1. (4 Punkte) Ein Suchproblem hat die Zustände A bis G. Die Tabelle gibt an, welche Folgezustände `nextstates` für einen Zustand liefert, und zwar in dieser Reihenfolge. Startzustand ist A, Zielzustand ist G.

Zustand	Folgezustände
A	B, C
B	A, D, E
C	A, F
D	B, G
E	B, G
F	C, G
G	D, E, F

   (a) In welcher Reihenfolge werden die Zustände von der Breitensuche untersucht?

   (b) Gib die frontier und das Dictionary `prev` an, nachdem der Zustand C untersucht wurde.

   (c) Welchen Weg liefert `reconstructPath` am Ende?

Lösung:

(a) A, B, C, D, E, F, G

(b) frontier: D, E, F

```
prev = {'A': None, 'B': 'A', 'C': 'A', 'D': 'B', 'E': 'B', 'F': 'C'}
```

(c) A, B, D, G

G wurde zuerst von D aus gefunden, deshalb ist D der Vorgänger von G.

2. (3 Punkte) (a) Warum findet die Breitensuche einen Weg mit möglichst wenigen Zügen?

   (b) In `bfs` wird `frontier.popleft()` durch `frontier.pop()` ersetzt. Was ändert sich?

   (c) Das Dictionary `prev` speichert die Vorgänger. Wozu wird es in `bfs` außerdem gebraucht?

Lösung:

(a) Die frontier ist eine Warteschlange. Deshalb werden zuerst alle Zustände untersucht, die einen Zug vom Start entfernt sind, dann alle mit zwei Zügen und so weiter. Der erste Weg zum Ziel hat also die kleinste Anzahl von Zügen.

(b) Die frontier wird zum Stack, zuerst wird der zuletzt gefundene Zustand untersucht. Das ist eine Tiefensuche. Sie findet einen Weg, wenn es einen gibt, aber meist nicht den kürzesten.

(c) Mit `v not in prev` wird geprüft, ob ein Zustand schon gefunden wurde. Ohne diese Prüfung kämen Zustände mehrfach in die frontier, und die Suche könnte endlos hin und her laufen (z.B. im Labyrinth einen Schritt nach links und wieder nach rechts).

3. (1 Punkt) Zeichne den Weg ein, den die Breitensuche von S nach E findet. (Markiere die entsprechenden Punkte oder verbinde sie mit einer Linie.) Die Breitensuche erzeugt die Folgezustände in der Reihenfolge oben, unten, links, rechts.

```
XXXXXXXXXXXXXXXXXX
X.............E...X
X..................X
X..XXXX..XXXXX....X
X..................X
X..S...............X
XXXXXXXXXXXXXXXXXX
```

Lösung:

```
XXXXXXXXXXXXXXXXXX
X         oooooo E   X
X         o          X
X   XXXXo XXXXX      X
X   ooooo            X
X   S                X
XXXXXXXXXXXXXXXXXX
```

4. (2 Punkte) Ein Feld hat die Koordinaten (Zeile, Spalte), das X oben links hat die Koordinaten (0, 0). Die Breitensuche erzeugt die Folgezustände in der Reihenfolge oben, unten, links, rechts. (Die Punkte sollen das Abzählen erleichtern.)

   Die Breitensuche wird angehalten, nachdem der Zustand (4, 1) untersucht wurde. Wie viele Zustände wurden untersucht, wie viele befinden sich in der frontier? Zeichne in das große Labyrinth die Zustände in der frontier mit einer Tilde ein, die untersuchten Zustände mit einem Punkt.

```
XXXXXXXXXX
X........X
X..E.....X
XXXXXXX..X
X....S...X
XXXXXXXXXX
```

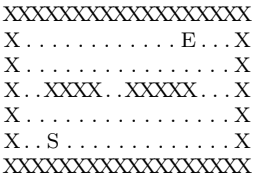
```
XXXXXXXXXX
X........X
X........X
X........X
X........X
XXXXXXXXXX
```

Lösung:

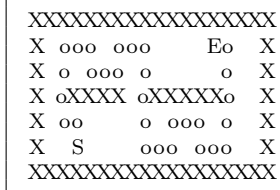
```
XXXXXXXXXX
X          X
X   E    ~ X
XXXXXXX.~X
X........X
XXXXXXXXXX
```

Untersucht: 9, in der frontier: 2

5. (2 Punkte) Zeichne den Weg ein, den die Tiefensuche von S nach E findet, und gib seine Länge an. (Markiere die entsprechenden Punkte oder verbinde sie mit einer Linie.) Die Tiefensuche legt die Folgezustände so auf den Stack, dass sie in der Reihenfolge oben, unten, links, rechts wieder herausgenommen werden.

