Karlsruher Institut für Technologie

Institut für Theoretische Informatik
T.T.-Prof. Dr. Thomas Bläsius
https://scale.iti.kit.edu

Zusatzaufgaben 02

Algorithmen I – Sommersemester 2022

Gesamtpunkte: 40

Aufgabe 1 - Des Graphen neue Kleider (10 Punkte)

Um einen gerichteten Graphen $G = (V, E)$ in Computerprogrammen darzustellen gibt es verschiedene Möglichkeiten. Die wohl simpelste Darstellung ist dabei die Kantenliste E. Aus der Kantenliste kann sehr einfach V berechnet werden:

$$V = \bigcup_{(u,v)\in E} \{u, v\}$$

Andere Operationen, wie die Nachbarschaft von $N(v) = \{u \mid (v, u) \in E\}$ von $v \in V$ oder den Grad $deg(v) = |N(v)|$ zu berechnen, sind in dieser Darstellung jedoch nicht effizient und somit nur in $\mathcal{O}(|E|)$ umsetzbar.
Deswegen werden Graphen häufig in einem *Adjazenzarray* dargestellt. Hierfür nehmen wir an, das wir die Knoten durchnummerieren können, also $V = \{0, \ldots, n-1\}$ gilt. Der Index von $v \in \{0, \ldots, n\}$ ist definiert durch:

$$I(v) = \begin{cases} 0 & \mid v = 0 \\ I(v-1) + |N(v-1)| & \mid \text{sonst} \end{cases}$$

Eine Adjazenzarray-Datenstruktur besteht aus zwei Arrays $\mathcal{V}, \mathcal{E}$. $\mathcal{V}$ speichert die Indizes $\mathcal{V} = \{I(0), I(1), \ldots, I(n)\}$. $\mathcal{E}$ speichert alle Nachbarn aller Knoten: $\mathcal{E} = N(v_0) \circ N(v_1) \circ \cdots \circ N(v_{n-1})$. Hierbei ist $\circ$ die Verkettung von Arrays (Beispiel: $\{1, 2\} \circ \{3, 4\} = \{1, 2, 3, 4\}$).

1. Gegeben sei $\mathcal{V} = \{0, 2, 4, 7, 8, 8, 10\}$ und $\mathcal{E} = \{1, 2, 0, 4, 1, 3, 5, 1, 2, 4\}$. Zeichne den zugehörigen Graphen. (1 Punkt)

2. Bei genauer Beobachtung fällt auf, dass $|\mathcal{V}| = |V| + 1$. In der Vorlesung haben wir Dummy- bzw. Sentinel-Elemente kennengelernt. Welchen Zweck erfüllen diese? Bei welchen Datenstrukturen aus der Vorlesung haben wir Sentinel-Elemente kennengelernt? Welchen Zweck erfüllt das Sentinel-Element im Adjazenzarray? (2 Punkte)

3. Wie kann in einem Adjazenzarray in $\mathcal{O}(1)$ der Grad eines Knotens v bestimmt werden? Wie können wir in $\mathcal{O}(deg(v))$ über $N(v)$ iterieren? (2 Punkte)

4. Gib einen Algorithmus in Pseudocode an, der ein Adjazenzarray in $\mathcal{O}(|V|+|E|)$ in eine Kantenliste konvertiert. Begründe, warum dein Algorithmus die Laufzeitanforderung erfüllt. Gegeben sei dazu die folgende Signatur:

$\textsc{arrayToList}(\mathcal{V}\colon [\mathbb{N}; n+1], \mathcal{E}\colon [\mathbb{N}; m])\colon \mathsf{List}\langle \mathbb{N} \times \mathbb{N} \rangle$

(2 Punkte)

5. Gib einen Algorithmus in Pseudocode an, der eine Kantenliste in ein Adjazenzarray in $\mathcal{O}(|V| + |E|)$ konvertiert. Begründe, warum dein Algorithmus die Laufzeitanforderung erfüllt. Gegeben sei dazu die folgende Signatur:

