Graphen

Einführung

In diesem Praktikum verbinden Sie die Knoten eines Graphen, die sie seit dem vorherigen Praktikum anlegen können, mit Kanten. Sie verarabeiten weiterhin Eingabedaten von der Standardeingabe und geben das Graphviz[1]-kompatible Ergebnis über die Standardausgabe aus. Damit können Sie Ihr Ergebnis per *Pipe* mit `neato` weiterverarbeiten und sich den erzeugten Graphen visualisieren lassen.

Adjazenzmatrix

Zur Verwaltung der Verknüpfungen im Graphen – also der Kanten – sollten Sie in Ihrer Graphklasse eine Adjazenzmatrix anlegen. Auf diese Weise erleichtern Sie sich in diesem Praktikum die Abfrage, ob zwei Knoten durch eine Kante verbunden sind. Auch die Graph-Algorithmen im nächsten Praktikum sollten sich so etwas leichter implementieren lassen als mit erine Inzidnezmatrix oder einer Kantentabelle.
Als Vorbereitung auf das nächste Praktikum sollten Sie schon in diesem Praktikum Werte vom Typ `float` in der Adjazenzmatrix speichern. Für dieses Praktikum wäre `bool` zwar ausreichend, um Verbindungen anzugeben, dann müssten Sie im nächsten Praktikum allerdings die Logik, die Sie in diesem Praktikum entwickeln, wieder umbauen, um Kantengewichte zu unterstützen. Arbeiten Sie schon jetzt mit `float`, werden Sie den Code voraussichtlich lediglich ergänzen.
Wenn Sie `float` verwenden, können Sie *verbunden* mit dem Wert `1.0` ausdrücken und *nicht verbunden* mit dem Wert Unendlich. Diesen stellt Ihnen C++ mit dem Ausdruck `std::numeric_limits<float>::infinity()`[2] zur Verfügung. Um diesen Ausdruck im Code zu verwenden, binden Sie den Header `limits` ein.

Kommandozeilenparameter

In diesem Praktikum implementieren Sie eine einfache Behandlung von Kommandozeilenparametern. Durch Kommandozeilenparameter lässt sich z.B. das Verhalten von Applikationen steuern. Sie haben Kommandozeilenprameter in GIP1 bereits kennengelernt, als Sie mit dem `gcc` Ihre eigenen Programme kompiliert haben: Mit dem Parameter `-o` haben Sie den Namen der zu erstellenden, ausführbaren Datei angegeben.
Ihre `main`-Funktion

```
int main(int argc, char** argv)
```

ermöglicht den Zugriff auf die Kommandozeilenparameter über `argv` – genauer auf die einzelnen Elemente der Befehlszeile, mit der Sie Ihr Programm aufgerufen haben. `argc` gibt dabei die Anzahl dieser Elemente an.
Für den folgenden Aufruf des `gcc`,

```
gcc -o main main.c
```

den Sie so oder so ähnlich bereits verwendet haben, ergeben sich folgende Werte der Parameter:

- `argc`: `4`
- `argv[0]`: `gcc`
- `argv[1]`: `-o`
- `argv[2]`: `main`
- `argv[3]`: `main.c`

Getter-Methoden

Um die Vorteile von Kapselung und Zugriffsschutz voll auszunutzen, sind Membervariablen von Klassen in aller Regel *private*. *Getter*-Methoden, die Ihnen lediglich den Wert einer solchen Membervariablen zurückgeben, unterwandern dies. Legen Sie daher nicht automatisch für jede private Membervariable einen Getter an. Meist sind diese überflüssig und das gewünschte Verhalten lässt sich besser verpacken. Gleiches gilt sinngemäß für *Setter*.
In diesem Praktikum sollten Sie davon eine Ausnahme machen. Zur Ausgabe von Kanten im Graphviz-kompatiblen Format benötigen Sie die Namen der beteiligten Knoten. In diesem Fall wiegt der Nutzen eines Getters deutlich dessen prinzipiellen Nachteile auf. Anderenfalls müssten Sie einigen Aufwand treiben, um die Knoten-Namen getrennt von den Knoten zu verwalten. Dieser Aufwand ist im Rahmen dieses Praktikums nicht gerechtfertigt.

