Inhaltsverzeichnis

[1. Abstrakter Datentyp Gerichteter Graph 2](#_Toc24286884)

[1.1. Lösungsidee 2](#_Toc24286885)

[1.1.1. Liste 2](#_Toc24286886)

[1.1.2. Matrix 2](#_Toc24286887)

[1.2. Code 3](#_Toc24286888)

[1.3. Testfälle 3](#_Toc24286889)

[2. Topologisches sortieren 4](#_Toc24286890)

[2.1. Lösungsidee 4](#_Toc24286891)

[2.2. Code 4](#_Toc24286892)

[2.3. Testfälle 4](#_Toc24286893)

# Abstrakter Datentyp Gerichteter Graph

## Lösungsidee

### Liste

Um einen gerichteten Graphen in einer Listenstruktur darstellen zu können, benötigt es eine Liste, in der alle Nodes mit ihrer Payload gespeichert werden. Zusätzlich wird für jede „node“ eine Liste benötigt, in der alle „edges“ zu einer anderen „node" gespeichert werden. In den „edge-nodes“ wird allerdings nicht die Payload der „target-node“ gespeichert, sondern nur eine Referenz auf den „node“. Mit dieser Methode erspart man sich bei diversen Operationen am Graphen einige „strcmp“ - Vergleiche.

Beim Einfügen einer „edge“ wird geprüft, ob diese bereits existiert. Wenn nicht, dann wird sie an die „edge-list“ der entsprechenden „node“ angehängt.

Beim Entfernen einer „node“ werden zuvor alle „edges“ entfernt, die auf diese Node verweisen. Im Anschluss kann die „edge-list“ der „node“ sowie die „node“ selbst entfernt bzw. freigegeben werden.

Beim Einfügen und Entfernen von „nodes“ und „edges“ werden Fehler (wie z.b.: Fehlen der angegeben „target-node“) durch eine Fehlermeldung abgefangen.

🡪 Topologische Sortierung ist Teil der Graph-List-Implementierung (Näheres siehe unter Punkt 2)

### Matrix

Die Matrix-Implementation wird mit einer Struktur (Dyn. Boolean Array, Dyn. String Array und Int ElemCount) umgesetzt. Diese Struktur ist notwendig, um einen Abstrakten Datentyp zu erstellen). Durch die Adressarithmetik von C ist es möglich die Elemente eines dynamisch allokierten Feldes ident zu einem traditionellen Array anzusprechen. Die Anzahl der Nodes wird immer in ElemCount aktualisiert.

🡪 Krankheitsbedingt wurde das dynamische Allokieren nicht zu 100% fertig implementiert. Dadurch auch Abzüge bei der Implementation ;)

Es wurde eine Höchstgrenze an Elementen mit einer Konstante gesetzt. Wird diese überschritten, so müsste das Feld mit dem Befehl „realloc“ neu allokiert werden. Einfügen und Entfernen von Edges funktioniert jedoch einwandfrei. Löschen von Nodes bezieht sich auf die Reallokation, um den nicht benutzten Speicherbereich wieder freizugeben.

## Code

|  |
| --- |
|  |

## Testfälle

# Topologisches sortieren

## Lösungsidee

Um das Topologische Sortieren umzusetzen, wird die Listen – Implementation des Graphen verwendet. Grundsätzlich wird immer die Node gesucht, die am wenigsten oft in einer Edge referenziert wird. Wurde diese Node gefunden, so kann sie ausgegeben werden und aus dem Graphen samt aller Edges die von ihr ausgehen entfernt werden. Dieser Vorgang wird solange wiederholt, bis sich keine Node mehr im Graphen befindet (sprich graph\_list == NULL). Zu diesem Zeitpunkt wurden alle Nodes in sortierter Reihenfolge am Terminal ausgegeben.

Der Sinn hinter dem topologischen Sortieren ist es, dass die Nodes so angeordnet werden, dass alle Pfeile nach rechts zeigen und es keine rückläufigen Pfeile nach links gibt.

## Code

|  |
| --- |
|  |

## Testfälle