# Aufgabe 4: Schrebergärten

Team: Pied Piper
Team-ID: 00342

26. November 2018

## Inhalt



## Lösungsidee

Um eine ideale Anordnung der Schrebergärten zu finden, werden verschiedene Anordnungen von Rechtecken erstellt und bewertet. Ist bei der Erstellung einer Anordnung bereits ersichtlich, dass sie nicht zu einer vernünftigen Lösung führen kann, so wird diese abgebrochen und eine andere wird durchgespielt. Bei einer vollständigen Anordnung mit einem Platzverlust (Grundstück, das abgekauft werden muss – Grundstück, das eigentlich verwendet wird) von 0 hat man bereits die optimalste Lösung für das Problem. Sofern keine Anordnung mit einem Platzverlust von 0 bewertet werden sollte, so wird die Anordnung als Lösung ausgegeben, die am besten bewertet wurde.

Zur Veranschaulichung einer Anordnung wird ein Koordinatensystem zur Hilfe gezogen. Ein Schrebergarten im Koordinatensystem lässt sich dabei mit zwei Vektoren darstellen, einem Ortsvektor, der angibt, wo sich das Rechteck befindet und einem Richtungsvektor, der das Rechteck beschreibt.

Im Folgenden werden die Erstellung und Bewertung von Anordnungen erläutert. Um die Anzahl der möglichen Anordnungen zu verkleinern werden Rechtecke nur an Eckpunkten der aktuellen Anordnung positioniert. Diese Regel ohnehin sinnvoll, da man glatte Kanten haben will, also wenig Eckpunkte, für einen minimalen Platzverlust. Ist noch keine Anordnung vorhanden, so fungiert der Ursprung (0,0) als Eckpunkt. Sei n die Anzahl der Schrebergärten, so gibt es zu Beginn also nur n Möglichkeiten für die erste Anordnung. Bei allen dieser Möglichkeiten befindet sich somit zunächst ein Schrebergarten im Ursprung. Im zweiten Schritt wird also ein anderer Schrebergarten an einem der nun drei vorhandenen Eckpunkte gesetzt, sofern dieser nicht mit den Achsen des Koordinatensystems oder dem bereits vorhandenen Rechteck interferiert. Das Setzen von Rechtecken setzt sich solange fort bis entweder alle Rechtecke gesetzt worden sind oder bereits absehbar ist, dass eine aktuelle Teillösung nicht besser werden kann als die bereits vorhandene beste Anordnung. In beiden Fällen wird ein Schritt oder mehrere Schritte zurückgenommen, und es werden andere Möglichkeiten durchgespielt. Wie man sieht basiert diese Lösungsidee also auf dem Backtracking-Algorithmus, der bei zunehmender Anzahl an Rechtecken im Worst-Case-Szenario eine exponentielle Laufzeit aufweist.

Beim Setzen eines Rechtecks an einem Eckpunkt gibt es vier Möglichkeiten, da der Richtungsvektor des Rechtecks in vier verschiedene Richtungen zeigen kann (schräg nach oben rechts, schräg nach unten rechts,

schräg nach unten links und schräg nach oben links). Es werden alle gültigen Ausrichtungen weiterverfolgt. Gültig heißt dabei, dass die Ausrichtung weder den Rahmen des Koordinatensystems überschreitet noch die aktuelle Anordnung überschneidet.

# Umsetzung

Die Lösungsidee wurde in Python implementiert. Es muss die Datei Schrebergaerten.py gestartet, anschließend kann man in der Konsole eine gegebene Menge an Schrebergärten (definiert durch ihre Länge und Breite) eingeben und die optimale Anordnung berechnen lassen. Es folgt ein Fenster mit der grafischen Ausgabe.

Für die Berechnung einer optimalen Anordnung wurden drei Klassen geschrieben. Bei der ersten handelt es sich um die Klasse „Koordinate“, die einen x- und einen y-Wert annimmt. Mit ihr kann man zum Beispiel nicht zugeordnete Rechtecke darstellen (x steht dann für die Breite und y für die Höhe). Später hat die Klasse noch eine andere Bedeutung.

Bei der zweiten handelt es sich um die Klasse „RechteckVektor“. Mit dieser Klasse lässt sich ein Schrebergarten-Rechteck im Koordinatensystem darstellen. Er nimmt einen x – und y-Wert für den Richtungsvektor an und einen zweiten x- und y-Wert für den Ortsvektor.