[1] `https://www.graphviz.org`
[2] `https://en.cppreference.com/w/cpp/types/numeric_limits/infinity`

EINSTIEG

1. Adjazenzmatrix

Beginnen Sie mit einer Hello-World-Applikation.

In dieser Aufgabe legen Sie in einer globalen Variable eine Adjazenzmatrix an, legen Kanten an und prüfen, ob Graph-Knoten verbunden sind.

(a) Legen Sie eine globale Variable vom Typ `std::vector<std::vector<float>>` an. Diese fungiert als Adjazenzmatrix. Initialisieren Sie diese Variable so, dass die nötigen Einträge für einen Graphen mit vier Knoten enthalten sind. Initialisieren Sie alle Einträge auf `std::numeric_limits<float>::infinity()`. Dieser Wert entspricht *nicht verbunden*.

(b) Geben Sie in Ihrer `main`-Funktion den Inhalt der Adjazenzmatrix mithilfe zweier verschachtelter `for`-Schleifen aus.

(c) Legen Sie eine Funktion `Connect` an, die zwei per Index angegebene Knoten in der globalen Adjazenzmatrix mit einer ungerichteten Kante verbindet. Verwenden Sie den Wert `1.0` für *verbunden*. Der Knoten-Index beginnt bei `0`.
Legen Sie selbst fest, ob die Zeilen der Adjazenzmatrix den Start- oder den Endvertex der Kante bezeichnen. Die Spalten bezeichnen den jeweils anderen Vertex.

(d) Legen Sie eine Funktion `AreConnected` an, die zurückgibt, ob zwei per Index angegebene Knoten durch eine Kante in der globalen Adjazenzmatrix verbunden sind.

(e) Testen Sie diese beiden Funktionen, indem Sie
- mehrere Kanten erzeugen;
- prüfen, ob für *eine* erzeugte Kante die entsprechenden Knoten sowohl in der Reihenfolge Start-Ziel als auch in der Reihenfolge Ziel-Start verbunden sind;
- sich den Inhalt der Adjazenzmatrix ansehen.

2. Kantentabelle

Arbeiten Sie mit der Projektmappe der vorherigen Aufgabe weiter.

In dieser Aufgabe geben Sie die durch die Adjazenzmatrix vorgegebenen Kanten als Kantentabelle aus.

(a) Iterieren Sie mithilfe zweier verschachtelter `for`-Schleifen jeweils über Indizes der Knoten, die sich aus den Zeilen/Spalten der Adjazenzmatrix ergeben.

(b) Prüfen Sie in der inneren Schleife, ob die durch die beiden Laufvariablen bezeichneten Knoten mit einer Kante verbunden sind. Ist das der Fall, geben Sie die Kante in folgendem Format aus:

```
[Start-Index] [Ziel-Index]
```

(c) Stellen Sie sicher, dass die Kanten nur einmal ausgegeben werden.
Zur Erinnerung: Wir arbeiten mit ungerichteten Kanten.

3. Kommandozeilenparameter

Beginnen Sie mit einer Hello-World-Applikation. Sie können den Programmnamen frei wählen. Diese Aufgabe geht allerdings davon aus, dass das erzeugte Programm `main` heißt.

Weiterhin geht diese Aufgabe davon aus, dass der Funktionskopf Ihrer `main`-Funktion wie folgt lautet:

```
int main(int argc, char** argv)
```

(a) Ändern Sie die Ausgabe so, dass nach dem Begrüßungstext der Wert des ersten an die `main`-Funktion übergebenen Parameters `argc` ausgegeben wird. Rufen Sie Ihr Programm nun einmal folgendermaßen auf

```
./main
```

und einmal folgendermaßen auf

```
./main -a
```

und beobachten den Unterschied.

(b) Geben Sie nun zusätzlich, falls beim Programmaufruf angegeben, den Kommandozeilenparameter aus. Diesen finden Sie (falls angegeben) als Zeichenkette mit dem Index `1` im zweiten Parameter `argv` – einem Array von Zeichenketten.

