Inhaltsverzeichnis

1 Lösungsidee

1.1 Genereller Ansatz

Zuerst werden einige Funktionen eingeführt, die im Folgenden die Ausdrucksweise vereinfachen sollen. Die Funktion z ordnet jedem Paket P sein Zielhaus zu. Die Funktion d_M bezeichnet die Manhattan-Distanz [2] zwischen zwei Häusern, es ist also $d_M((x,y),(x',y')) = |x-x'|+|y-y'|$. Der Wert eines Hauses an Stelle (x,y) in einem $n \times n$-Quadrat ist wie folgt definiert:

$$\omega : (x,y) \mapsto \begin{cases} x \cdot n + y, & \text{für } x \equiv 0 \mod 2 \\ x \cdot n + n - y - 1, & \text{für } x \not\equiv 0 \mod 2 \end{cases}$$

Man kann sich leicht davon überzeugen, dass diese Funktion bijektiv auf die Zielmenge $\{0, 1, \ldots, n^2-1\}$ ist. So ergibt sich die Wert-Distanz d zweier Häuser H und G als $d(H,G) = |\omega(H) - \omega(G)|$. Der Wert eines Paketes ist der Wert seiner aktuellen Position. Somit ergibt sich eine Bewertung ϕ eines Feldes, das aus einer $n \times n$ Matrix M mit nullbasierten Indices, die mit Paketen gefüllt ist, besteht, als ein Paar nicht-negativer, ganzer Zahlen. Dabei ist die erste Zahl das Maximum der Wert-Distanzen aller Pakete zur ihren Zielhäusern und die zweite Zahl die Anzahl, wie oft diese Maximum angenommen wird. Ein Feld, in dem alle Pakete richtig verteilt sind, hat offensichtlich die Bewertung $(0, n^2)$.

In einem Feld kann sich ein Paket P in einem Umsortierungsschritt maximal um eine Position in entweder $x-$ oder $y-$Richtung bewegen. Daher benötigt es mindestens $d_M(P, z(P))$ viele Umsortierungsschritte um zu seinem Ziel zu gelangen. Ein ganzes Feld benötigt folglich mindestens soviele Umsortierungsschritte wie die größte Manhattan-Distanz zwischen einem seiner Pakete und dessen Ziel. Jedoch ist diese Distanz kein Minimum für die Anzahl der Schritte sondern nur eine untere Schranke, so lässt sich bei folgender Ausgangssituation keine Lösung in 2 Schritten finden, da jedes der Pakete auf seinem kürzesten Weg durch die Mitte muss, was nicht alle gleichzeitig können.

(0, 0)	(2, 1)	(0, 2)
(1, 2)	(1, 1)	(1, 0)
(2, 0)	(0, 1)	(2, 2)

Abbildung 1: Ein Feld, das mehr als $\max(d_M(P, z(P)))$ Schritte benötigt.

Wenn man sich die Werte aller Häuser in einem $n \times n$ ansieht ergibt sich folgende Situation (am Beispiel eines 6×6-Quadrates):

0	1	2	3	4	5
11	10	9	8	7	6
12	13	14	15	16	17
23	22	21	20	19	18
24	25	26	27	28	29
35	34	33	32	31	30

Abbildung 2: Die Werte von eines 6×6-Quadrates.

Das ist auch der Zielzustand, den man mit den Werten der Pakete erreichen will. Diese schlangenartige Sortierung ließe sich mit dem Shearsort-Algorithmus[3] leicht erreichen, jedoch liefert dieser keine besonders guten Ergebnisse (in ersten Tests erreichte konnte er Amacity in 56 Zügen sortieren). Die anderen in der Quelle erwähnten Algorithmen erschienen mir jedoch sehr schwer und können vorallem bei $2^n \times 2^n$ Matrizen ihre Vorteile entfalten. Zumal sind sie darauf optimiert die Matrix möglichst schnell zu sortieren, was nicht unbedingt gleichbedeutend damit ist, sie in möglichst wenig Schritten zu sortieren, wie es in der Aufgabe gefordert wird. So kann man sich, da der Plan vor der Umverteilung berechnet wird, erlauben, das Feld zwischen oder während der Berechnung der Umverteilungsschritte ausgiebig zu evaluieren, was bei einer möglichst schnellen Echtzeitsortierung verheerend wäre. Auch wird bei diesen Algorithmen davon ausgegangen, dass jede Position nur Informationen über sein eigenes Paket und über die seiner unmittelbaren Nachbarn hat, was wieder nicht der Fall ist.

