Laufzeit beträgt $\mathcal{O}(n^2)$. `DFS` wird grundsätzlich für jeden Knoten genau 1 mal aufgerufen. Die Methode prüft in jedem Aufruf ob vll. einer der Adjazenten Knoten noch nicht behandelt wurde und hat dabei konstante Kosten. Wenn `DFS` rekursiv aufgerufen wird, dann wird `DFS` halt nicht nocheinmal aus der obersten `completeDFS`-Schleife aufgerufen, welche aber trotzdem asymptotisch in n vergleiche benötigt. Früher oder später wird auf jedem Knoten `DFS` aufgerufen und eine for-Schleife mit linear vielen vergleichen ausgeführt.

b) Die vergleiche in den Schleifen fallen nun weg. Nach dem gleichen Argument wie in a), dass schlussendlich `DFS` genau 1 mal pro Knoten aufgerufen wird und konstant viele print aufrufe hat, folgt dann dass die asymptotische Laufzeit sich in $\mathcal{O}(n)$ befindet.

Aufgabe 3

W für WHITE, B für BLACK, G für GRAY. Beachte wir haben keinen Knoten 0, also 0. index in arrays ignoriert. Stack von links nach rechts ist von top zu bottom.

Phase 1: Nach 1. Iteration (0. Iteration ignoriert):
`color = [W, B, W, W, B, B, B, B, B, B, B, B], stack = [1,5,6,7,10,9,8,11,4]`

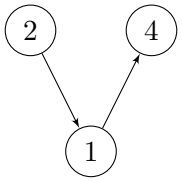
Nach 2. Iteration:
`color = [W, B, B, B, B, B, B, B, B, B, B, B], stack = [2,3,1,5,6,7,10,9,8,11,4]`

Phase 3: Nach 1. Aufruf:
`color = [W, W, B, B, W, W, W, W, W, W, W, W], stack = [3,1,5,6,7,10,9,8,11,4]`
`scc = [0,0,2,2,0,0,0,0,0,0,0,0]`

Nach 2. Aufruf:
`color = [W, B, B, B, W, B, B, B, B, B, B, B], stack = [5,6,7,10,9,8,11,4]`
`scc = [0,1,2,2,0,1,1,1,1,1,1,1]`

Nach 3. Aufruf:
`color = [W, B, B, B, B, B, B, B, B, B, B, B], stack = []`
`scc = [0,1,2,2,4,1,1,1,1,1,1,1]`

b)



Aufgabe 4

a)

topo(v)	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Knoten v	2	1	17	13	19	14	9	5	6	10	15	11	7	8	4	3	12	20	16	18

b)

```
1  void topoSort(List adj[n], int n, int &topo[n]) {
2      int color[n] = WHITE, topoNum = 0;
3      boolean allChildrenDone = true;
4      for (int v = 0; v < n; ++v) {
5          if (color[v] == WHITE) {
6              foreach (w in adj[v]) {
7                  if (color[w] == WHITE) {
8                      allChildrenDone = false;
9                      break;
10                 }
11             }
12
13             if (allChildrenDone) {
14                 color[v] = BLACK;
15                 topo[v] = ++topoNum;
16                 v = -1; // incremented after current cycle ends, so we start from 0
17             }
18
19             allChildrenDone = true;
20         }
21     }
22 }
```

Aufgabe 5

Knoten	A	B	C	D	E	F	G	H	I	J	K	L	M	N
est	16	11	10	8	13	3	6	6	0	4	0	0	0	0
critDep	E	G	D	F	C	M	J	L	-	N	-	-	-	-
eft	20	13	13	10	16	8	11	9	3	6	3	6	3	4

Reihenfolge Schwarzfärbung: N, J, G, B, M, F, K, D, I, C, L, H, E, A
Gesamtdauer: 20
Kritischer Pfad: M, F, D, C, E, A

Aufgabe 6

```
boolean isBipartit(List adj[n]) {
    int color[n] = WHITE;

    for (int v = 0; v < n; ++v) {
        if (color[v] == WHITE && findOddCycleDFS(adj, v, color, false))
            return false;
    }

    return true;
}

boolean findOddCycleDFS(List adj[n], int start, int &color[n], boolean odd) {
    color[start] = GRAY;
    foreach (next in adj[start]) {
        if (odd && color[next] == GRAY)
            return true;

        if (color[next] == WHITE)
            findOddCycleDFS(adj, next, color, !odd);
    }

    color[start] = BLACK;

    return false;
}
```

Der Algorithmus führt eine normale DFS durch und sucht dabei nach Zykeln ungerader Länge. Wenn wir während eines DFS auf einen grauen Knoten stossen, so haben wir einen Zykel im Graphen. Nebenbei merkt sich der Algorithmus die Parität der Länge des Pfades vom ursprünglichen Startpunkt. Somit können wir sofort erkennen wenn wir auf einen Zykel ungerader Länge treffen und dies returnen.