(c) Ändern Sie Ihre Hello-World-Applikation so, dass die Ausgabe abhängig vom Kommandozeilenparameter wird:
- Parameter: `-a`
  Ausgabe: `G'day!`
- Parameter: `-b`
  Ausgabe: `Hello!`
- Parameter: `-d`
  Ausgabe: `Guten Tag!`

AUFGABEN

4. Knoten verbinden

Arbeiten Sie mit der Projektmappe aus dem vorherigen Praktikum weiter.

Ergänzen Sie Ihre Graph-Klasse um einen geeigneten Member, der die Adjazenzmatrix des Graphen darstellt. Initialisieren Sie diesen im Konstruktor des Graphen. Dazu ist es erforderlich, dass der Konstruktor die Anzahl der Knoten im Graph übernimmt.

Hinweis: Der Konstruktor legt selbst keine Knoten an. Für diesem Zweck haben Sie im vorherigen Praktikum bereits eine Methode angelegt.

Fügen Sie Ihrer Graph-Klasse Methoden hinzu, mit denen Sie Kanten einfügen können und prüfen können, ob zwei Knoten durch eine Kante verbunden sind. Stellen Sie sicher, dass diese Funktionen korrekt arbeiten.

5. Graphviz-Ausgabe

Arbeiten Sie mit der Projektmappe aus der vorherigen Aufgabe weiter.

Erweitern Sie die `ToString`-Methode Ihrer Graph-Klasse so, dass auch die Kanten im Graphviz-Kompatiblen Format ausgegeben werden:

```
␣␣[ID1]␣--␣[ID2]
```

`[IDn]` steht dabei für den Namen des jeweiligen Knoten. Damit das einfach funktioniert, ist der Einsatz eines *Getters* durchaus gerechtfertigt. Beachten Sie, dass die Kanten wie auch die Knoten mit zwei Leerzeichen eingerückt werden.

Stellen Sie sicher, dass Kanten nicht doppelt ausgegeben werden. Weiterhin benötigen die Kanten eine bestimmte Reihenfolge, damit später die automatische Überprüfung Ihrer Abgabe funktioniert:

- Die Kanten werden in der Reihenfolge ausgegeben, in der die zugehörigen Knoten angelegt wurden.
- Zunächst wird nach dem Knoten links vom horizontalen Strich sortiert.
- Ist dieser identisch, ist der Knoten rechts ausschlaggebend.
- Der Knoten rechts vom horizontalen Strich darf nicht vor dem Knoten links angelegt worden sein.

Überprüfen Sie Ihr Programm für verschiedene Graphen mit unterschiedlicher Konnektivität. Verwenden Sie Graphviz, um das Ergebnis visuell zu überprüfen.

6. Verarbeitung von Eingabedaten

Arbeiten Sie mit der Projektmappe aus der vorherigen Aufgabe weiter.

Ihr Programm wird nun neben den Knoten auch die Kanten von der Standardeingabe einlesen. In den Eingabedaten liegen die Kanten als Kantentabelle vor:

- Um dies zu kennzeichnen, steht in den Eingabedaten unmittelbar nach den Knoten eine einzelne Zeile

  ```
  E
  ```

- Anschließend folgt eine Kante pro Zeile in folgendem Format bis zum Dateiende bzw. bis zum Drücken von `Strg` + `D`:

  ```
  [Index_1]␣[Index_2]
  ```

- Die Kante verbindet die Knoten mit den Indizes `[Index_1]` und `[Index_2]`. Die Indizes ergeben sich aus der Reihenfolge, in der die Knoten in den Eingabedaten angegeben sind.