Die eigentliche Berechnung einer optimalen Anordnung findet in der Klasse „Schrebergaerten“ statt, genauer gesagt in dessen Methode „finde_anordnung“. Alle weiteren Methoden der Klasse dienen zur Auslagerung kleinerer Teilprobleme.

Die eingegebenen Schrebergärten werden in der Liste „eingabe“ in der Klasse „Schrebergaerten“ gespeichert, sodass bei jedem rekursiven Aufruf der Methode „finde_anordnung“ auf sie zugegriffen werden kann.

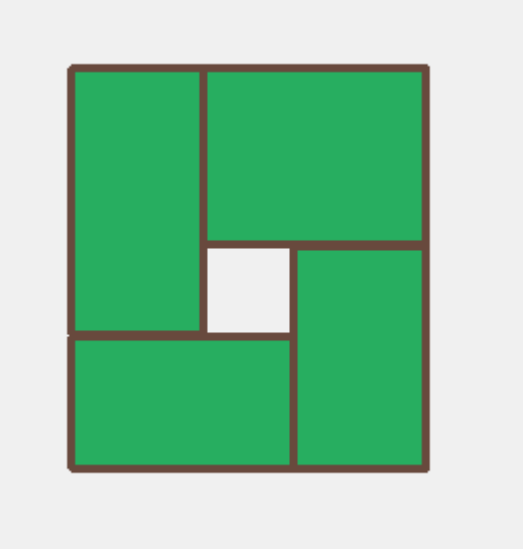
Die Parameter „aktuelle_anordnung“ und „benutzte_rechtecke“ dienen als Ausgangspunkt für das nächst zu setzende Rechteck und tragen später beim rekursiven Methoden-Aufruf eine wichtige Rolle.

Zunächst wird mithilfe einer for-Schleife jedes Rechteck der Eingabe durchgegangen. Wenn ein Rechteck in der aktuellen Anordnung vorhanden ist, dessen Index sich also in der Liste benutzte_rechtecke befindet, so wird dieses übersprungen. Andernfalls werden nun alle Eckpunkte in einer for-Schleife durchgegangen. Die Liste aller Eckpunkte für die aktuelle Anordnung gibt die Methode „get_eckpunkte“ zurück. Es handelt sich dabei um eine Liste von Koordinaten-Objekte. Für jeder in Frage kommender Eckpunkt wird nun jede mögliche Ausrichtung (Es geht hierbei nur um die positionelle Ausrichtung, die Nord-Süd- bzw. Ost-West-Ausrichtung bleibt unangetastet) des zu setzenden Rechtecks betrachtet, ebenfalls mithilfe einer for-Schleife. Eine Liste mit allen Rechteck-Ausrichtungen liefert die Methode „alle_ausrichtungen“. Nun wird überprüft, ob das aktuelle Rechteck mit der spezifischen Ausrichtung am spezifischen Eckpunkt an irgendeiner Stelle mit der aktuellen Anordnung interferiert und sonst überschneidet. Diese Validierung erfolgt in der Methode „ueberschneidung_vorhanden“. Ist die Validierung positiv, so wird als nächstes überprüft, ob die neue Anordnung aussichtslos ist oder nicht, indem als erstes der Platzverlust der neuen Anordnung berechnet wird. Dies erledigt die Methode „get_platzverlust“. Anschließend wird geschaut, ob die Differenz aus Platzverlust und Fläche der übrigbleibenden Rechtecke größer ist als der Platzverlust der aktuell besten Anordnung. Ist dies der Fall, ist ein weiteres Verfolgen dieser Teillösung redundant. Sofern die neue Anordnung komplett ist, wird geschaut, ob sie perfekt ist (kein Platzverlust). Falls ja, wird sie zurückgegeben, falls nein, wird sie als beste Anordnung gespeichert, wenn sie platzsparender als die beste Anordnung zuvor. Ist die neue Anordnung nicht komplett kommt es zum rekursiven Methodenaufruf mit der neuen Anordnung und der neuen Liste an benutzten Rechtecken.