Darum habe ich mich dazu entschieden, statt einen anderen Algorithmus zu benutzen, den Shearsort-Algorithmus zu verbessern. Beim Shearsort-Algorithmus kommt es zu sehr starren Abläufen der verschiedenen Teilalgorithmen, folglich kann es passieren, dass es womöglich besser wäre einen anderen Teilalgorithmus auszuführen, als den, der gerade an der Reihe ist, vorallem ist es interessant, möglichst viele Umverteilungen gleichzeitig durchzuführen und den maximalen Abstand der Pakete zu ihren Zielen zu erniedrigen, da dies die Mindestanzahl an noch zu tuenden Schritten ist. Auch werden bei Shearsort die Reihen einzeln sortiert, wodurch an den Enden vielleicht Schritte verschwendet werden, weil z.B. der Anfang der Reihe beim Odd-Durchgang nicht verändert wird. Dabei lässt einfach sich auf die gesamte Schlange Odd-Even-Sort anwenden, was nur höchstens die beiden Positionen an den Enden der gesamten Schlange unbenutzt lässt anstatt zwei Positionen pro Reihe.

Daher habe ich mich für eine etwas freiere Abfolge der Teilalgorithmen entschieden. Hierbei werden als Teilalgorithmen jeweils Odd- und Even-Durchgangs des Odd-Even-Sorts über die gesamte Schlange sowie eine einzelne Anwendung der horizontalen Sortierung gezählt. Die Abfolge dieser Teilalgorithmen wird über das Ergebnis bewertet, dass sie erzielen. Dazu sortiert jeder der Algorithmen eine Kopie des aktuellen Feldes und dann der Algorithmus, dessen Ergebnis am besten bezüglich der Bewertung ϕ ist, wird dann wirklich auf das Feld

angewandt. Dabei erfolgt die Sortierung der Bewertungen lexikographisch, da es vorallem wichtig ist, die maximalen Distanzen zu minimieren, da diese die minimale Anzahl an noch zu gehenden Schritten bei reiner Odd-Even-Sortierung angeben.

Durch diese Ungleichbehandlung von Reihen und Spalten kann es die Lösung stark beeinflussen, ob das Quadrat so sortiert wird, wie es gegeben ist, oder ob die an der Nordwest-Südost-Diagonalen gespiegelte Version sortiert wird. Bei der gespiegelten Version sind die Reihen des Originals die Spalten und umgekehrt. Dadurch ändert sich die Anwendung der Algorithmen grundlegend und damit auch deren Ergebnisse. Allerdings sind beide Sortierungen auf dem Original anwendbar, man muss einfach nur jegliche Sortierung der gespiegelten Version wieder zurückspiegeln. Da man nicht sagen kann, welche Version die besseren Ergebnisse liefert, werden beide sortiert und die kürzere Version zurückgegeben.

1.2 Algorithmen

1.2.1 Odd-Even-Sort

Der gewöhnliche Odd-Even-Sortieralgorithmus besteht aus abwechselnden Odd- und Even-Durchgängen. Diese funktionieren folgendermaßen:

procedure ODD(Feld)
  for all ungerade Positionen im Feld **do**
    if $\omega(Position.Paket.Ziel) > \omega(Position.next.Paket.Ziel)$ **then**
      Tausche Pakete von Position und Position.next
    end if
  end for
  return Feld
end procedure

Der Even-Durchgang macht das gleiche mit den geraden Positionen, d.h. mit den Positionen mit geradem Index. Nach höchstens n kombinierten Odd-Even-Durchgängen ist eine Liste der Länge n sortiert.[1] Diese Odd- und Even-Durchgänge werden in meinem Algorithmus zum Einen direkt aufs Feld angewandt, zum Anderen in der Spaltensortierung benutzt.