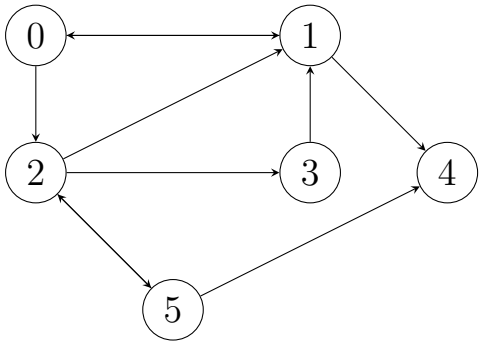
   - n: Anzahl der Knoten, $V = \{1, \ldots, n-1\}$
   - m: Anzahl der Kanten
   - L: Liste der Kanten

$\textsc{listToArray}(n\colon \mathbb{N}, m\colon \mathbb{N}, L\colon \mathsf{List}\langle \mathbb{N} \times \mathbb{N} \rangle)\colon ([\mathbb{N}; n+1], [\mathbb{N}; m])$

(3 Punkte)

Lösung 1

1. Der Graph, der durch $\mathcal{V}$ und $\mathcal{E}$ definiert ist:



2. Sentinel-Elemente erfüllen die Aufgabe, Invarianten aufrecht zu erhalten und Sonderfälle zu vermeiden.
   Wir haben Sentinel-Elemente bei folgenden Datenstrukturen aus der Vorlesung kennengelert:

   Liste: Als Dummy-Element, sodass selbst in einer leeren Liste $e.prev.next = e.next.prev = e$

   ab-Bäume: Bei ab-Bäume haben wir den „Unendlich-Trick“ kennengelernt. Dadurch existiert immer ein maximales Element, sodass wir immer ein valides Element bei `find()` zurückgeben können.

   Im Adjazenzarray bildet der letzte Eintrag des $\mathcal{V}$-Arrays ein Sentinel-Element, dadurch haben wir die Invariante für Knoten $v \in \mathbb{N}$:

   $$N(v) = \{\mathcal{E}[\mathcal{V}[v]], \dots \mathcal{E}[\mathcal{V}[v+1]-1]\}$$

   Zudem speichert $\mathcal{V}[|V|]$ die Anzahl an Kanten.

3. Der Grad eines Knotens kann wie folgt bestimmt werden:
   $deg(v) := \mathcal{V}[v+1] - \mathcal{V}[v]$
   Über die Nachbarschaft von v kann wie im folgenden Pseudocode iteriert werden:

```
...
v: ℕ = ...
for k ∈ {V[v], ..., V[v+1] − 1} do
    u: ℕ = E[k]
    ...
end
...
```

4.

```
arrayToList(𝒱: [ℕ; n+1], ℰ: [ℕ; m]): List⟨ℕ × ℕ⟩
    L : List⟨ℕ × ℕ⟩
    for v ∈ {0, ..., n−1} do
        for k ∈ 𝒱[v], ... 𝒱[v+1] − 1 do
            u : ℕ = ℰ[k]
            L.pushBack(v, u)
        end
    end
    return L
```

5.

```
listToArray(n: ℕ, m: ℕ, L: List⟨ℕ × ℕ⟩): ([ℕ; n+1], [ℕ; m])
    𝒱 : [ℕ; n+1] = ⟨0, ..., 0⟩
    ℰ : [ℕ; m] = ⟨0, ..., 0⟩
    for (v, u) ∈ L do
        𝒱[v+1] += 1
    end
    for i ∈ {1, ... n} do
        𝒱[i] += 𝒱[i−1]
    end
    vi : [ℕ; n] = ⟨0, ..., 0⟩
    for (v, u) ∈ L do
        ℰ[𝒱[v] + vi[v]] = u
        vi[v] := vi[v] + 1
    end
    return (𝒱, ℰ)
```

Aufgabe 2 - Datenstrukturen Zuordnen (15 Punkte)

Gegeben sind insgesamt 15 Szenarien und fünf Datenstrukturen (Array, Listen: Stack, FIFO, (allgemeine) doppelt-verkettete Listen und Hashtabellen). Deine

Aufgabe ist es nun, die Szenarien der passenden Datenstruktur zuzurodnen. Dabei gibt es pro Datenstruktur genau drei zugehörige Szenarien. Begründe kurz für jede Datenstruktur, warum Du genau diese Szenarien ausgewählt hast.