Bietet eine Anordnung keine weiteren Lösungen, so wird [-1] als Fehlerzustand zurückgegeben. Wenn es also nicht zu einer perfekten Anordnung kommt, dann ist das Ergebnis [-1], jedoch kann man auf die Klassen-Variable „Schrebergaerten.beste_anordnung“ für eine möglichst platzsparende Anordnung zurückgreifen.

# Beispiele

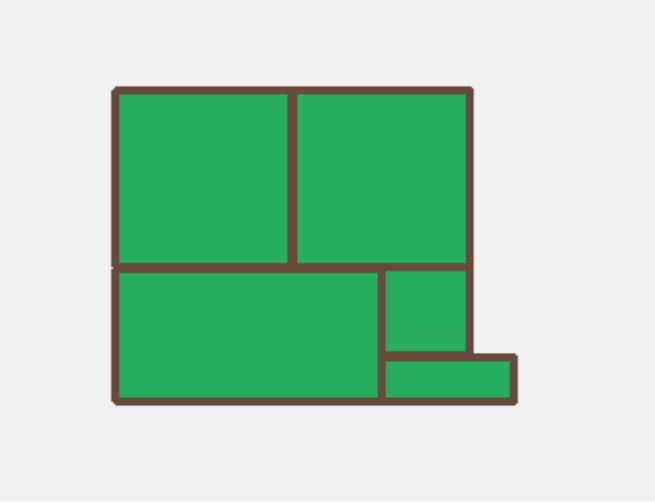
### *Beispiel 1) 25 x 15, 15 x 30, 15 x 25, 25 x 20*



```
-->Platzverbrauch: 100
-->Breite: 40
-->Höhe: 45
```

Wie man sehen kann wurde eine optimale Lösung gefunden. Der Platzverbrauch ist derselbe wie in der Aufgabenstellung. Einziger Unterschied ist die Anordnung, wobei diese punktsymmetrisch zu der in der Aufgabenstellung ist.

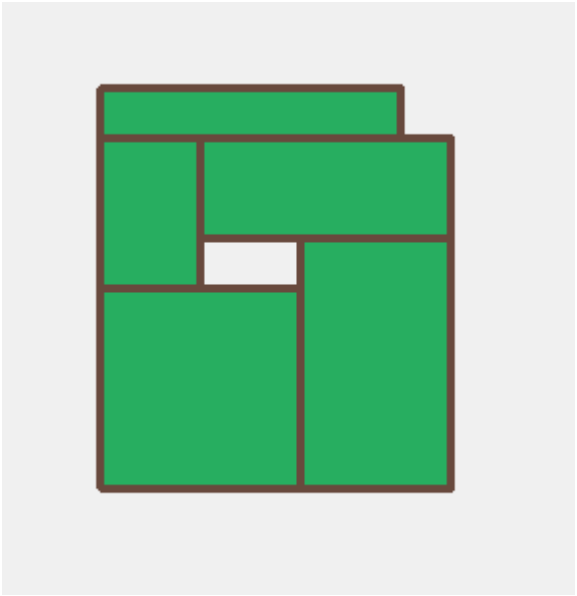
### *Beispiel 2) 6 x 3, 2 x 2, 3 x 1, 4 x 4, 4 x 4*



```
-->Platzverbrauch: 6
-->Breite: 9
-->Höhe: 7
```

Auch hier ist eine möglichst platzsparende Lösung vorhanden. Ich, als Mensch, habe keine bessere Anordnung finden können.

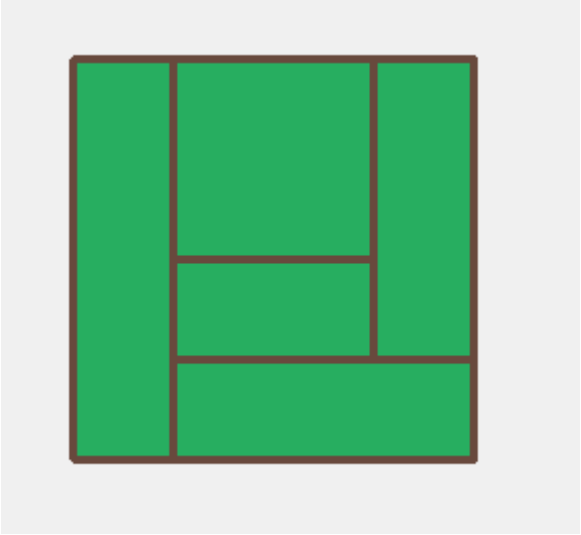
### *Beispiel 3) 4 x 4, 2 x 3, 6 x 1, 5 x 2, 3 x 5*



```
-->Platzverbrauch: 3
-->Breite: 7
-->Höhe: 8
```

Hier ist ebenfalls eine sehr platzsparende Lösung mit einem Platzverbrauch von nur drei Einheiten. Wahrscheinlich ist mit den gegebenen Beispielen keine perfekte Anordnung möglich, weshalb ich an dieser Stelle ein eigenes Beispiel ergänze.

