Übungsaufgabe

Graphenrepräsentationen: Adjazenzlisten vs. Adjazenzmatrizen, dynamische Graphen

Universität: Technische Universität Berlin
Kurs/Modul: Algorithmen und Datenstrukturen
Erstellungsdatum: September 6, 2025

Aufgabe 1: Grundlagen der Graphenrepräsentationen

Gegeben sei ein ungerichteter Graph $G = (V, E)$ mit $V = \{v_1, v_2, v_3, v_4\}$ und $E = \{\{v_1, v_2\}, \{v_1, v_3\}, \{v_2, v_4\}\}$
Die Aufgaben befassen sich mit Adjazenzlisten, Adjazenzmatrix und dynamischen Graphen.

a) Geben Sie die Adjazenzliste von G an (eine Liste pro Knoten). Geben Sie außerdem die Adjazenzmatrix A von G an, wobei $A_{ij} = 1$ gilt, falls $\{v_i, v_j\} \in E$.

b) Diskutieren Sie kurz Vor- und Nachteile der beiden Repräsentationen im Hinblick auf Speicherbedarf und typische Graph-Operationen (addEdge(u,v), removeEdge(u,v), isAdjacent(u,v), neighbours(u)).

c) Angenommen, ein weiterer Knoten v_5 wird hinzugefügt und die Kante $\{v_3, v_5\}$ eingefügt. Beschreiben Sie, wie sich sowohl die Adjazenzliste als auch die Adjazenzmatrix verändern würden. Vermeiden Sie dabei unnötige Details, geben Sie die wichtigsten Änderungen an.

Aufgabe 2: Dynamische Graphen – Operationen und Kosten

Gehen Sie davon aus, dass der Graph ungerichtet bleibt. Unterscheiden Sie die Repräsentationen Adjazenzliste und Adjazenzmatrix bzgl. dynamischer Updates.

a) Geben Sie Pseudocode in Verbatim-Umgebungen an, um folgende Operationen für die Adjazenzliste zu implementieren:

```
function addEdgeList(u,v):
    if v not in L[u]:
        append v to L[u]
    if u not in L[v]:
        append u to L[v]

function removeEdgeList(u,v):
    remove v from L[u]
    remove u from L[v]
```

b) Geben Sie Pseudocode in Verbatim-Umgebungen an, um folgende Operationen für die Adjazenzmatrix zu implementieren:

```
function addEdgeMat(M,u,v):
    M[u][v] = 1
    M[v][u] = 1
end

function removeEdgeMat(M,u,v):
    M[u][v] = 0
    M[v][u] = 0
end
```

c) Diskutieren Sie grob die Zeitkosten der obigen Operationen in Abhängigkeit von $n = |V|$ und $m = |E|$ für beide Repräsentationen. Berücksichtigen Sie Worst-Case-Szenarien und typische Implementierungsdetails.

Aufgabe 3: Anwendungen, Reflexion und Auswahl der Repräsentation

Untersuchen Sie praxisnahe Aspekte zur Wahl der Repräsentation in verschiedenen Szenarien.

a) Welches Repräsentationsmodell würden Sie bevorzugen, wenn Sie häufig Nachbarn eines beliebigen Knotens abfragen möchten? Erläutern Sie Ihre Begründung.

b) Welche Repräsentation eignet sich besser für Szenarien mit vielen Edge-Updates (Hinzufügen/Löschen von Kanten) in einem großen, aber eher spärlich besetzten Graphen? Begründen Sie Ihre Wahl.

c) Fassen Sie zwei bis drei zentrale Vor- und Nachteile der Adjazenzliste gegenüber der Adjazenzmatrix zusammen (Speicher, Laufzeit, Implementierbarkeit) und nennen Sie typische Einsatzkontexte.

Lösungen

Aufgabe 1: Grundlagen der Graphenrepräsentationen

Gegeben sei ein ungerichteter Graph $G = (V, E)$ mit $V = \{v_1, v_2, v_3, v_4\}$ und $E = \{\{v_1, v_2\}, \{v_1, v_3\}, \{v_2, v_4\}\}$
Die Aufgaben befassen sich mit Adjazenzlisten, Adjazenzmatrix und dynamischen Graphen.

Lösung zu Aufgabe 1

a) Die Adjazenzliste und die Adjazenzmatrix von G sind:

- Adjazenzliste

$$L(v_1) = \{v_2, v_3\}, \quad L(v_2) = \{v_1, v_4\}, \quad L(v_3) = \{v_1\}, \quad L(v_4) = \{v_2\}.$$

- Adjazenzmatrix

$$A = \begin{bmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}.$$

b) Vor- und Nachteile der Repräsentationen (im Hinblick auf Speicherbedarf und typische Graph-Operationen)

- Speicherbedarf:
  - Adjazenzliste: $O(n + m)$ Speicherbedarf (n = |V|, m = |E|).
  - Adjazenzmatrix: $O(n^2)$ Speicherbedarf.
- Typische Graph-Operationen:
  - addEdge(u,v):
    - Liste: Worst-Case $O(\deg(u) + \deg(v))$ (bei Bedarf Duplikate vermeiden; meist durch Membership-Checks).
    - Matrix: $O(1)$.
  - removeEdge(u,v):
    - Liste: Worst-Case $O(\deg(u) + \deg(v))$.
    - Matrix: $O(1)$.
  - isAdjacent(u,v):
    - Liste: $O(\deg(u))$ bzw. $O(\deg(v))$ (je nachdem, in welcher Liste gesucht wird).
    - Matrix: $O(1)$.
  - neighbours(u):
    - Liste: $O(\deg(u))$ (direkter Zugriff auf Nachbarn).
    - Matrix: $O(n)$ (Zeile/Spalte durchsuchen).

c) Angenommen, ein weiterer Knoten v_5 wird hinzugefügt und die Kante $\{v_3, v_5\}$ eingefügt. Beschreiben Sie, wie sich sowohl die Adjazenzliste als auch die Adjazenzmatrix verändern würden.

