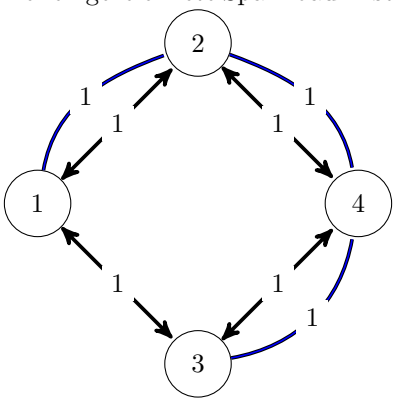
Schubert Julian, Gruppe 4
Philipp Wahl, Gruppe 4

Abgabe zum 1. Übungsblatt (AGT 21)

Aufgabe 1:

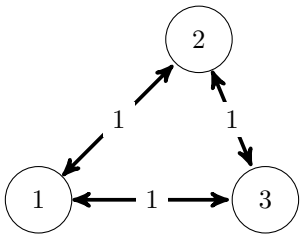
a) Diese Aussage ist korrekt. In einem zusammenhängenden Graphen besucht die Breitensuche jeden knoten genau ein mal. Der Breitensuchenbaum enthält genau $|V| - 1$ Kanten (da jeder Knoten über genau eine Kante besucht wird) und hat somit ein gesamtes Gewicht von $|V| - 1$. Da der minimale Spannbaum ebenfalls $|V| - 1$ Kanten enthalten muss (Graph ist zusammenhängend) und alle Kanten das Gewicht 1 haben, muss das Gewicht jedes minimalen Spannbaums ebenfalls gleich $|V| - 1$ sein. Da also alle möglichen Breitensuchenbäume das selbe gewicht wie der minimale Spannbaum (bzw. die minimalen Spannbäume) hat, ist jeder Breitensuchbaum somit ein minimaler Spannbaum.

b) Der blau gezeichnete minimale Spannbaum vom Graphen G ist kein Breitensuchbaum mit Quelle 1. Dies ist gegeben da wenn man mit der Breitensuche von 1 startet, man zunächst den Knoten 2, dann den Knoten 3 besuchen würde (oder umgekehrt). Der eingezeichnete Spannbaum ist also kein Breitensuchbaum mit Startpunkt 1.



Aufgabe 2:

a) Die Skize zeigt den kleinsten nicht zweifärbbaren Graphen.
Graph mit 1 Knoten: Eigenschaft trivial erfüllt
Graph mit 2 Knoten: Färbe Knoten 1 rot, Knoten 2 blau
$\Rightarrow$ Der kleinste nicht zweifärbbare Graph hat 3 Knoten.



Ein Graph mit 0 Knoten und 0 Kanten verletzt die Eigenschaft der zweifärbbarkeit nicht, da es ja keine Kante gibt die die Eigenschaft verletzt.

b) Der Algorithmus hat eine Laufzeit von O($|V| + |E|$) da unsere erste For-Schleife jeden Knoten genau einmal behandelt (wir gehen davon aus das c für jeden Knoten eine Färbung besitzt) und wir in der zweiten Schleife ein mal über alle Kanten iterieren.

```
TesteFärbung(G, c)
    // Map die für jeden Knoten die Färbung speichert
    f = {}
    foreach k ∈ c do Alle Knoten färben
    |   // Färbung des Knoten speichern
    |   f[k] = k.color
    foreach e ∈ V do Testen ob alle Knoten passend gefärbt sind
    |   if f[e.start] == f[e.end] then Wenn die Knoten am Start und Ende
    |    der Kante nicht unterschiedlich gefärbt sind
    |    └ return false
    return true
```

c) Der Algorithmus läuft in O($|V| + |E|$) da von jedem Knoten alle ausgehenden Kanten behandelt werden.

```
Färbbar(G)
    c = 0
    foreach k ∈ V do
    |   if k.c == (c + 1%2) then Wenn der aktuelle Knoten schon eine
    |    Färbung hat die anders ist als die aktuelle Färbung
    |    └ Return false
    |   // Den aktuellen Knoten in der aktuelle Farbe färben k.color = c
    |   foreach n ∈ Adj[k] do
    |   |   if n.color == k.color then Wenn beide Knoten die selbe Farbe
    |   |    haben
    |   |    └ Return false
    |   // C von 0 auf 1 bzw von 1 auf 0 setzen
    |   c = c + 1 % 2
    return true
```