***Eigenes Beispiel) 1 x 4, 2 x 2, 2 x 1, 3 x 1, 1 x 3***



Quadratisch, praktisch, gut. Mit diesen drei Worten lässt sich das eigene Beispiel beschreiben. Hiermit soll gezeigt werden, dass der Algorithmus auch „perfekte“ Anordnungen finden kann.

```
-->Platzverbrauch: 0
-->Breite: 4
-->Höhe: 4
```

Es fällt auf, dass in allen Beispielen, im letzten ganz deutlich, die Form der Anordnung einem Quadrat nahekommt.

# Quellcode

```
import sys
from PyQt5.QtGui import QPainter, QBrush, QPen, QPaintEvent, QColor
from PyQt5.QtWidgets import QApplication, QMainWindow
from PyQt5.QtCore import Qt


class Koordinate:
    def __init__(self, x, y):
        self.x = x
        self.y = y


# Mit einem RechteckVektor-Objekt lässt sich ein Rechteck im Koordinatensystem darstellen
class RechteckVektor:
    def __init__(self, richtung_x, richtung_y, ortsvektor_x, ortsvektor_y):
        self.x = richtung_x
        self.y = richtung_y
        self.ox = ortsvektor_x
        self.oy = ortsvektor_y


class Schrebergaerten:
    eingabe = []
    kleinster_platzverlust = -1  # -1 dient als Platzhalter, falls noch kein Platzverlust berechnet wurde
    beste_anordnung = []
```

```python
@staticmethod
def finde_anordnung(aktuelle_anordnung, benutzte_rechtecke):
    # Es wird jedes Rechteck durchgegangen...
    for rechteck_index in range(len(Schrebergaerten.eingabe)):
        # ... mit Ausnahme von Rechtecken, die in der aktuellen Anordnung bereits vorhanden sind
        if rechteck_index in benutzte_rechtecke:
            continue
        # Jeder Eckpunkt der aktuellen Anordnung wird durchgegangen
        for eckpunkt in Schrebergaerten.get_eckpunkte(aktuelle_anordnung):
            # Jede mögliche Ausrichtung wird in Betracht gezogen
            for rechteck in Schrebergaerten.alle_ausrichtungen(Schrebergaerten.eingabe[rechteck_index]):
                # Es wird überprüft, ob ausgerichtetes Rechteck an diesem Eckpunkt valide gesetzt werden kann
                if not Schrebergaerten.ueberschneidung_vorhanden(aktuelle_anordnung,
                                                                 RechteckVektor(rechteck.x, rechteck.y,
                                                                                eckpunkt.x,
                                                                                eckpunkt.y)):
                    # Neue Anordnung wird in einer Liste gespeichert
                    neue_anordnung = []
                    neue_anordnung.extend(aktuelle_anordnung)
                    neue_anordnung.append(RechteckVektor(rechteck.x, rechteck.y,
                                                         eckpunkt.x,
                                                         eckpunkt.y))
                    # Platzverlust der neuen Anordnung wird ermittelt
                    platzverlust = Schrebergaerten.getPlatzverlust(neue_anordnung)

                    # Falls es noch keine beste Anordnung gibt
                    if Schrebergaerten.kleinster_platzverlust != -1:
                        # Falls die Teillösung aussichtslos ist
                        if platzverlust - Schrebergaerten.getUebrigeFlaeche(
                                benutzte_rechtecke) >= Schrebergaerten.kleinster_platzverlust:
                            continue
                    # Wenn die neue Anordnung vollständig ist
                    if len(neue_anordnung) == len(Schrebergaerten.eingabe):
                        if platzverlust == 0:
                            # perfekte Lösung wird zurückgegeben
                            return neue_anordnung
                        # Lösung ist nicht perfekt, aber die platzsparendste bisher
                        elif platzverlust < Schrebergaerten.kleinster_platzverlust or Schrebergaerten.kleinster_platzverlust == -1:
                            # Beste anordnung und kleinster Platzverlust bisher werden gespeichert
                            Schrebergaerten.beste_anordnung.clear()
                            Schrebergaerten.beste_anordnung.extend(neue_anordnung)
                            Schrebergaerten.kleinster_platzverlust = platzverlust

                    # Falls Anordnung noch nicht komplett
                    else:
                        used = []
                        used.extend(benutzte_rechtecke)
                        used.append(rechteck_index)
                        x = Schrebergaerten.finde_anordnung(neue_anordnung, used)
                        # Wenn es für die nächste Anordnung eine perfekte Lösung gibt...
                        if not x == [-1]:
                            # ... wird diese zurückgegeben
                            return x
    # Es konnte keine perfekte Lösung gefunden werden
    return [-1]
```

```python
@staticmethod
def get_eckpunkte(anordnung):
    # Falls noch keine Anordnung...
    if len(anordnung) == 0:
        # ...fungiert der Ursprung als Eckpunkt
        return [Koordinate(0, 0)]

    # Zunächst werden alle Eckpunkt-Koordinaten in einer Liste gespeichert
    eckpunkte_aller_rechtecke = []
    for rechteck in anordnung:
        eckpunkte_aller_rechtecke.append(Koordinate(rechteck.ox, rechteck.oy))
        eckpunkte_aller_rechtecke.append(Koordinate(rechteck.ox + rechteck.x, rechteck.oy + rechteck.y))
        eckpunkte_aller_rechtecke.append(Koordinate(rechteck.ox + rechteck.x, rechteck.oy))
        eckpunkte_aller_rechtecke.append(Koordinate(rechteck.ox, rechteck.oy + rechteck.y))

    # und in ein Feldersystem umgewandelt, das einer Tabelle gleichkommt
    # Die Felder in denen Rechtecke vorhanden werden auf True gesetzt
    feldersystem = Schrebergaerten.anordnung_in_felder(anordnung)
    eckpunkte_der_anordnung = []
    for eckpunkt in eckpunkte_aller_rechtecke:
        # Ursprung diente bereits als Eckpunkt, deswegen übersprungen
        if eckpunkt.x == 0 and eckpunkt.y == 0:
            continue

        # Schauen ob es sich am Rand um einen Eckpunkt handelt
        if eckpunkt.x == 0:
            if not feldersystem[0][eckpunkt.y] or not feldersystem[0][eckpunkt.y - 1]:
                eckpunkte_der_anordnung.append(eckpunkt)
            continue

        if eckpunkt.y == 0:
            if not feldersystem[eckpunkt.x][0] or not feldersystem[eckpunkt.x - 1][0]:
                eckpunkte_der_anordnung.append(eckpunkt)
            continue

        # Prüfen, ob im Umfeld von einem Kästchen ein Feld False ist. Falls ja --> Eckpunkt, da am Rand
        if not feldersystem[eckpunkt.x][eckpunkt.y] or not feldersystem[eckpunkt.x - 1][eckpunkt.y] or not \
                feldersystem[eckpunkt.x][eckpunkt.y - 1] or not feldersystem[eckpunkt.x - 1][eckpunkt.y - 1]:
            eckpunkte_der_anordnung.append(eckpunkt)
            continue

    return eckpunkte_der_anordnung
```

```python
@staticmethod
# Erhalte Breite und Höhe für das gesamte Vieleck der aktuellen Anordnung
def get_grenze(anordnung):
    breite = 0
    hoehe = 0
    for rechteck in anordnung:
        # Es wird der Wert für die Breite zurückgegeben, der am weitesten rechts liegt
        if rechteck.ox > breite:
            breite = rechteck.ox
        if rechteck.x + rechteck.ox > breite:
            breite = rechteck.x + rechteck.ox
        # Es wird der Wert für die Höhe zurückgegeben, der am weitesten oben liegt
        if rechteck.y + rechteck.oy > hoehe:
            hoehe = rechteck.y + rechteck.oy
        if rechteck.oy > hoehe:
            hoehe = rechteck.oy
    return [breite, hoehe]

@staticmethod
def alle_ausrichtungen(rechteck):