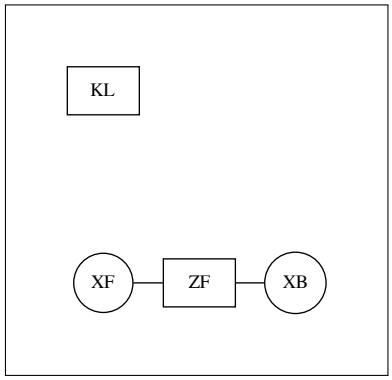
Überprüfen Sie Ihr Programm. Für diese Eingabe

```
4↩
L␣XF␣0␣0↩
H␣ZF␣1␣0↩
L␣XB␣2␣0↩
H␣KL␣0␣2↩
E↩
0␣1↩
1␣2↩
```

soll es die folgende Ausgabe produzieren:

```
graph␣{↩
␣␣XF␣[pos="0,0!",shape=circle]↩
␣␣ZF␣[pos="1,0!",shape=box]↩
␣␣XB␣[pos="2,0!",shape=circle]↩
␣␣KL␣[pos="0,2!",shape=box]↩
␣␣XF␣--␣ZF↩
␣␣ZF␣--␣XB↩
}↩
```

`neato` macht daraus die folgende Zeichnung:



7. Ausgabe der Adjazenzmatrix

Arbeiten Sie mit der Projektmappe aus der vorherigen Aufgabe weiter.

Ihr Programm wird nun neben der Graphviz-kompatiblen Ausgabe auch die jeweilige Adjazenzmatrix ausgeben können. Die Auswahl des Ausgabeformats erfolgt über einen Kommandozeilenparameter.

Sie könnten z.B. folgendes Verhalten implementieren:

- kein Kommandozeilenparameter oder Parameter `-g`:
  Ausgabe ist Graphviz-kompatibel
- Parameter `-a`:
  Ausgabe der Asjazenzmatrix

Damit die Adjazenzmatrix zur Pflichtabgabe automatisch überprüft werden kann, muss folgendes Format eingehalten werden:

- Die Zeilen und Spalten beziehen sich jeweils auf die Knoten in der Reihenfolge, in der diese in den Eingabedaten stehen.
- In jeder Zelle der Matrix steht entweder
  - `0` für *nicht verbunden* oder
  - `1` für *verbunden*.
- Zwischen den einzelnen Zellen steht ein einzelnes Leerzeichen.

- In jeder Zeile steht nach der letzten Spalte vor dem Zeilenwechsel noch ein einzelnes Leerzeichen. Auf diese Weise lässt sich die Ausgabe leichter implementieren.

Überprüfen Sie Ihr Programm. Für diese Eingabe

```
4↩
L␣XF␣0␣0↩
H␣ZF␣1␣0↩
L␣XB␣2␣0↩
H␣KL␣0␣2↩
E↩
0␣1↩
1␣2↩
```

soll es bei Aufruf mit dem Kommandozeilenparameter für die Ausgabe der Adjazenzmatrix die folgende Ausgabe produzieren:

```
0␣1␣0␣0␣↩
1␣0␣1␣0␣↩
0␣1␣0␣0␣↩
0␣0␣0␣0␣↩
```

Achten Sie darauf, dass am Ende jeder Zeile tatsächlich ein einzelnes Leerzeichen steht.

VERTIEFUNG

8. Ausgabe der Kantentabelle

Arbeiten Sie mit der Projektmappe aus der vorherigen Aufgabe weiter.

Ergänzen Sie Ihr Programm um die Möglichkeit, die Kantentabelle auszugeben. Verwenden Sie dazu z.B. den Kommandozeilenparameter `-e`.

Die Ausgabe hat dabei das Format

```
[a]␣[b]␣↩
[c]␣[d]␣
```

Wie bei der Ausgabe der Adjazenzmatrix steht am Ende jeder Zeile ein einzelnes Leerzeichen. `[a]`, `[b]`, `[c]`, `[d]` geben die Indizes der jeweiligen Knoten an. Der Index eines Knotens entspricht dessen Position in den Eingabedaten.

Damit Sie die ausgegebene Kantentabelle im Moodle-Kurs automatisch überprüfen lassen können (freiwillig), muss für obige Ausgabe gelten, dass $a \leq b$, $c \leq d$ und $a \leq c$. Diese Regel setzt sich entsprechend zeilenweise fort.

