Aufgabe 3: Wortsuche

Team-ID: 00155

Team: Opensourcehacker

Bearbeiter/-innen dieser Aufgabe:
Ron Jost

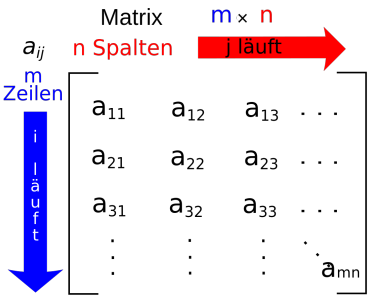
22. November 2021

Inhaltsverzeichnis

Lösungsidee

Ich habe zur Lösung der Aufgabe vier verschiedene Schwierigkeitsstufen definiert, die ein Wortsuchrätsel aufweisen kann, und nach denen dann jeweils das Programm das Rätsel erstellt. Zur Repräsentation des Wortsuchrätsels habe ich einen 2d Array kreiert, der wie folgt aufgebaut ist:

Der Wert m ist die Anzahl der Zeilen, welche aus der Textdatei ausgelesen wird, der Wert n die Anzahl der Spalten, welche ebenso aus der Textdatei ausgelesen wird.



Zu Beginn wird die Matrix, der 2d Array mit ebenjenen Größen kreiert und mit Nullen aufgefüllt. Die Schwierigkeitsstufen, nach denen die Matrix aufgefüllt wird sind folgende:

- Einfach: Nur vertikale und horizontale Ausrichtungen sind der Wörter erlaubt.
- Mittel: Vertikale, horizontale, und von links nach rechts verlaufende diagonale Ausrichtungen der Wörter sind erlaubt.
- Schwer: Vertikale, horizontale, von links nach rechts verlaufende Diagonalen sind erlaubt. Ebenso ist es die Wörter rückwärts zu schreiben. Somit also auch Diagonalen von rechts nach links.

- Extrem: Die möglichen Ausrichtungen der Stufe Schwer, jedoch wird der 2d Array nicht mit vollständig zufälligen Buchstaben aufgefüllt, sondern mit Fragmenten der einzelnen Wörter.

Je nachdem, welche Schwierigkeitsstufe gewünscht ist, stehen die pro Stufe aufgeführten Richtungsoptionen zur Verfügung und es wird für jedes Wort, welches untergebracht werden soll, eine Richtung zufällig ausgesucht. Generelle Voraussetzung der Eintragung eines Wortes in den 2d Array ist es, dass nie ein Buchstabe des jeweilig einzutragenden Wortes einen anderen Buchstaben, welcher nicht dem Buchstaben des Wortes entspricht zu überschreiben. Deswegen wird der Array zu Beginn nur mit Nullen aufgefüllt und nicht direkt mit zufälligen Buchstaben.

Horizontale Ausrichtung

Für die Horizontale Wörterplatzierung wird für den Startpunkt(x_{start}, y_{start}) x zufällig ein Wert, zwischen 0 und der Zeilenanzahl m, ausgewählt. Für y_{start} wird zufällig ein Wert, zwischen 0 und der Spaltenanzahl n – die Länge des Wortes, gewählt. Daraufhin wird das Wort mithilfe der Veränderung der y_{start} Koordinate, um $y_{start} = y_{start} + 1$ pro Buchstabe in den 2d Array geschrieben, vorausgesetzt, die oben formulierte, global geltende Regel, wird nicht verletzt. Sollte im Falle der Schweren oder Extrem Schwierigkeitsstufe, Horizontal und rückwärts gewünscht werden, ändert sich am Verfahren nichts, außer das vor dem Einsetzen, das Wort umgekehrt wird.

Vertikale Ausrichtung

Der Startpunkt (x_{start}, y_{start}), x_{start} zufällig ein Wert, zwischen 0 und m – die Länge des Wortes, ausgewählt. Für y_{start} wird zufällig ein Wert, zwischen 0 und n, gewählt. Daraufhin wird das Wort mithilfe der Veränderung der x_{start} Koordinate, um $x_{start} = {}_{xstart} + 1$ pro Buchstabe, in den 2d Array geschrieben, vorausgesetzt, die oben formulierte, global geltende Regel, wird nicht verletzt. Sollte hier eine Umkehrung der Vertikalen Richtung gewünscht sein, bleibt das Verfahren gleich, nur das Wort wird vor der Eintragung einmal umgekehrt.

Diagonale Ausrichtungen

Es gibt vier verschiedene Diagonale Ausrichtungsmöglichkeiten, welche sich in zwei Kategorien unterteilen lassen.

Die erste Kategorie umfasst die Diagonalen von links nach rechts. Es ist möglich, von unten nach oben oder von oben nach unten die Diagonale zu konstruieren. Der Startpunkt für die Diagonale von ***oben nach unten*** (x_{start}, y_{start}) wird gewählt indem man x_{start} einen Wert zwischen 0 und m – Länge des Wortes und für y_{start} einen Wert zwischen 0 und n – Länge des Wortes zuweist. Das Wort wird mithilfe der Veränderung beider Koordinaten eingesetzt, x ändert sich immer um +1 pro einzutragenden Buchstaben, y ebenso. Der Start bei von ***unten nach oben*** (x_{start}, y_{start}) wird gewählt indem man x_{start} einen Wert zwischen m – Länge des Wortes und m und für y_{start} einen Wert zwischen 0 und n – Länge des Wortes zuweist. Das Wort wird mithilfe der Veränderung beider Koordinaten eingesetzt, x ändert sich immer um -1 pro einzutragenden Buchstaben, y um +1.

Die zweite Kategorie umfasst die Diagonalen von rechts nach links. Der Startpunkt für die Diagonale von ***oben nach unten*** (x_{start}, y_{start}) wird gewählt indem man x_{start} einen Wert zwischen Länge des Wortes und m, für y_{start} einen Wert zwischen n – Länge des Wortes und n zuweist. Das Wort wird mithilfe der Veränderung beider Koordinaten eingesetzt, x ändert sich immer um +1 pro einzutragenden Buchstaben, y um -1. Der Start bei von ***unten nach oben*** (x_{start}, y_{start}) wird gewählt indem man x_{start} einen Wert zwischen Länge des Wortes und m und für y_{start} einen Wert zwischen Länge des Wortes und n zuweist. Das Wort wird mithilfe der Veränderung beider Koordinaten eingesetzt, x ändert sich immer um -1 pro einzutragenden Buchstaben, y um -1 ebenso.