    # Jede mögliche Ausrichtung durch jeweils Tauschen eines oder beider Vorzeichen
    return [Koordinate(rechteck.x, rechteck.y), Koordinate(-rechteck.x, rechteck.y),
            Koordinate(rechteck.x, -rechteck.y), Koordinate(-rechteck.x, -rechteck.y)]
```

```python
@staticmethod
# Prüfen, ob Rechteck valide gesetzt werden kann, ohne Überschneidung
def ueberschneidung_vorhanden(anordnung, rechteck):
    # Erstmal wird überprüft, ob das zu setzende Rechteck das Koordinatensystem nicht überschneidet
    if rechteck.x + rechteck.ox < 0 or rechteck.y + rechteck.oy < 0:
        return True

    feldersystem_anordnung = Schrebergaerten.anordnung_in_felder(anordnung)
    feldersystem_rechteck = Schrebergaerten.anordnung_in_felder([rechteck])

    if len(feldersystem_rechteck) >= len(feldersystem_anordnung):
        x_range = len(feldersystem_rechteck)
    else:
        x_range = len(feldersystem_anordnung)

    if len(feldersystem_rechteck[0]) >= len(feldersystem_anordnung[0]):
        y_range = len(feldersystem_rechteck[0])
    else:
        y_range = len(feldersystem_anordnung[0])

    # Feldersystem, in das beide einzelenen Feldersysteme reinpassen
    feldersystem_kombi = []
    for x_koordinate in range(x_range):
        feldersystem_kombi.append([])
        for y_koordinate in range(y_range):
            feldersystem_kombi[x_koordinate].append(False)

    # Im großen Feldersystem werden die Felder in denen sich das Rechteck befindet auf True gesetzt
    for x in range(len(feldersystem_rechteck)):
        for y in range(len(feldersystem_rechteck[0])):
            if feldersystem_rechteck[x][y]:
                feldersystem_kombi[x][y] = True

    # Es wird Feld für Feld im Feldersystem der Anordnung durchgegangen
    for x in range(len(feldersystem_anordnung)):
        for y in range(len(feldersystem_anordnung[0])):
            # Überschneidung des Rechtecks mit der Anordnung
            if feldersystem_anordnung[x][y] and feldersystem_kombi[x][y]:
                return True

    return False
```

```python
@staticmethod
def anordnung_in_felder(anordnung):
    grenzen = Schrebergaerten.get_grenze(anordnung)
    breite = grenzen[0]
    hoehe = grenzen[1]
    feldersystem = []

    # Tabelle/Feldersystem wird erstellt, das ein Kästchen weiter in die Breite und Höhe geht
    for x_koordinate in range(breite + 1):
        feldersystem.append([])
        for y_koordinate in range(hoehe + 1):
            # Alle Felder des Koordinatensystems werden erstmal auf False gesetzt
            feldersystem[x_koordinate].append(False)

    # Die Felder in denen sich Rechtecke befinden, werden nun auf True gesetzt
    for rechteck in anordnung:
        if rechteck.x > 0:
            x_range = range(rechteck.ox, rechteck.ox + rechteck.x)
        else:
            x_range = range(rechteck.ox + rechteck.x, rechteck.ox)

        if rechteck.y > 0:
            y_range = range(rechteck.oy, rechteck.oy + rechteck.y)
        else:
            y_range = range(rechteck.oy + rechteck.y, rechteck.oy)

        for x in x_range:
            for y in y_range:
                feldersystem[x][y] = True

    return feldersystem

@staticmethod
def getPlatzverlust(anordnung):
    # Platzverlust = Fläche der Anordnung - Fläche aller Rechtecke
    belegte_flaeche = 0
    for rechteck in anordnung:
        belegte_flaeche += abs(rechteck.x * rechteck.y)
    dimensionen = Schrebergaerten.get_grenze(anordnung)
    return dimensionen[0] * dimensionen[1] - belegte_flaeche

@staticmethod
# Fläche der übrigbleibenden Rechtecke
def getUebrigeFlaeche(benutzte_rechtecke):
    flaeche = 0
    for rechteck_index in benutzte_rechtecke:
        flaeche += Schrebergaerten.eingabe[rechteck_index].x * Schrebergaerten.eingabe[rechteck_index].y

    return flaeche
```