1. Für ein Programm soll eine Undo/Redo-Funktion entwickelt werden, bei dem die vergangenen Veränderungen gespeichert werden sollen.
2. Für einen Online-Shop soll für ein Produkt (gegeben durch Produktnummer) vor dem Verkauf überprüft werden, wie viele noch vorhanden sind.
3. In einer App gibt es einen Content-Feed, bei dem man nach rechts/ links swipen kann, um aktuellere/ältere Nachrichten zu erhalten.
4. Für eine Veranstaltung mit 5000 durchnummerierten Tickets soll am Einlass kontrolliert werden, welche Tickets bereits entwertet wurden.
5. Für eine Anwendung soll jeweils die letzte noch nicht gelesene Nachricht auf eurem Telefon anzeigt werden.
6. Es werden (unpriorisiert) E-Mails an einen Newsletter versendet.
7. Für eine Anwendung sollen Bewegungen zwischen Zugabteilen bei einem Zug beliebiger Länge simuliert werden. Es soll dabei gespeichert werden, wie viele Personen sich zu einem Zeitpunkt in einem Zugabteil befinden.
8. In einer Bar sind leider schon einige Menschen auffällig geworden und haben Hausverbot erhalten. Um dies festzuhalten wurden ihre den Vor- und Nachnamen und die Ausweisnummer des Personalausweises gepeichert. Für eine Person soll nun festgestellt werden können, ob sie schoneinmal Hausverbot erhalten hat.
9. Für ein Multiple-Choice Quiz mit 20 Fragen je 4 Antwortmöglichkeiten soll ausgegeben werden, wie viele Studenten die entsprechende Antwort ausgewählt haben.
10. Bei einem rundenbasierten Online-Game ohne feste Spieleranzahl soll die Spieler-Reihenfolge gespeichert werden.

11. Für eine neue Speedrun-Website soll es pro Spiel ein Leaderbord geben, in dem die 10 besten Spieler stehen sollen.

12. Für einen Music-Player soll eine Playlist abgespielt werden. Dabei soll es auch möglich sein, einen vergangenen Song anzuhören oder Songs zu skippen.

13. Zur Ausgabe eines Browserverlaufs sollen die zuletzt besuchten Websiten gespeichert werden.

14. Für eine passwortgeschützte Website sollen für Neukunden Profile angelegt werden. Bei späterem Einloggen soll überprüft werden, ob ein eingegebenes Passwort richtig war. Um Hacking zu vermeiden sollte das Passwort allerdings nicht gespeichert werden.

15. Ein Nutzer lädt Videos von einer Website herunter. Falls ein Video noch nicht vollständig heruntergeladen ist, soll erst sein Download abgeschlossen werden, bevor ein Video heruntergeladen wird, das erst danach ausgewählt wurde.

Lösung 2

- Array: 4,9,11
- Stack: 1,5,13
- FIFO: 6,10,15
- (allgemeine) doppelt-verkettete Listen: 3,7,12
- Hashtabelle: 2,8,14

Aufgabe 3 - Immer in Paaren laufen $(7\ \text{Punkte})$

Im Folgenden wollen wir in erwartet Linearzeit Paare von ganzen Zahlen in einem Array finden, die eine gewisse Eigenschaft erfüllen.

1. Beschreibe einen Algorithmus, der für eine gegeben Zahl k und ein Array A mit n Einträgen in erwartet $\mathcal{O}(n)$ Zeit entscheidet, ob es Indizes i, j gibt, sodass $A[i] \cdot A[j] = k$. (2 Punkte)

2. Beschreibe einen Algorithmus, der für eine gegebene Zahl k und ein Array A mit n Einträgen in erwartet $\mathcal{O}(n)$ Zeit entscheidet, ob es Indizes i, j gibt, sodass $\frac{2 \cdot A[i]}{3} + 4 \cdot A[j] + 5 = k$. (2 Punkte)

*. Finde ein generelles Verfahren, womit wir für eine gegebene Zahl $k \in \mathbb{Z}$, ein Array A mit n Einträgen und eine Funktion $f(x, y) : \mathbb{Z} \times \mathbb{Z} \to \mathbb{Z}$ in erwartet $\mathcal{O}(n)$ entscheiden können, ob es zwei Indizes i, j gibt, sodass $f(A[i], A[j]) = k$. Welche Anforderungen muss die Funktion f erfüllen? (3 Punkte)