Sollte die Wahl des Nutzers auf den Schwierigkeitsgrad Extrem fallen, wird der 2d Array nach dem auffüllen mit allen Worten nicht mit zufällig ausgewählten Großbuchstaben aufgefüllt, sondern nur mit Buchstaben, die in den zu suchenden Wörtern zu finden sind.

Umsetzung

Das Programm wird in python implementiert. Zu Beginn wird per Kommandozeilenparameter die Wörterliste und der gewünschte Schwierigkeitsgrad übergeben. Daraufhin wird je nach gewünschten Schwierigkeitsgrad die gleichnamige Funktion aufgerufen. Diese habe jeweils als Parameter m, also die Zeilenanzahl, n, die Spaltenanzahl und die Wörterliste. Alle Funktionen sind gleich aufgebaut, einzig und allein die Menge der möglichen Richtungen unterscheidet die funktionalen Schwierigkeitsstufen. Zu Beginn einer jeden Funktion wird per list comprehension der 2d Array mit den gewünschten Dimensionen erstellt und mit Nullen gefüllt. Daraufhin wird in einer großen while Schleife, solange es möglich ist ein Wort zu platzieren, für jedes Wort zufällig eine Richtung, aus den, der Schwierigkeitstufe möglichen, Richtungen ausgewählt und es dann mit Hilfe der oben beschriebenen Regeln der jeweiligen Richtung und der global gültigen Regel, dass kein Buchstabe eines Wortes, einen anderen, nicht identischen, überschreiben darf, in den 2d Array eingetragen wird. Sollte es nicht möglich sein, ein Wort innerhalb des 2d Arrays zu platzieren, wird der 2d Array neu erstellt und die Funktion beginnt von neuem.