- Adjazenzliste:
$$L(v_3) \leftarrow \{v_1, v_5\}, \quad L(v_5) \leftarrow \{v_3\}.$$
Andere Listen unverändert.

- Adjazenzmatrix: Es wird eine neue Zeile/Spalte für v_5 eingefügt. Setze $A_{3,5} = A_{5,3} = 1$; alle übrigen Einträge bleiben unverändert. In der 5×5-Matrix erhält man damit eine neue Randzeile/-spalte mit dem einzigen 1-Eintrag an Position (3,5) bzw. (5,3).

Aufgabe 2: Dynamische Graphen – Operationen und Kosten

Gehen Sie davon aus, dass der Graph ungerichtet bleibt. Unterscheiden Sie die Repräsentationen Adjazenzliste und Adjazenzmatrix bzgl. dynamischer Updates.

Lösung zu Aufgabe 2

a) Pseudocode für Adjazenzliste (Verbatim-Umgebung):

```
function addEdgeList(u,v):
    if v not in L[u]:
        append v to L[u]
    if u not in L[v]:
        append u to L[v]

function removeEdgeList(u,v):
    remove v from L[u]
    remove u from L[v]
```

b) Pseudocode für Adjazenzmatrix (Verbatim-Umgebung):

```
function addEdgeMat(M,u,v):
    M[u][v] = 1
    M[v][u] = 1
end

function removeEdgeMat(M,u,v):
    M[u][v] = 0
    M[v][u] = 0
end
```

c) Grobe Diskussions der Zeitkosten

- Adjazenzliste:
  - addEdgeList(u,v): $O(\deg(u)+\deg(v))$ in Worst-Case (wegen der Membership-Checks).
  - removeEdgeList(u,v): $O(\deg(u) + \deg(v))$ (Suche + Entfernen in beiden Listen).
  - Speicherbedarf: $O(n + m)$.
- Adjazenzmatrix:
  - addEdgeMat(M,u,v), removeEdgeMat(M,u,v): $O(1)$ pro Update.
  - Speicherbedarf: $O(n^2)$ (unabhängig von m).

Hinweis: Falls dynamische Erweiterungen des Vertex-Sets erfolgen (Erhöhung von n), muss die Matrix ggf. neu alloziert bzw. vergrößert werden, was in amortisierter Form Kosten von $O(n^2)$ verursachen kann. Als Folge davon eignet sich die Matrix besser für statische oder dichte Graphen, die Liste besser für dynamische oder spärliche Graphen.

Aufgabe 3: Anwendungen, Reflexion und Auswahl der Repräsentation

Lösung zu Aufgabe 3

a) Welches Repräsentationsmodell würden Sie bevorzugen, wenn Sie häufig Nachbarn eines beliebigen Knotens abfragen möchten? Erläutern Sie Ihre Begründung.

- Bevorzugte Repräsentation: Adjazenzliste.
- Begründung: Die Nachbarn eines Knotens u liegen direkt in der Liste $L(u)$ bzw. in der entsprechenden Struktur. Das Abrufen der Nachbarn erfolgt in Zeit $O(\deg(u))$ statt der Notwendigkeit, eine ganze Zeile der Matrix zu scannen ($O(n)$ im Worst-Case). Gerade bei spärlichen Graphen ist die Liste speicherschonender, und der Zugriff auf die Nachbarn ist effizienter, da keine teuren Scanoperationen der gesamten Matrix nötig sind.

b) Welche Repräsentation eignet sich besser für Szenarien mit vielen Edge-Updates (Hinzufügen/Löschen von Kanten) in einem großen, aber eher spärlich besetzten Graphen? Begründen Sie Ihre Wahl.

- Bevorzugte Repräsentation: Adjazenzliste.
- Begründung: Updates betreffen vor allem das Anlegen/Entfernen von Kanten. In einer Liste erfolgen solche Operationen in typischer Weise proportional zur Betroffenheit der Aktionen (in der Regel $O(\deg(u))$ zum Entfernen/Überprüfen von Existenz). Die Matrix dagegen erfordert konstanten Zeitaufwand pro Update, hat aber hohen Speicheraufwand ($O(n^2)$) und ist weniger flexibel bei dynamischen Größenänderungen. Für große, spärliche Graphen bietet die Liste daher deutliche Vorteile.

c) Fassen Sie zwei bis drei zentrale Vor- und Nachteile der Adjazenzliste gegenüber der Adjazenzmatrix zusammen (Speicher, Laufzeit, Implementierbarkeit) und nennen Sie typische Einsatzkontexte.

- Vorteile der Adjazenzliste:
  - Speichereffizienz bei sparsamen Graphen ($O(n+m)$ statt $O(n^2)$).
  - Schneller Zugriff auf die Nachbarn eines Knotens.
  - Bessere Unterstützung dynamischer Updates in großen, spärlichen Graphen.
- Nachteile der Adjazenzliste:
  - IsAdjacent(u,v) kann langsamer sein (notwendige Suche in einer Liste, $O(\deg(u))$).
  - Kontrolle auf Duplikate erfordert zusätzliche Checks.
- Typische Einsatzkontexte:
  - Adjazenzliste: große, spärliche Graphen, dynamische Kantenupdates, häufige Nachbarn-Abfragen.
  - Adjazenzmatrix: mittelgroße bis dichte Graphen, häufige IsAdjacent-Anfragen, geringe Updates, oder wenn eine feste maximale Graphgröße gegeben ist.