# Lernzettel

Graphenrepräsentationen: Adjazenzlisten vs. Adjazenzmatrizen, dynamische Graphen

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

**Lernzettel: Graphenrepräsentationen: Adjazenzlisten vs. Adjazenzmatrizen, dynamische Graphen**

**(1) Grundbegriffe**
Ein Graph G besteht aus einer Menge von Knoten $V = \{v_1, \ldots, v_n\}$ und einer Menge von Ecken $E \subseteq V \times V$. Graphen können - gerichtet oder ungerichtet sein, - einfach oder mehrstufig (Mehrfachkanten) auftreten, - gewichtet oder ungewichtet sein.
Ziel dieses Abschnitts ist die Gegenüberstellung zweier Repräsentationen: Adjazenzlisten und Adjazenzmatrizen, sowie der Umgang mit dynamischen Graphen.

**(2) Adjazenzlisten**
Definition: Für jeden Knoten v  V wird eine Liste N(v) der Nachbarn definiert:

$$N(v) = \{\, u \in V \mid (v, u) \in E \,\}.$$

Bei gerichteten Graphen enthält N(v) alle Knoten u, zu denen eine Kante (v,u) existiert.

**Eigenschaften** - Speicherbedarf: $O(n + m)$, mit m = |E|. - Nachbarschaftsauflistung: Durchlaufen von N(v) liefert alle Nachbarn in Zeit $O(|N(v)|)$. - Zugriff auf einen konkreten Nachbarn: $O(|N(v)|)$ im Worst-Case, falls N(v) als einfache Liste implementiert ist. - Kantenprüfung (v,w vorhanden?): $O(|N(v)|)$ im Worst-Case. - Einfügen einer Kante (v,w): Falls Duplikate vermieden werden sollen, muss ggf. N(v) nach u durchsucht werden, ansonsten amortisiert $O(1)$ bis $O(|N(v)|)$. - Entfernen einer Kante (v,w): $O(|N(v)|)$ Zeit.

**(3) Adjazenzmatrizen**
Definition: Für n=|V| existiert eine Matrix M  0,1$^{nn} mit M[v, w] = 1$ genau dann, wenn (v,w)E (bei gerichteten Graphen; bei ungerichteten Graphen muss $M[v, w] = M[w, v]$).

**Eigenschaften** - Speicherbedarf: $O(n^2)$. - Zugriff auf eine Kante: $M[v, w]$ direkt in $O(1)$. - Nachbarschaftsauflistung aus einer Zeile von v: $N(v) = \{\, u \mid M[v, u] = 1 \,\}$ in $O(n)$. - Kantenprüfung: $O(1)$ (durch Abfrage von $M[v, w]$). - Einfügen/Entfernen einer Kante: $O(1)$ (falls keine Neuanordnung notwendig). - Vorteil bei dicht besetzten Graphen: verringert sich der Aufwand beim Nachbarschaftszugriff nicht linear zur Knotenzahl.

**(4) Dynamische Graphen**
Ziel: Graphen, bei denen Knoten und Kanten häufig eingefügt/entfernt werden.

**Repräsentationen im dynamischen Setting** - Adjazenzlisten mit dynamischen Strukturen (z. B. Hash-Sets pro Vertex oder verkettete Listen) bieten amortisierte $O(1)$ für Edge-Inserts/Deletes, vorausgesetzt Duplikate werden vermieden. Nachbarschaftsiteration bleibt $O(|N(v)|)$. - Adjazenzmatrizen erfordern meist Neustrukturierung der gesamten Matrix bei Zuwachs der Knotenzahl oder müssen in Blöcken verwaltet werden; das führt zu höheren Kosten (oft $O(n^2)$ für Reallokation oder Resizing). - Orientierungsunterschiede: Bei gerichteten Graphen bleibt die Richtung wichtig; bei ungerichteten Graphen müssen Einträge symmetrisch geführt werden.

**Operationen (dynamisch, grob)** - Edge-Insertion: Liste - durchschnittlich $O(1)$; Matrix - $O(1)$ bis übersetzt, sofern Platz vorhanden, sonst Resize-Kosten. - Edge-Deletion: Liste - $O(|N(v)|)$ (duplizierte Prüfung möglich); Matrix - $O(1)$ sofern Edge existiert. - Edge-Abfrage:

Liste - $O(|N(v)|)$; Matrix - $O(1)$. - Skalierung der Knotenzahl: Liste - meist effizienter; Matrix - teurer, benötigt Resize.

**(5) Vergleich und Kriterien f00fcr die Wahl**
- Sparsity-Kriterium: Verwende Adjazenzlisten bei sparsamen Graphen ($m \ll n^2$); Adjazenzmatrizen bei dichten Graphen ($m \approx n^2$). - Zugriff auf Kante vs. Nachbarn: Matrix liefert schneller Edge-Checks; Listen liefern schnellere Nachbarschaftsiteration, besonders wenn $\deg(v)$ klein ist. - Speicherbedarf: Liste $O(n+m)$ vs Matrix $O(n^2)$; bei großen n bevorzugen Listen. - Dynamik: Bei vielen Edge-Updates bevorzugt Listen (mit geeigneten Strukturen); Matrizen erfordern teure Resizes. - Gewichtete Graphen: Unterscheiden sich in der Repräsentation nicht grundlegend, jedoch Speicherbedarf kann etwas ansteigen, wenn Gewichte zusätzlich gespeichert werden müssen.

**(6) Beispielhafte Formeln**
- Nachbarschaftsmenge von v: $N(v) = \{u \in V \mid (v,u) \in E\}$.
- Speicherbedarf bei Liste: $S_{\text{list}} = O(n+m)$.
- Speicherbedarf bei Matrix: $S_{\text{matrix}} = O(n^2)$.
- Zeit für Nachbarschaftsiteration in Liste: $T_{\text{list}}(v) = O(|N(v)|)$.
- Zeit für Nachbarschaftsiteration in Matrix: $T_{\text{matrix}}(v) = O(n)$.
- Zeit für Edge-Check: Liste $O(|N(v)|)$, Matrix $O(1)$.

**(7) Kleines Beispiel (Textual)**
Für einen ungerichteten Graphen G mit V={a,b,c,d,e} und Kanten (a,b),(a,c),(b,d),(c,e) gilt: - Adjazenzliste: N(a)=b,c, N(b)=a,d, N(c)=a,e, N(d)=b, N(e)=c. - Adjazenzmatrix: Zeile a enthält 1en an Spalten b und c, weitere Nullen.

**(8) Fazit**
- Wähle Adjazenzlisten für knappe Graphen und häufige dynamische Updates. - Wähle Adjazenzmatrix bei dichten Graphen und schnellen Edge-Checks, sofern der Speicher kein begrenzender Faktor ist. - Für dynamische Graphen sind Listen mit passenden Strukturen oft die praktikablere Wahl.