Beispiele

```
hacker5preme:~/BWINF-40/Runde-1/Aufgabe-3$ python3 Aufgabe3.py worte0.txt Einfach
                VBTNW
                OFOWN
                RRADY
                GXEVA
                TORFL

Wörter versteckt:
VOR, RAD, EVA, TORF
hacker5preme:~/BWINF-40/Runde-1/Aufgabe-3$ python3 Aufgabe3.py worte0.txt Mittel
                PGEVA
                QXDFV
                KARNN
                ROBKF
                TVORG

Wörter versteckt:
VOR, RAD, EVA, TORF
hacker5preme:~/BWINF-40/Runde-1/Aufgabe-3$ python3 Aufgabe3.py worte0.txt Schwer
                DREGP
                TOWVD
                GORAA
                AORQG
                VCZFE

Wörter versteckt:
VOR, RAD, EVA, TORF
hacker5preme:~/BWINF-40/Runde-1/Aufgabe-3$ python3 Aufgabe3.py worte0.txt Extrem
                FVFRF
                DERRR
                VORAO
                TAVDT
                VEEVE

Wörter versteckt:
VOR, RAD, EVA, TORF
```

```
hacker5preme:~/BWINF-40/Runde-1/Aufgabe-3$ python3 Aufgabe3.py worte2.txt Einfach
                  FIKBWEKXDH
                  EATIWTCDBF
                  SFALIDOVHT
                  TDSDMZMDVM
                  PWTSAUPMGS
                  LJACUSUNIR
                  AZTHSBTVSS
                  TIUIWVEYNZ
                  TVRRBQRSTT
                  EXSMXEPIMC

Wörter versteckt:
MAUS, TASTATUR, BILDSCHIRM, FESTPLATTE, USB, COMPUTER
hacker5preme:~/BWINF-40/Runde-1/Aufgabe-3$ python3 Aufgabe3.py worte2.txt Mittel
                  CUUBPMDTRC
                  FESTPLATTE
                  XTASTATURP
                  COMPUTERZX
                  AMUYCGUXSN
                  NRAAPCLQLB
                  CCOUSBAIPD
                  BILDSCHIRM
                  QRETXBFKQN
                  KIORXPJLIO

Wörter versteckt:
MAUS, TASTATUR, BILDSCHIRM, FESTPLATTE, USB, COMPUTER
hacker5preme:~/BWINF-40/Runde-1/Aufgabe-3$ python3 Aufgabe3.py worte2.txt Schwer
                  MRIHCSDLIB
                  IXZZZNWUHO
                  QPHEMUNZSB
                  VYPNNSPBFB
                  OKIQWMAUSI
                  RUTATSATHK
                  OOVKHUSRCU
                  FESTPLATTE
                  LPRNHCRBDV
                  LCOMPUTERG

Wörter versteckt:
MAUS, TASTATUR, BILDSCHIRM, FESTPLATTE, USB, COMPUTER
```

```
hacker5preme:~/BWINF-40/Runde-1/Aufgabe-3$ python3 Aufgabe3.py worte2.txt Extrem
                HDICRAAMEB
                HORLUHLRTC
                OFLHTDRITO
                UIOHAMMHAM
                SUSBTABCLP
                AOUDSUTSPU
                DBMBASODTT
                DPORTRPLSE
                FFUPHOUIER
                OOFOADHBFH

Wörter versteckt:
MAUS, TASTATUR, BILDSCHIRM, FESTPLATTE, USB, COMPUTER
hacker5preme:~/BWINF-40/Runde-1/Aufgabe-3$ python3 Aufgabe3.py worte3.txt Einfach
                COSRYCFJRFRFFJLXBEXWDXDU
                SSTHDFNCGHUIMQEGVMRTYJQE
                EGETSCSOBKEEJDGNEJBPPVJA
                NFMDMQBPSEDFEEICRBBWVQYA
                EJYEJXHYDMEROOTOSHOHUVWB
                QXMCFMHEBBHEVUIHBCBSPLNM
                XXRALDGJGYKVCRMTXWMLZXMU
                NIGJHUQDOVROZEAEHVLMJNIB
                MKQUWAJENUVLNFTAIJMOBTMU
                ALRXEFTKBMHUJEIYNWQNBSPN
                OTNWRQVOBZKTQROZFELOBLSL
                VIEEDZXRWTCIBANUEKDGCFJM
                FOMMLQBATIXODTFLKOSRANUM
                SLIVJGNTGNCNAZZRTNGAGJNE
                VSSIWWRIWTTTHVGFIJTMDTRM
                ZLSYOBOOAUOINRZKOUMMXVYP
                TTITHYVNWIOSNNGENNNBJZCA
                JLORZMEYMTYDXUDPRKJLJVCT
                CQNHQBSAMIYNKPRUUTCXEJCH
                LRQCRBLSSOEGKZRWKUNJKVUI
                RVHJQGKFJNFKDUXSORLJNWUE
                UEUILQPGWBLHEGCHRONIKFWB

Wörter versteckt:
INTUITION, INFEKTION, MONOGRAMM, REVOLUTION, KONJUNKTUR, CHRONIK, EMISSION, DEKORATION, LEGITIMATION, EMPATHIE, REFERAT, VERS
hacker5preme:~/BWINF-40/Runde-1/Aufgabe-3$
```

```
hacker5preme:~/BWINF-40/Runde-1/Aufgabe-3$ python3 Aufgabe3.py worte3.txt Mittel
                IPRVVQXVGKSBVHKENBAEWLMB
                OCWFEJLDLTAUALLSGNNBSXPP
                YSVMTCNADLIUEMPATHIEJKWT
                RSKUOWKUTTLZKMILGOKZJTJD
                MKONJUNKTURGEFDPHIIKZBWA
                RPVQVFMIDDMTLEGITIMATION
                OZEYELTYVOQUITBZNLBCHYDB
                GKRFAATMIPSYDXREVOLUTION
                BHSRRFKQPZYCJSIAJJLQMDOC
                JHZEHBEBNDCUNPQMLRIMPIED
                SNFKYXJYNMDITZMSCLAWONOL
                REENLLMDHLOQNAWCMHKYWTEJ
                RBLVGSPAEGRVRFPYFEREJUMU
                AROPZDITGKKGGQEFQSXOJIHN