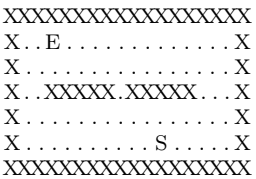


Lösung:

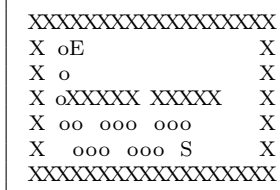


Weglänge: 30

6. (2 Punkte) Zeichne den Weg ein, den die Tiefensuche von S nach E findet, und gib seine Länge an. (Markiere die entsprechenden Punkte oder verbinde sie mit einer Linie.) Die Tiefensuche legt die Folgezustände so auf den Stack, dass sie in der Reihenfolge oben, unten, links, rechts wieder herausgenommen werden.



Lösung:

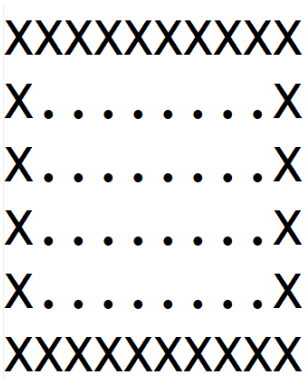


Weglänge: 18

7. (2 Punkte) Ein Feld hat die Koordinaten (Zeile, Spalte), das X oben links hat die Koordinaten (0, 0). Die Tiefensuche legt die Folgezustände so auf den Stack, dass sie in der Reihenfolge oben, unten, links, rechts wieder herausgenommen werden. (Die Punkte sollen das Abzählen erleichtern.)

   Die Tiefensuche wird angehalten, nachdem der Zustand (1, 1) untersucht wurde, seine Folgezustände liegen also schon auf dem Stack. Wie viele Zustände wurden untersucht, wie viele befinden sich in der frontier? Zeichne in das große Labyrinth die Zustände in der frontier mit einer Tilde ein, die untersuchten Zustände mit einem Punkt.

```
XXXXXXXXXX
X........X
X..E.....X
X..XXXX..X
X....S...X
XXXXXXXXXX
```



Lösung:

```
XXXXXXXXXX
X..~     X
X~.E     X
X~.XXXX  X
X~...S~  X
XXXXXXXXXX
```

Untersucht: 8, in der frontier: 6. Das Ziel E liegt schon in der frontier.

8. (1 Punkt) Zeichne ein Labyrinth, in dem die Tiefensuche einen Weg der Länge 7 findet.

Lösung: z.B.

```
XXXXX
X E X
X   X
X   X
X   X
X   X
X   X
X   X
X S X
XXXXX
```

Die Tiefensuche geht zuerst nach oben und findet so den Weg mit 7 Schritten.

9. (3 Punkte) Beim 8-Puzzle ist ein Zustand ein Tupel mit den Zahlen von links oben nach rechts unten, die Leerstelle ist die 0.

```
def nextstates(state):
    tmp = []
    i = state.index(0)
    for d in [-3, 3, -1, 1]:
        j = i + d
        if 0 <= j < 9 and (i % 3 == j % 3 or i // 3 == j // 3):
            a = list(state)
            a[i], a[j] = a[j], a[i]
            tmp.append(tuple(a))
    return tmp
```

   (a) Gib die Folgezustände von `(3, 1, 2, 0, 4, 5, 6, 7, 8)` in der Reihenfolge an, in der `nextstates` sie liefert.

   (b) Für $d = -1$ ist $j = 2$ ein gültiger Index. Warum wird trotzdem kein Folgezustand erzeugt?

Lösung:

(a) `(0, 1, 2, 3, 4, 5, 6, 7, 8)`
`(3, 1, 2, 6, 4, 5, 0, 7, 8)`
`(3, 1, 2, 4, 0, 5, 6, 7, 8)`

(b) Die Leerstelle steht an Index 3, also links in der mittleren Zeile. Index 2 ist das Feld rechts oben. Es liegt weder in derselben Spalte ($3\%3 = 0$, $2\%3 = 2$) noch in derselben Zeile ($3//3 = 1$, $2//3 = 0$).

10. (3 Punkte) Ein String aus den Zeichen M und O soll mit möglichst wenigen Operationen in den String MOO umgewandelt werden. Erlaubt sind: den ersten oder letzten Buchstaben durch den anderen ersetzen (M durch O oder O durch M) und den ersten oder letzten Buchstaben löschen.

   (a) Gib alle Folgezustände von MOMO an.

   (b) Wie viele Operationen braucht man mindestens für OMOMO? Gib die Zwischenschritte an.

   (c) Warum kann MMM nie zu MOO werden?

Lösung:

(a) OOMO, MOMM, OMO, MOM

(b) 3 Operationen: OMOMO → MOMO → MOM → MOO

(c) Der mittlere Buchstabe M kann nicht geändert werden, man kann nur den ersten und den letzten ändern. Löschen hilft nicht, denn danach ist der String kürzer als MOO und kann nicht mehr länger werden.

11. (3 Punkte) Es gibt einen 3-Liter-Krug und einen 5-Liter-Krug, beide sind leer. Ein Zug ist: einen Krug ganz füllen, einen Krug ganz leeren oder Wasser von einem Krug in den anderen gießen, bis der eine leer oder der andere voll ist. Ein Zustand ist das Paar (Liter im 3-Liter-Krug, Liter im 5-Liter-Krug).

   (a) Gib alle Folgezustände von (3, 2) an.