Überprüfen Sie Ihr Programm. Für diese Eingabe

```
4↩
L␣XF␣0␣0↩
H␣ZF␣1␣0↩
L␣XB␣2␣0↩
H␣KL␣0␣2↩
E↩
0␣1↩
1␣2↩
```

soll es bei Aufruf mit dem Kommandozeilenparameter für die Ausgabe der Kantentabelle die folgende Ausgabe produzieren:

```
0␣1␣↩
1␣2␣↩
```

Achten Sie darauf, dass am Ende jeder Zeile tatsächlich ein einzelnes Leerzeichen steht.

9. Ausgabe der Inzidenzmatrix

Arbeiten Sie mit der Projektmappe aus der vorherigen Aufgabe weiter.

Ergänzen Sie Ihr Programm um die Möglichkeit, die Inzidenzmatrix auszugeben. Verwenden Sie dazu z.B. den Kommandozeilenparameter `-i`.

Damit Sie die ausgegebene Kantentabelle im Moodle-Kurs automatisch überprüfen lassen können (freiwillig), muss folgendes Format eingehalten werden:

- Jede Zeile bezieht sich auf einen Knoten.
- Jede Spalte bezieht sich auf eine Kante.
- Der Startknoten einer Kante weiter links in der Matrix hat einen kleineren Index. Bei gleichem Index, hat der Endknoten einen kleineren Index. Der Index eines Knotens entspricht dessen Position in den Eingabedaten.
- In jeder Zelle der Matrix steht entweder
  - `0` Kante und Knoten sind *nicht inzident* oder
  - `1` Kante und Knoten sind *inzident*.
- Zwischen den einzelnen Zellen steht ein einzelnes Leerzeichen.
- In jeder Zeile steht nach der letzten Spalte vor dem Zeilenwechsel noch ein einzelnes Leerzeichen. Auf diese Weise lässt sich die Ausgabe leichter implementieren.

Überprüfen Sie Ihr Programm. Für diese Eingabe

```
4↩
L␣XF␣0␣0↩
H␣ZF␣1␣0↩
L␣XB␣2␣0↩
H␣KL␣0␣2↩
E↩
0␣1↩
1␣2↩
```

soll es bei Aufruf mit dem Kommandozeilenparameter für die Ausgabe der Kantentabelle die folgende Ausgabe produzieren:

```
1␣0␣↩
1␣1␣↩
0␣1␣↩
0␣0␣↩
```

Achten Sie darauf, dass am Ende jeder Zeile tatsächlich ein einzelnes Leerzeichen steht.

10. Writer-Klassen

Arbeiten Sie mit der Projektmappe aus der vorherigen Aufgabe weiter.

Vermutlich haben Sie – wie in der Anleitung vorgegeben – die Methoden zur Ausgabe direkt in Ihrer Graph-Klasse implementiert. Im Sinn von *Clean Code*[3] hat Ihre Graph-Klasse dadurch aber zu viele Aufgaben: Sie verwaltet einen Graphen *und* kann ihn in mehreren Formaten ausgeben. Besser wäre es, wenn Sie diese Funktionen trennen. Damit würden Sie das *Single Responsibility Principle*[4] umsetzen.

Dies gelingt am einfachsten, indem Sie für jedes Ausgabeformat eine eigene `Writer`-Klasse implementieren, die von einer `Writer`-Basisklasse ableitet. Letztere leitet wiederum von `Printable` ab (siehe Vertiefung im vorherigen Praktikum).

[3] Robert C. Martin: Clean Code. Prentice Hall. 2008. ISBN: 9780136083238
[4] Robert C. Martin: The Single Responsibility Principle. In The Clean Code Blog. 2014. Link

Jeder `Writer` implementiert das Konzept `Printable` – also eine eigene `ToString`-Methode. Diese verwendet dann lediglich die von Ihrer Graph-Klasse bereitgestellten Funktionen, um z.B. die Anzahl der Knoten abzufragen oder zu prüfen, ob zwei Knoten verbunden sind. Auf diese Weise übernimmt Ihre Graph-Klasse nur noch die Verwaltung, die `Writer` nur noch die Ausgabe. Letztere könnte im Code dann folgendermaßen erfolgen:

```
Graph g(5);
...
std::cout << AdjacencyMatrixWriter(g) << std::endl;
```