                EMISSIONKDOHSJRKKOCTNTRW
                XMKRVVJFANHRYUMYTRLGOIPO
                XYQIFXVJOIKZAPDEGIOAGOKI
                TVUUVRHMOVRODTAQYSOTLNLD
                HCORNICRNUVNJRIRVQTNQWNX
                CWMWTGFBUMVMUDCOLYPEGSWB
                SOYOVWZPHDPJNGKANUPMVPCO
                CNDNXHMKICFNLXPFAWDEWDKS

Wörter versteckt:
INTUITION, INFEKTION, MONOGRAMM, REVOLUTION, KONJUNKTUR, CHRONIK, EMISSION, DEKORATION, LEGITIMATION, EMPATHIE, REFERAT, VERS
hacker5preme:~/BWINF-40/Runde-1/Aufgabe-3$ python3 Aufgabe3.py worte3.txt Schwer
                KTDDJWSWBPEFDLOXSJMSZDRX
                OEWBDONGDXAGLIFRUUABNYZX
                TMNZPZISWCMREVOLUTIONOHS
                RINFEKTIONNJCEIPAHELZLSG
                NXRKKDKKVQDAAJLDOISIAZVL
                VWMIKGMEJKEHDMXNNBKZTQZT
                KXQNBDRACIKKJFSONDISQXVE
                CLLOQSPCGROGAVIGOQVVCVNO
                GBFRTUUIGDRKKTZHUSHEWSJU
                QIDHEIPILNANAYCUQRXMGADT
                XKFCRJBNGYTMEVHYUKCIRNBW
                DOZBESOODLINTUITIONSRVRB
                KZWLFLJSVTOGUOKYRDYSEVQU
                TSGLEINZIXNPNNZVCPXIXDEH
                ZRKFRYQGIMUDUKTWMZNODWAT
                LFBOAXEJUOTJLNHMGAGNSXSK
                GHKOTLRLECNDTLTKWQZXPYNC
                JCLGPIHNYOIHXCJEKTLSMHGG
                RDKDQRQMKYYUWMEMPATHIEQC
                ULBSYIHHCRGVMMARGONOMSSN
                XUAOQXOBQNQUOKUPFLLPAHQY
                RSFKPSSNHVKGXGZENMXMYLVX

Wörter versteckt:
INTUITION, INFEKTION, MONOGRAMM, REVOLUTION, KONJUNKTUR, CHRONIK, EMISSION, DEKORATION, LEGITIMATION, EMPATHIE, REFERAT, VERS
```

```
hacker5preme:~/BWINF-40/Runde-1/Aufgabe-3$ python3 Aufgabe3.py worte3.txt Extrem
               TPSIDIOAHIVCAIACGUHARMII
               AGVCKFPRSIECTSLMKFGGCVMV
               RPTFMNLJDVUANAVDOVSCOKLK
               ECUFKHFDFOPRJOERMNATNKNI
               FINMUUKMEAKGMENJSUOCAOIN
               EUPTSEPCAKAUMPPVIROGOVSO
               RHTEDVKTEPOIEMPATHIERIMR
               TENACIOAMTSRJUHGGUFEVAPH
               KCDUJLNPOSFDACREFAEUANMC
               VNNGLCJSIVPHITJHGOROHUAM
               OCVHSIUOROOIRDINGEPILLHI
               KSSNKFNNMEPMNICOVJVKKERJ
               KMRPTSKLESVONNKONFUIHGOH
               MFPVMPTATFIDATLKJFUHGIDK
               PRIDJKUHTTJPSUCSGSTECTFN
               NSKEPERAKFCJTIAVORVDOINI
               EMKMMIPESDGIHTOKMODSAMIP
               FKLSPGFJAGOJPIIVSDHOHATF
               TGTHANNNINGJTOEFVJHNFTLD
               SVLGITTRNEKGGNPSCJUJNIEH
               GVFRHATLKICTLMLKFEFHSOHS
               TKMCIHSRMTVPDPOHHRHINNEV

Wörter versteckt:
INTUITION, INFEKTION, MONOGRAMM, REVOLUTION, KONJUNKTUR, CHRONIK, EMISSION, DEKORATION, LEGITIMATION, EMPATHIE, REFERAT, VERS
```

```
hacker5preme:~/BWINF-40/Runde-1/Aufgabe-3$ python3 Aufgabe3.py worte4.txt Einfach
                ZTTAXOBOXKWIKTIONARYFNZFBOOLANDP
                COLNEFGYFTMUSIKCHARTSPZMQGXNGZPE
                NLINAVIGATIONSLEISTEPNROWYBKZIIR
                PAXUTKVDJKGFWYMKISSMILEYEACHBSNL
                GRUZYINOCACKZPQWQNJRZWWSYIGIBOGR
                ECÖSTERREICHBEZOGENCOQTUGNGNKRVP
                AHMIXLSFNGIADGIFARBLEGENDEZALTVL
                WIGDQDVMFWPROBMEDAILLENSPIEGELBX
                ZVFREALLMUSICDZXKBEGRIFFSKLÄRUNG
                XPBGVFJKATEGORIEGRAPHMDICQEBQAOO
                GZARLRVHGLUQOKCOMMONSCATLVFAGXLQ
                WICMBGFDQIIOPALKBWLHUGXEBGFBDMAI
                REHYBENUYZVFMLUGNPWMUVOFEKEERGRP
                ORTCNRIYQEJGBEAQALFZPEBBGXMLPICA
                AMULKRFSGNRLSNUCVLUIEBIERXMPJMHH
                OPNMJWORUZYCNDSWFHSTRANLISLOJDIZ
                LOGUZQUTMUBNLENIRTSALUFEFLWSRBVC
                NLCLFEGGQMBGRRGKAABTESOGFNTIDABH
                FGCTQYEJLSVWVSGIMNAIDTRESAFTQBON
                OZOIONOAZTXAATUDEJLOIEMEKMFIRRTQ
                LLOLPHQBNEYPUIBAAQLNGLAMLBFOWUJL
                GFRIIJUSGLUPDLKTSDDBTLTCÄGGNJFDU
                ERDNUSEALLREYASAPCAQWEIKRVFSVDIQ
                NAIGAJLTKUDNGLPICETEWIONUMRKLASN
                LGNUEOLZDNDRCAECDNEBPNNENBXAUTKH
                ENAAHQEBJGXEXNRKITNNSTCMGMORAUUO
                IKTLCANGNISCRGSABEPXZEQXSZPTZMSO
                SGEMSFILMNFHNJOSYRAMBRNUHCWETRSJ
                TXMKPKGRWNZTMYNTNLOJVNBFISBUIDIK
                EUAWIKISOURCEDEEOCDOIEZQNVNGPSOZ
                WVPHLZGSBGHQXWNNCHTNATRLWJNGQJNB
                ELJZTMLYDJMMISLLOAKLXQTGEURJVCSE
                BILBIBRECORDJSERMREKKUCEIMTWHUSN
                ATKEHAUTFOINEHIRMTJOCEYFSETJNTEU
                REIQSGHWVUNLYÖSLOSFKLLHXBRZWUFIT
                CRTPPOCDOXFQSHTNNWATPLEIZCDWYNTZ