Aufgabe 3:

a) Die Knoten des Graphen sind alle Felder des Schachbrettes. Von einem Knoten (also Schabrett-Feld) gibt es eine Kante zu all den Feldern, auf die ein Springer sich (vom aktuellen Feld aus) bewegen kann.

b) Schaut man sich alle 8 Möglichkeiten an die der Springer zum ziehen hat (Wenn er nicht am Rand steht), fällt auf, dass sich der Springer **IMMER** von einem weißen Feld auf ein Schwarzes Feld (auf dem Schachbrett) bewegt. Daher muss der Graph zweifärbbar sein.

c) Es ergeben sich im gerichteten Graphen genau 336 Kanten, im ungerichtetem demnach 168. Die berechnung haben wir mit folgendem Python-Code durchgeführt (der auf unserem Lösungsansatz aus paragraph d beruht):

```
def calculate_possible_moves(size: int) -> int:

    # To small for any moves
    if size < 3:
        return 0

    if size == 3:
        # Two move from each field (but the middle one)
        return 16

    # 4 Corners
    corners = sum([4 * corner for corner in [2, 3, 3, 4]])

    # Border of the board
    border_length = max(0, size - 4)
    borders = sum([4 * border * border_length for border in [4, 6]])

    # Inner part of the board
    inner_length = max(0, size - 4)
    inner = 8 * inner_length * inner_length

    return corners + borders + inner

print('\n'.join(str(calculate_possible_moves(size)) for size in range(0, 10)))
```

d) **1. Fall:** $n < 3$:
Auf einem solchen Schachfeld hat der Springer null Möglichkeiten, es gibt also null kanten.
2. Fall: $n = 3$:
Auf jeden Feld (außer dem Feld in der Mitte) hat der Springer 2 Möglichkeiten, es gibt also $[(n \cdot n) - 1] * 2 = 16$ Möglichkeiten.
3. Fall: $n > 3$
1. Betrachte alle 2x2 Ecken des Feldes:
Auf dem Feld ganz in der Ecken hat der Springer nur 2 Möglichkeiten, auf den anderen beiden Feldern am Rand hat er 3 Möglichkeiten, im letzen Feld der 2x2 Ecke hat er 4 Möglichkeiten
2. Betrachte die $2 \times m$ Teilstücke zwischen den Ecken:
Jedes dieser Teilstücke hat eine Länge von $n-4$ (da links und rechts die beiden Ecken sind). In dem einen Streifen am Rand hat der Springer 4 Möglichkeiten (pro Feld), auf dem anderen Streifen sind es 6 Möglichkeiten.

3. Betrachte die Mitte des Feldes:
Die Mitte des Feldes enthält genau $((n-4)\cdot(n-4))$ Felder, auf jedem dieser Felder hat der Springer 8 Möglichkeiten.
Da wir eine quadratisch wachsende Anzahl an Möglichkeiten haben ist die obere Schranke $O(n^2)$

Aufgabe 4:

1. Fall: Null Knoten mit $\deg(v)$ ist ungerade:
Offensichtlich erfüllt, da es in diesem Fall (laut Vorlesung) einen EulerKreis geben muss. Jeder Eulerkreis ist gleichzeitig ein Eulerweg, da ein Eulerkreis ein Eulerweg ist bei dem der Startknoten gleich dem Endknoten ist
2. Fall: Zwei Knoten mit $\deg(v)$ ist ungerade:
Da G zusammenhängend ist gibt es einen Pfad p der x-viele Kanten enthält der die beiden Knoten mit ungeradem Grad (u, v) verbindet. Wenn man nun alle Kanten im Pfad p entfernt, gibt es einen Eulerkreis im Restgraphen. Wenn man nun den Pfad p wieder hinzufügt, kann man den Eulerkreis aus dem Teilgraphen mit dem Pfad p kombinieren, wodurch man einen Eulerpfad mit start u und ende v erhält.