Lösung 3

1. Wir legen eine Hashmap der Größe n an, welche ganze Zahlen enthält. Wir fügen nun die Elemente aus A ein, hierbei verwenden wir für einen Eintrag $a = A[l]$ den Schlüssel a zum Wert l.
   Anschließend überprüfen wir für jeden Eintrag $b = A[i]$ ob der es einen Eintrag j zum Schlüssel $\frac{k}{b}$ in der Hashmap gibt. Sollte dies der Fall sein, so ist i, j so ein gewünschtes Paar.
   Dieses Verfahren fügt n Elemente in erwartet $\Theta(n)$ in die Hashmap ein und fragen dann maximal n mal jeweils in erwartet $\Theta(1)$ einen Wert in der Hashmap ab, also liegt die Gesamtlaufzeit in erwartet $\Theta(n)$.

2. Wieder legen wir eine Hashmap der Größe n an, welche ganzen Zahlen enthält und fügen die Elemente aus A ein wie bei der vorherigen Teilaufgabe.
   Nun überprüfen wir für jeden Eintrag $b = A[l]$ ob es einen Eintrag j zum Schlüssel $\frac{3}{2}(k - 5 - 4b)$ gibt. Sollte dies der Fall sein, so ist i, j ein gesuchtes Paar.
   Auch hier ist die Laufzeitabschätzung die gleiche, da wir die Umkehrfunktion in $\Theta(1)$ berechnen können.

*. Damit das Verfahren aus den vorherigen Teilaufgaben funktioniert, muss die Funktion f auf einem der beiden Eingaben invertierbar sein, also sei $c \in \mathbb{Z}$ fest, dann muss entweder $f(x, c)$ oder $f(c, x)$ invertierbar sein, also $f_c^{-1}(k) = \{x \in \mathbb{Z} | f(x, c) = k\}$ in $\Theta(1)$ berechenbar sein. Zusätzlich muss $|f_c^{-1}| \in \Theta(1)$ liegen, damit wir in erwartet $\Theta(n)$ die Werte berechnen können

Dann können wir das Verfahren aus den vorherigen Teilaufgaben anwenden. Wir legen eine Hashmap der Größe n an, welche ganze Zahlen enthält. Wir fügen die Elemente aus A ein. Für einen Eintrag $b = A[l]$ fügen wir den Wert l zum Schlüssel b ein.
Nun überprüfen wir für jeden Eintrag $c = A[i]$, ob es einen Wert j zu einem Schlüssel aus $f_c^{-1}(k)$ in der Hashmap gibt. Ist dies der Fall, dann ist i, j ein solches gewünschtes Paar.
Wie in der vorherigen Teilaufgaben fügen wir n Elemente in eine Hashmap in erwartet $\Theta(n)$ ein. Anschließend überprüfen wir für maximal n Elemente, ob es einen entsprechenden Partner gibt. Da es für jedes Element nur $\Theta(1)$ Partner gibt und wir diese in $\Theta(1)$ berechnen können und jedes dieser potentiellen Partner in erwartet $\Theta(1)$ abfragen können, liegt die ganze Abfrage in erwartet $\Theta(n)$. Damit liegt unser Verfahren insgesamt in erwartet $\Theta(n)$

Aufgabe 4 - Auf heißer Spur (8 Punkte)

Der brilliante und wachsame Superbösewicht Dr. Meta dachte, mit seinen Säuberungsaktionen sämtliche Maulwürfe unter seinen Mitarbeitern entfernt zu haben. Doch weit gefehlt!
Er hat herausbekommen, dass nicht nur der Eindringling, der in sein Labor eingebrochen ist, sondern auch dessen Komplizen von seiner Erzfeindin Theresa Trivial angeheuert wurden. Kurzerhand macht sich Dr. Meta auf, herauszufinden, was Theresa im Schilde führt. Mit seinen verlässlichsten Spitzeln zusammen infiltriert er die Burg von Theresa Trivial, in der es vor Spionen wimmelt.
Die Spitzel von Dr. Meta positionieren sich in der Burg und notieren jedes Mal, wenn sie einen verdächtigen Spion sehen. Somit entsteht eine Liste mit Tripeln der Form (Zeit, Ort, Name). Doch die Spione sind äußerst gewieft: Werden sie nach ihrem Namen gefragt, geben sie jedem Spitzel von Dr. Meta eine andere Antwort. Somit kennen zwei Spitzel den gleichen Spion unter unterschiedlichen Namen.
Dr. Meta will nun herausfinden, welche Decknamen zu welchem Spion gehören. Gegeben hat er dazu eine Liste L der Länge n, die Tripel der Form (Zeit, Ort, Name) enthält. Ein Tripel (t, p, d) beschreibt dabei, dass der Spion mit Decknamen d zum Zeitpunkt t am Ort p gesichtet wurde. Gibt es zudem einen