                HAUCOMMONSOSIEEISFIOFEIPOLDEUYEE
                ITYQXCOORDINATEARCHIVIERUNGPWRJR
                VUZFMDWKIWZITGQBELJLLKACCSJDKEZA
                ORAAUTOARCHIVHRYHTEXTBCSOWZPPDVU

Wörter versteckt:
ABRUFDATUM, ABSATZ, ACHTUNG, ALLMUSIC, ARCHIV, ARCHIVBOT, ARCHIVIERUNG, AUS, AUT, AUTOARCHIV, BABEL, BAUSTELLE, BBKL, BEGRIFFSKLÄRUNG, BEGRIFFSKLÄRUNGSHINWEIS, BEL, B
ELEGE, BENUTZER, BGR, BIBRECORD, BOOLAND, CAN, CENTER, CHARTS, COL, COMMONS, COMMONSCAT, COORDINATE, COORDINATEMAP, DDB, DEU, DISKUSSIONSSEITE, DOI, ERLEDIGT, FARBLEG
ENDE, FILM, FN, FNZ, FOLGENLEISTE, FRA, FUSSBALLDATEN, GEOQUELLE, GER, GNIS, HÖHE, IMDB, INFO, INFORMATION, INTERNETQUELLE, IPA, KALENDERSTIL, KASTEN, KATEGORIEGRAPH,
 LANG, LITERATUR, LIZENZUMSTELLUNG, MEDAILLENSPIEGEL, MULTILINGUAL, MUSIKCHARTS, NAVFRAME, NAVIGATIONSLEISTE, NOCOMMONS, ÖSTERREICHBEZOGEN, PERSONENLEISTE, PING, POSI
TIONSKARTE, PRO, SMILEY, SORT, TAXOBOX, TEXT, WAPPENRECHT, WEBARCHIV, WIKIDATA, WIKISOURCE, WIKTIONARY, ZITATION
```

```
hacker5preme:~/BWINF-40/Runde-1/Aufgabe-3$ python3 Aufgabe3.py worte4.txt Mittel
                DGOAKPVJIAAIISPERSONENLEISTEKYUM
                IIPBXRTQBWSCYKKZMRZWTSGFRWMIWDDB
                DJTSEAEGWWSPCGAOPTAXOBOXUZKMEZUE
                IHYATWGKCKALENDERSTILMIXEGHUOAAG
                NMWTRNHWSBUCFOLGENLEISTENAVFRAME
                TWIZEDPIFZNKFRPXKABRUFDATUMXKAWP
                EAKXSZGKNUSMILEYDHCPDVPXDPNAALJC
                ROIUAFJTMOSPXQGEOQUELLEGREEXSLOS
                NLDKGZFICNCSCTLITERATURIGBHLTMDO
                ELAICKQOHWAOBZGQYBGRUSPOGXEGEUGQ
                TITRFAWNRIAVMAEGPDDOQSZCCBOANSBC
                QNADNCNALKRBIMLZROURHEFEAABCIIZG
                UFTIZZJRUICFKGOLXUAHBPWBPOJBUCSH
                EOASFOLYASHVIMANDMUHQBBCFQXBUTVG
                LRVKBBKLJOIIXWJTSACVPPMUGSLTRANG
                LMPUJNTCDUVFQQENIITOCLXNMEBASUUB
                EADSFLOBTRJIPVOBEOFEABIMGPHGRBVE
                XTDSAJAXBCATCMWRANNUNPOEHCCÄWIJL
                DIYIRAENSEAHMZRPIRGSZSIARWLJDNWE
                POOOBTGOGAGOVEQXUNCELPMEAKMNTLEG
                RNANLHMGHCCOTFRXIFPHSEZZSISHAOAE
                OCRSEMQRJLMSKGHLUACNITIFKVLATPRZ
                IDCSGAZXZNÖFJGIOMBEOUVFSGEWUVWCA
                ZKHEEUKBRFFSGTAEKLSNMIWOTLVSONHA
                ISIINTDAHÖHELVTFLXEORMCFLEJTCMIN
                TGVTDOVOJUEUQAAIIBZGRJOORFOEEUVI
                ASBEEAQYISMHNFABKSEEOTMNLXLANSIH
                TSOZAROZIFNIADDXSBXCQQMYSRQLTIEF
                IDTACCDNYBDLEBBAUSTELLELVCUZEKRH
                OSEUQHGAVRVMQFIACHTUNGIPACAMRCUT
                NIFUPIYAOTRIPOSITIONSKARTEXTGHNL
                NSOLCVWOKULNHAFMZJPERLEDIGTJZAGY
                AMRVTVCYCFEDULIZENZUMSTELLUNGRYA
                AECOORDINATEQOMQBDMONFILMAUTFTUS
                GBEGRIFFSKLÄRUNGSHINWEISDAXGFSWW
                NEBEMAZFUJKOYTVZDEFDOICAQHAORWFU
                ALUKHCRCHZUABNLXAPVZYVCVKYCMANLJ
                WAPPENRECHTIMDBMVRXCZGZSTDPIHRJD
                QUPMMDEIVBIBRECORDXRXQYBWPAHBCEP
                IKGBHDKATEGORIEGRAPHFBOOLANDXBKG
```

```
hacker5preme:~/BWINF-40/Runde-1/Aufgabe-3$ python3 Aufgabe3.py worte5.txt Einfach
                YJLYPDGKCRNGGDBCVCLMTEQXPVJDQH
                ZANHRQPPWQYXZPDNPXQKPFEACIJBLR
                MFYSVTTOQMIQLKQEECJQEPXSRLUWDJ
                YQQZHMUEAHELJDHGBMQWIQYHKRVPVK
                TSMLHQAAZCDWUBIWJUBSVXIGJWQISM
                XGPAEYLQEPKEBODTJUYZPWVEXHYLHM
                JJWVDJETNKTZITCSWPAMZYDBETCXLM
                VYXIVCMYXIBRRMTOXYDODPPRJNBFPN
                VIHVRBRJGFFVLQEZDVWNOEDGAUPNXX
                TCTHPHKMTUTDAJHPTWVZSIIFEKRGBU
                UASMTZFDPNIFPEVFRJXWLVNUMGPZWD
                NAMRWWTBYFLPLTHLZFPITPQFHHWBMJ
                ARGQRPZZRRJFBHLGQAYANSQITPIAAB
                LFHYWZJUGCUPHNULDEZZGBRDYMVRLO
                GBNQEMNLELGDTRSNYSPXUTGMKVRADQ
                XUHJCMYCPTIGZAYPWLGPRGERZJZEAM
                FFPXVVLFQRRCJXBQYMNFRELALETGSP
                AAUVQKCBPIRPBSTYTICUXQCZWVRCRR
                DIRLDTZSUSVWRLTAAENLHNAIHCRVFK
                KFVKJSJJRQITEKWRVJTLZJKILKEZVB
                FRZYUYGHUQEXMSKJLJAPIBBBHAYMWQ
                ELCZDNOQYGIZNRVSDWCTCPUUPYNOVF
                ALIVUWCIIWYTUZIQLLYWGUDKDPWGXQ
                UAKKQXQQBALYOILMQPLLIGSXZKJNRN
                YIYDRKDKASVTAAOHBMCMJWHOKAVATZ
                PYFQPFMJWVLDGUIWCEYDVRGCGJGAJC
                QUFKNXTQEWVGCCZYHQAEXLXZRLJZCO
                UXAFZQTOJYWPRARAWUWINKVJCUFRXO
                GJUTMEKHRPNYHGXYGUPCEAATGEDHTG
                ZFOZNGDLUHEZGNJLJMIVINRMWZIMIY

Wörter versteckt:
DAS
```