   (b) Wie viele Züge braucht man mindestens, bis im 5-Liter-Krug genau 4 Liter sind? Gib die Zustände an.

Lösung:

(a) (3, 5), (0, 2), (3, 0), (0, 5)

(b) 6 Züge: (0, 0) → (0, 5) → (3, 2) → (0, 2) → (2, 0) → (2, 5) → (3, 4)

12. (4 Punkte) Auf der Westseite eines Flusses sind ein Mensch, ein Wolf, eine Ziege und ein Kohl. Nur der Mensch kann die Fähre steuern und höchstens ein Tier oder den Kohl mitnehmen. Ohne den Menschen dürfen Wolf und Ziege nicht zusammen auf einer Seite bleiben, Ziege und Kohl auch nicht. Alle sollen auf die Ostseite.

    (a) Wie kann man einen Zustand modellieren? Gib Start- und Zielzustand an.

    (b) Gib alle erlaubten Folgezustände des Startzustands an.

    (c) Wie viele Überfahrten sind mindestens nötig? Gib eine mögliche Reihenfolge an.

Lösung:

(a) Als Tupel mit der Seite von Mensch, Wolf, Ziege und Kohl, 0 für Westen und 1 für Osten. Start: (0, 0, 0, 0), Ziel: (1, 1, 1, 1)

(b) Nur (1, 0, 1, 0). Fährt der Mensch allein, mit dem Wolf oder mit dem Kohl, bleibt die Ziege mit dem Kohl bzw. dem Wolf allein.

(c) 7 Überfahrten: Mensch mit Ziege nach Osten, Mensch allein nach Westen, Mensch mit Wolf nach Osten, Mensch mit Ziege nach Westen, Mensch mit Kohl nach Osten, Mensch allein nach Westen, Mensch mit Ziege nach Osten

13. (5 Punkte) Auf der Westseite eines Flusses sind 3 A und 3 B. In das Boot passen höchstens 2, bei jeder Überfahrt muss mindestens ein A oder B im Boot sein. Ist auf einer Seite mindestens ein A, dann dürfen dort nicht mehr B als A sein. Alle sollen auf die Ostseite.

    Löse die Aufgabe mit der Breitensuche aus dem Unterricht (`bfs`, `reconstructPath`). Modelliere einen Zustand und schreibe die Funktionen `nextstates` und `goaltest`. Wie viele Überfahrten sind mindestens nötig?

Lösung:

```
def erlaubt(a, b):
    return a == 0 or b <= a

def nextstates(state):
    a, b, boot = state              # A und B auf der Westseite, boot: Boot im Westen?
    tmp = []
    for da, db in [(1, 0), (2, 0), (0, 1), (0, 2), (1, 1)]:
        if boot:
            a1, b1 = a - da, b - db
        else:
            a1, b1 = a + da, b + db
        if 0 <= a1 <= 3 and 0 <= b1 <= 3 and erlaubt(a1, b1) and erlaubt(3 - a1, 3 - b1):
            tmp.append((a1, b1, not boot))
    return tmp

def goaltest(state):
    return state == (0, 0, False)

prev, goal = bfs((3, 3, True))
path = reconstructPath(prev, goal)
print(len(path) - 1, path)
```

Man braucht 11 Überfahrten.

14. (5 Punkte) Gegeben sind drei Strings `s1`, `s2` und `s3` aus den Zeichen 1 und 2. Ein Zug besteht darin, das letzte Zeichen eines Strings zu entfernen und hinten an einen anderen String anzuhängen. Ist das neue Zeichen gleich dem bisher letzten Zeichen, verschmelzen die beiden zu einem.

```
s1, s2, s3 = '121', '1', ''
# Zug: letztes Zeichen von s1 an s2 anhängen
s1, s2, s3 = '12', '1', ''    # die beiden 1 sind verschmolzen
```

Ziel ist, dass `s1` und `s2` nur noch aus einem Zeichen bestehen und `s3` leer ist.

Modelliere einen Zustand und schreibe die Funktionen `nextstates` und `goaltest` für die Breitensuche aus dem Unterricht. Wie viele Züge braucht man mindestens für `s1, s2, s3 = '121', '12', ''`?

Lösung: Ein Zustand ist das Tupel `(s1, s2, s3)`.

```
def nextstates(state):
    tmp = []
    for i in range(3):
        for j in range(3):
            if i != j and state[i] != '':
                a = list(state)
                c = a[i][-1]
                a[i] = a[i][:-1]
                if a[j] == '' or a[j][-1] != c:
                    a[j] = a[j] + c
                tmp.append(tuple(a))
    return tmp

def goaltest(state):
    s1, s2, s3 = state
    return len(s1) == 1 and len(s2) == 1 and s3 == ''

prev, goal = bfs(('121', '12', ''))
path = reconstructPath(prev, goal)
print(len(path) - 1, path)
```

Man braucht 5 Züge, z.B. (121, 12, -) → (12, 12, 1) → (1, 12, 1) → (-, 12, 1) → (2, 1, 1) → (2, 1, -) (- steht für einen leeren String).