weiteren Eintrag (t, p, d') in L, so weiß Dr. Meta, dass d und d' Decknamen des gleichen Spions sind.
Wir wollen nun einen Algorithmus entwickeln, der die Decknamen in L den Spionen zuordnet.

1. Beschreibe, wie eine Liste sämtlicher Decknamen in L erstellt werden kann, die keine Duplikate enthält. Dein Algorithmus soll dabei eine asymptotische Laufzeit haben, die besser als $\Theta(n^2)$ ist. (2 Punkte)

2. Welche Datenstruktur eignet sich am Besten, um die Decknamen in L zu verwalten? Begründe deine Wahl. (1 Punkt)

3. Beschreibe einen Algorithmus, der für alle gesichteten Spione die Menge ihrer Decknamen bestimmt. (3 Punkte)

4. Gib die asymptotische Laufzeit deines Algorithmus aus Teilaufgabe 3 an und begründe deine Antwort. (2 Punkte)

Lösung 4

1. Wir können zunächst in Linearzeit alle Decknamen in L in einer Liste `names` sammeln und anschließend in $\Theta(n \log(n))$ sortieren. Nun können wir ein Mal über `names` iterieren und dabei Duplikate löschen. Dazu merken wir uns stets das Vorgänger-Element und löschen das aktuelle Element, wenn beide übereinstimmen. Insgesamt benötigen wir somit Zeit in $\Theta(n \log(n))$.

2. Da wir die Menge der Decknamen in disjunkte Teilmengen zerlegen wollen, eignet sich die UnionFind-Datenstruktur für unsere Zwecke. Sie erlaubt es, die Teilmenge der zugehörigen Decknamen für jeden Spion zu verwalten.

3. Wir erstellen zunächst eine duplikatfreie Liste `names` aller Decknamen in L wie in Teilaufgabe 1 beschrieben. Nun legen wir eine Hashmap `sightings` und eine UnionFind-Datenstruktur `spies` an. `sightings` verwendet dabei Tupel der Form (Zeit, Ort) als Schlüssel, `spies` wird auf `names` initialisiert.
   Nun iterieren wir über L. Für jedes Tupel (t, p, n) überprüfen wir, ob

bereits ein Eintrag in `sightings` für den Schlüssel (t, p) existiert. Wenn nicht, dann fügen wir n als Eintrag für den Schlüssel (t, p) ein. Ansonsten existiert bereits ein Eintrag n'. Dann rufen wir `union` für die Elemente n und n' auf `spies` auf.
Zuletzt können wir die Repräsentanten der Teilmengen in `spies` bestimmen, also diejenigen Elemente in `names`, für die `find` sie selbst zurückliefert. Dann legen wir für jeden dieser Repräsentanten eine Liste an und fügen jeden Decknamen in die Liste ein, die zu seinem Repräsentanten in `spies` gehört. Somit gehört jede Liste von Decknamen zu genau einem Spion.

4. Wie bereits in Teilaufgabe 1 beschrieben, benötigt das Erstellen von `names` Zeit in $\Theta(n \log(n))$. Das anschließende Iterieren über L benötigt Zeit in erwartet $\Theta(n \cdot \log^*(n))$, denn: Wir müssen jedes Element in `sightings` suchen. Dies geschieht in erwartet konstanter Zeit. Zudem rufen wir bis zu $n - 1$ Mal `union` mit Zeitbedarf $\Theta(\log^*(n))$ auf. Für das erstellen der Listen, die unsere Ausgabe bilden, müssen wir für jedes Element in `names` `find` aufrufen. Dies benötigt Zeit in $\Theta(n \log^*(n))$. Insgesamt hat unser Algorithmus einen Zeitbedarf in erwartet $\Theta(n \log(n))$.