Quellcode

```
# BWINF: Nr. 40
# Runde: 1
# Aufgabe: 3
# Autor: Ron Jost
# Team: Opensourcehacker (00155)

import random
import sys
import string
import time


# Dateneinlese
worte_datei = open(str(sys.argv[1]), 'r')
lines = worte_datei.readlines()
worte_datei.close()
```

```
lines_bearbeitet = []
for line in lines:
    lines_bearbeitet.append(line.replace('\n', ''))

ZeilenNr = int(lines_bearbeitet[0][:lines_bearbeitet[0].find(' ')])
SpaltenNr = int(lines_bearbeitet[0][lines_bearbeitet[0].find(' ') + 1:])
WörterNr = lines_bearbeitet[1]
Wörter_Liste = []
for i in range(1, int(WörterNr) + 1):
    Wörter_Liste.append(lines_bearbeitet[1 + i])

# 1. Einfach: Nur Horizontal und Vertikal
def Einfach(ZeilenNr, SpaltenNr, Wörter_Liste):
    Wörter_Liste = sorted(Wörter_Liste, key=len)
    Wörter_Liste = Wörter_Liste[::-1]
    # Konstruktion des grids, dem leeren Wortsucherätsel
    grid = [['0' for _ in range(int(SpaltenNr))] for _ in range(int(ZeilenNr))]
    Richtungen = ['horizontal', 'vertikal']
    Error = False
    # Sollte es unmöglich sein, ein Wort unterzubringen, wird wieder von vorne gestartet, mit allen
Worten
    while Error == False:
        try:
            for Wort in Wörter_Liste:
                Wortlänge = len(Wort)
                platziert = False
                t_end = time.time() + 5
                while not platziert:
                    if time.time() < t_end:
                        failed = False
                        #Zufällige Entscheidung, welche Richtung genommen wird.
                        Richtung = random.choice(Richtungen)

                        if Richtung == 'vertikal':
                            step_Zeile = 1
                            step_Spalte = 0
                            maximum_Zeile = ZeilenNr - Wortlänge
                            Zeilen_position = random.randrange(0, maximum_Zeile + 1)
                            Spalten_position = random.randrange(0, SpaltenNr +1)

                        if Richtung == 'horizontal':
                            step_Zeile = 0
```

```
                step_Spalte = 1
                maximum_Spalte = SpaltenNr - Wortlänge
                Zeilen_position = random.randrange(ZeilenNr +1)
                Spalten_position = random.randrange(maximum_Spalte + 1)

            for i in range(Wortlänge):
                # Das Wort wird Buchstabe für Buchstabe mit den Werten in der Tabelle
verglichen
                Buchstabe = Wort[i]
                neu_Zeile = Zeilen_position + i * step_Zeile
                neu_Spalte = Spalten_position + i * step_Spalte
                Buchstabe_neue_Position = grid[neu_Zeile][neu_Spalte]
                if Buchstabe_neue_Position != '0':
                    if Buchstabe_neue_Position == Buchstabe:
                        continue
                    # Wenn der Buchstabe an der neuen Position x keine 0 sondern ein anderer
Buchstabe,
                    # als der einzusetztende ist, muss abgebrochen werden und das Wort muss neu
platziert werden
                    else:
                        failed = True
                        break
            if failed:
                continue
            else:
                # Das Wort je nach gewählter Richtung Buchstabe für Buchstabe eingetragen
                for i in range(Wortlänge):
                    Buchstabe = Wort[i]
                    neu_Zeile = Zeilen_position + i * step_Zeile
                    neu_Spalte = Spalten_position + i * step_Spalte
                    grid[neu_Zeile][neu_Spalte] = Buchstabe
                platziert = True
                break

    # Die restlichen Leerstellen werden mit zufälligen Großbuchstaben aufgefüllt
    for x in range(ZeilenNr):
        for y in range(SpaltenNr):
            if grid[x][y] == '0':
                grid[x][y] = random.choice(string.ascii_uppercase)

    # Ausgabe des Wortsucherätsels
    for x in range(ZeilenNr):
```

```
            print('\t' * 2 + ''.join(grid[x]))
        Error = True
    except IndexError:
        grid = [['0' for _ in range(int(SpaltenNr))] for _ in range(int(ZeilenNr))]

# 2. Mittel: Horizontal, Vertikal und diagonal von links nach rechts von unten nach oben und von oben nach unten:
def Mittel(ZeilenNr, SpaltenNr, Wörter_Liste):
    Wörter_Liste = sorted(Wörter_Liste, key=len)
    Wörter_Liste = Wörter_Liste[::-1]
    # Konstruktion des grids, dem leeren Wortsucherätsel
    grid = [['0' for _ in range(int(SpaltenNr))] for _ in range(int(ZeilenNr))]
    Richtungen = ['vertikal', 'horizontal', 'diagonal-lr-down', 'diagonal-lr-up']
    Error = False
    while Error == False:
        try:
            for Wort in Wörter_Liste:
                Wortlänge = len(Wort)
                platziert = False
                t_end = time.time() + 5
                while not platziert:
                    if time.time() < t_end:
                        failed = False
                        # Zufälliges Aussuchen der Richtung
                        Richtung = random.choice(Richtungen)

                        if Richtung == 'vertikal':
                            step_Zeile = 1
                            step_Spalte = 0
                            maximum_Zeile = ZeilenNr - Wortlänge
                            Zeilen_position = random.randrange(0, maximum_Zeile + 1)
                            Spalten_position = random.randrange(0, SpaltenNr + 1)

                        if Richtung == 'horizontal':
                            step_Zeile = 0
                            step_Spalte = 1
                            maximum_Spalte = SpaltenNr - Wortlänge

                            Zeilen_position = random.randrange(ZeilenNr + 1)
                            Spalten_position = random.randrange(maximum_Spalte + 1)

                        if Richtung == 'diagonal-lr-down':
```

```
                step_Zeile = 1
                step_Spalte = 1
                maximum_Zeile = ZeilenNr - Wortlänge
                maximum_Spalte = SpaltenNr - Wortlänge
                Zeilen_position = random.randrange(maximum_Zeile + 1)
                Spalten_position = random.randrange(maximum_Spalte + 1)

            if Richtung == 'diagonal-lr-up':
                step_Zeile = -1
                step_Spalte = 1
                minimum_Zeile = Wortlänge
                maximum_Spalte = SpaltenNr - Wortlänge
                Zeilen_position = random.randrange(minimum_Zeile, ZeilenNr +1)
                Spalten_position = random.randrange(maximum_Spalte + 1)

            for i in range(Wortlänge):
                # Das Wort wird Buchstabe für Buchstabe mit den Werten in der Tabelle
verglichen
                Buchstabe = Wort[i]
                neu_Zeile = Zeilen_position + i * step_Zeile
                neu_Spalte = Spalten_position + i * step_Spalte
                Buchstabe_neue_Position = grid[neu_Zeile][neu_Spalte]
                if Buchstabe_neue_Position != '0':
                    if Buchstabe_neue_Position == Buchstabe:
                        continue
                    else:
                        failed = True
                        break
            if failed:
                continue
            else:
                for i in range(Wortlänge):
                    # Das Wort je nach gewählter Richtung Buchstabe für Buchstabe eingetragen
                    Buchstabe = Wort[i]
                    neu_Zeile = Zeilen_position + i * step_Zeile
                    neu_Spalte = Spalten_position + i * step_Spalte
                    grid[neu_Zeile][neu_Spalte] = Buchstabe
                platziert = True
                break

    for x in range(ZeilenNr):
        for y in range(SpaltenNr):
```

```
                # Die übrig gebliebenen Leerstellen werden aufgefüllt
                if grid[x][y] == '0':
                    grid[x][y] = random.choice(string.ascii_uppercase)

        for x in range(ZeilenNr):
            # Ausgabe des Wortsucherätsels
            print('\t' * 2 + ''.join(grid[x]))
        Error = True
    except IndexError:
        grid = [['0' for _ in range(int(SpaltenNr))] for _ in range(int(ZeilenNr))]

# 3. Schwer: Horizontal, Vertikal, Diagonal l-r, Diagonal r-l und alle Wörter können rückwärts vorkommen
def Schwer(ZeilenNr, SpaltenNr, Wörter_Liste):
    Wörter_Liste = sorted(Wörter_Liste, key=len)
    Wörter_Liste = Wörter_Liste[::-1]
    grid = [['0' for _ in range(int(SpaltenNr))] for _ in range(int(ZeilenNr))]
    Richtungen = ['vertikal', 'horizontal', 'diagonal-lr-down', 'diagonal-lr-up', 'diagonal-rl-down', 'diagonal-rl-up']
    Richtungen.append('vertikal-rev')
    Richtungen.append('horizontal-rev')
    Error = False
    # Sollte es unmöglich sein, ein Wort unterzubringen, wird wieder von vorne gestartet, mit allen Worten
    while Error == False:
        try:
            for Wort in Wörter_Liste:
                Wortlänge = len(Wort)
                platziert = False
                t_end = time.time() + 5
                while not platziert:
                    if time.time() < t_end:
                        failed = False
                        Richtung = random.choice(Richtungen)
                        if Richtung == 'vertikal':
                            step_Zeile = 1
                            step_Spalte = 0
                            maximum_Zeile = ZeilenNr - Wortlänge
                            Zeilen_position = random.randrange(0, maximum_Zeile + 1)
                            Spalten_position = random.randrange(0, SpaltenNr + 1)

                        if Richtung == 'horizontal':
```

```
        step_Zeile = 0
        step_Spalte = 1
        maximum_Spalte = SpaltenNr - Wortlänge
        Zeilen_position = random.randrange(ZeilenNr + 1)
        Spalten_position = random.randrange(maximum_Spalte + 1)

    if Richtung == 'diagonal-lr-down':
        step_Zeile = 1
        step_Spalte = 1
        maximum_Zeile = ZeilenNr - Wortlänge
        maximum_Spalte = SpaltenNr - Wortlänge
        Zeilen_position = random.randrange(maximum_Zeile + 1)
        Spalten_position = random.randrange(maximum_Spalte + 1)

    if Richtung == 'diagonal-lr-up':
        step_Zeile = -1
        step_Spalte = 1
        minimum_Zeile = Wortlänge
        maximum_Spalte = SpaltenNr - Wortlänge
        Zeilen_position = random.randrange(minimum_Zeile, ZeilenNr +1)
        Spalten_position = random.randrange(maximum_Spalte + 1)

    if Richtung == 'diagonal-rl-down':
        step_Zeile = 1
        step_Spalte =  -1
        maximum_Zeile = ZeilenNr - Wortlänge
        minimum_Spalte = Wortlänge
        Zeilen_position = random.randrange(maximum_Zeile +1)
        Spalten_position = random.randrange(minimum_Spalte, SpaltenNr +1)

    if Richtung == 'diagonal-rl-up':
        step_Zeile = -1
        step_Spalte = -1
        minimum_Zeile = Wortlänge
        minimum_Spalte = Wortlänge
        Zeilen_position = random.randrange(minimum_Zeile, ZeilenNr +1)
        Spalten_position = random.randrange(minimum_Spalte, SpaltenNr +1)

    if Richtung == 'vertikal-rev':
        Wort = Wort[::-1]
        step_Zeile = 1
        step_Spalte = 0
```

```
            maximum_Zeile = ZeilenNr - Wortlänge
            Zeilen_position = random.randrange(0, maximum_Zeile + 1)
            Spalten_position = random.randrange(0, SpaltenNr + 1)

        if Richtung == 'horizontal-rev':
            Wort = Wort[::-1]
            step_Zeile = 0
            step_Spalte = 1
            maximum_Spalte = SpaltenNr - Wortlänge
            Zeilen_position = random.randrange(ZeilenNr + 1)
            Spalten_position = random.randrange(maximum_Spalte + 1)

        for i in range(Wortlänge):
            Buchstabe = Wort[i]
            neu_Zeile = Zeilen_position + i * step_Zeile
            neu_Spalte = Spalten_position + i * step_Spalte
            Buchstabe_neue_Position = grid[neu_Zeile][neu_Spalte]
            if Buchstabe_neue_Position != '0':
                if Buchstabe_neue_Position == Buchstabe:
                    continue
                else:
                    failed = True
                    break
        if failed:
            continue
        else:
            for i in range(Wortlänge):
                Buchstabe = Wort[i]
                neu_Zeile = Zeilen_position + i * step_Zeile
                neu_Spalte = Spalten_position + i * step_Spalte
                grid[neu_Zeile][neu_Spalte] = Buchstabe
            platziert = True
            break

for x in range(ZeilenNr):
    for y in range(SpaltenNr):
        if grid[x][y] == '0':
            grid[x][y] = random.choice(string.ascii_uppercase)

for x in range(ZeilenNr):
    print('\t' * 2 + ''.join(grid[x]))
Error = True
```

```
        except IndexError:
            grid = [['0' for _ in range(int(SpaltenNr))] for _ in range(int(ZeilenNr))]

# 4. Extrem: Horizontal, Vertikal, Diagonal l-r, Diagonal r-l und alle Wörter können rückwärts vorkommen
#           Das Wortsucherätsel wird mit Buchstaben, der unterzubringenden Wörter aufgefüllt
def Extrem(ZeilenNr, SpaltenNr, Wörter_Liste):
    Wörter_Liste = sorted(Wörter_Liste, key=len)
    Wörter_Liste = Wörter_Liste[::-1]
    grid = [['0' for _ in range(int(SpaltenNr))] for _ in range(int(ZeilenNr))]
    Richtungen = ['vertikal', 'horizontal', 'diagonal-lr-down', 'diagonal-lr-up', 'diagonal-rl-down', 'diagonal-rl-up']
    Richtungen.append('vertikal-rev')
    Richtungen.append('horitontal-rev')
    Buchstaben = []
    for Wort in Wörter_Liste:
        for char in Wort:
            Buchstaben.append(char)
    Buchstaben = list(dict.fromkeys(Buchstaben))
    Error = False
    while Error == False:
        try:
            for Wort in Wörter_Liste:
                Wortlänge = len(Wort)
                platziert = False
                t_end = time.time() + 5
                while not platziert:
                    if time.time() < t_end:
                        failed = False
                        Richtung = random.choice(Richtungen)
                        if Richtung == 'vertikal':
                            step_Zeile = 1
                            step_Spalte = 0
                            maximum_Zeile = ZeilenNr - Wortlänge
                            Zeilen_position = random.randrange(0, maximum_Zeile + 1)
                            Spalten_position = random.randrange(0, SpaltenNr + 1)

                        if Richtung == 'horizontal':
                            step_Zeile = 0
                            step_Spalte = 1
                            maximum_Spalte = SpaltenNr - Wortlänge
                            Zeilen_position = random.randrange(ZeilenNr + 1)
```

```
        Spalten_position = random.randrange(maximum_Spalte + 1)

    if Richtung == 'diagonal-lr-down':
        step_Zeile = 1
        step_Spalte = 1
        maximum_Zeile = ZeilenNr - Wortlänge
        maximum_Spalte = SpaltenNr - Wortlänge
        Zeilen_position = random.randrange(maximum_Zeile + 1)
        Spalten_position = random.randrange(maximum_Spalte + 1)

    if Richtung == 'diagonal-lr-up':
        step_Zeile = -1
        step_Spalte = 1
        minimum_Zeile = Wortlänge
        maximum_Spalte = SpaltenNr - Wortlänge
        Zeilen_position = random.randrange(minimum_Zeile, ZeilenNr + 1)
        Spalten_position = random.randrange(maximum_Spalte + 1)

    if Richtung == 'diagonal-rl-down':
        step_Zeile = 1
        step_Spalte = -1
        maximum_Zeile = ZeilenNr - Wortlänge
        minimum_Spalte = Wortlänge
        Zeilen_position = random.randrange(maximum_Zeile + 1)
        Spalten_position = random.randrange(minimum_Spalte, SpaltenNr + 1)

    if Richtung == 'diagonal-rl-up':
        step_Zeile = -1
        step_Spalte = -1
        minimum_Zeile = Wortlänge
        minimum_Spalte = Wortlänge
        Zeilen_position = random.randrange(minimum_Zeile, ZeilenNr + 1)
        Spalten_position = random.randrange(minimum_Spalte, SpaltenNr + 1)

    if Richtung == 'vertikal-rev':
        Wort = Wort[::-1]
        step_Zeile = 1
        step_Spalte = 0
        maximum_Zeile = ZeilenNr - Wortlänge
        Zeilen_position = random.randrange(0, maximum_Zeile + 1)
        Spalten_position = random.randrange(0, SpaltenNr + 1)
```

```
            if Richtung == 'horizontal-rev':
                Wort = Wort[::-1]
                step_Zeile = 0
                step_Spalte = 1
                maximum_Spalte = SpaltenNr - Wortlänge
                Zeilen_position = random.randrange(ZeilenNr + 1)
                Spalten_position = random.randrange(maximum_Spalte + 1)

            for i in range(Wortlänge):
                Buchstabe = Wort[i]
                neu_Zeile = Zeilen_position + i * step_Zeile
                neu_Spalte = Spalten_position + i * step_Spalte
                Buchstabe_neue_Position = grid[neu_Zeile][neu_Spalte]
                if Buchstabe_neue_Position != '0':
                    if Buchstabe_neue_Position == Buchstabe:
                        continue
                    else:
                        failed = True
                        break
            if failed:
                continue
            else:
                for i in range(Wortlänge):
                    Buchstabe = Wort[i]
                    neu_Zeile = Zeilen_position + i * step_Zeile
                    neu_Spalte = Spalten_position + i * step_Spalte
                    grid[neu_Zeile][neu_Spalte] = Buchstabe
                platziert = True
                break

    for x in range(ZeilenNr):
        # Hier wird anstatt string.ascii.uppercase Buchstaben als Basis verwendet
        for y in range(SpaltenNr):
            if grid[x][y] == '0':
                grid[x][y] = random.choice(Buchstaben)

    for x in range(ZeilenNr):
        print('\t' * 2 + ''.join(grid[x]))
    Error = True
except IndexError:
    grid = [['0' for _ in range(int(SpaltenNr))] for _ in range(int(ZeilenNr))]
```

```
# Ausführug der Funktionen:
Schwierigkeit = sys.argv[2]
if Schwierigkeit == 'Einfach':
    Einfach(ZeilenNr,SpaltenNr,Wörter_Liste)
if Schwierigkeit == 'Mittel':
    Mittel(ZeilenNr, SpaltenNr, Wörter_Liste)
if Schwierigkeit == 'Schwer':
    Schwer(ZeilenNr, SpaltenNr, Wörter_Liste)
if Schwierigkeit == 'Extrem':
    Extrem(ZeilenNr, SpaltenNr, Wörter_Liste)
print('')
print('Wörter versteckt:')
print(', '.join(Wörter_Liste))
```