1.2.2 Spaltensortierung

Die Spaltensortierung ist eine Anwendung von Odd-Even-Sort auf die Spalten der Matrix. Dabei wird auf jede Spalte nur einer der beiden Durchläufe angewandt, damit die Sortierung vergleichbar mit den einzelnen Odd- und Even-Durchläufen bleibt. Deswegen wird auch wieder der jeweils bessere der beiden Durchläufe angewandt:

procedure SPALTENSORTIERUNG(Feld)
  for all Spalten in Feld **do**
    odd = ODD(Spalte)
    even = EVEN(Spalte)
    if $\phi(odd) < \phi(even)$ **then**
      Spalte = odd
    else

```
            Spalte = even
        end if
    end for
    return Feld
end procedure
```

1.2.3 Quadratsortierung

Bei einigen Beispielen ist mir aufgefallen, dass, vorallem zu Ende der Sortierung, oft ein Rundtausch in einem 2×2-Quadrat, d.h. von jedem Haus wird das Paket an das nächste Haus im/entgegen dem Uhrzeigersinn weitergegeben, die beste Lösung wäre und manchmal die Pakete sofort ans Ziel bringt. Jedoch lässt sich so ein Rundtausch durch die Paartausche, also Tausche bei denen zwei Häuser ihre Pakete direkt austauschen, wie sie bei den bisherigen Sortieralgorithmen ausschließlich vorkommen, nur sehr schlecht emulieren. Daher wird die Quadratsortierung eingeführt. Diese ist anstatt einer zusätzlichen Sortierung eine Aufbesserung der letzten Sortierung. Dazu werden alle 2×2 Subquadrate durchgegangen. Wenn alle Tausche der 4 Häuser, die das Quadrat bilden, innerhalb dieses Quadrates waren, kann man den Log dieser 4 Häuser manipulieren, ohne dass es einen Einfluss auf die Häuser außerhalb diese Quadrats hat. Wenn die Pakete nach der letzten Optimierung nicht optimal platziert waren, kann man sie folglich umsortieren und den Log entsprechend manipulieren ohne Inkonsistenzen zu erzeugen. Diese Methode ist sehr laufzeitintensiv, wie ich später noch aufzeigen werde. Daher hat die Nutzer die Möglichkeit sie für große Felder abzuschalten.

2 Implementierung

Das Programm ist in Python3 geschrieben und befindet sich in der Datei *aufgabe2.py*. Für die Ausgabe ist das Python3-Package PyQt4 von Nöten. Zur Installation lässt sich unter Arch-Linux z.B. *$sudo pacman -S python-pyqt4* aufrufen. Zu Beginn wird der Ausführer nach der Eingabedatei gefragt. Diese muss im Format der vorgegebenen Beispieldateien vorliegen. Danach wird gefragt, ob die Quadratsortierung ausgeführt werden soll. Nach Angabe dieser beiden Parameter beginnt ein Timer und die `main1`-Methode wird mit den Parametern aufgerufen. Diese ruft `shearsort` und für das Originalfeld und das gespiegelte Feld die `main`-Methode auf und filtert aus den beiden Ergebnissen das kürzere heraus, was sie dann auch zurückgibt. Die Methode ist nur vom Skriptteil der Ausführung getrennt um Tests zu vereinfachen.

2.1 Shearsort

Zusätzlich zu meinem eigenen Algorithmus habe ich, wie ich später erklären werde, in `shearsort` noch den Shearsort-Algorithmus implementiert. Die Implementierung ist sehr einfach direkt [3] nachempfunden. Dazu werden $\lceil log_2(n) \rceil$-mal jeweils die Zeilen und Spalten Odd-Even-sortiert und danach nocheinmal die Zeilen sortiert. Für die Odd-Even-Sortierung wird jeweils zweimal `oddeven` aufgerufen, je einmal mit dem `start`-Parameter 0 und 1. Die Logs werden pro Durchgang in `newlog` für den Even- und `newlog1` für den Odd-Durchgang gespeichert.

Dann werden nur die Logs nur übernommen, wenn sie wirklich eine Veränderung durchgeführt haben, also sie nicht leer sind. Dabei wird zuerst `newlog` übernommen, da für jede Zeile bzw. Spalte stets zuerst der Even-Durchgang ausgeführt wird. Wenn beide Logs leer sind, sind die Zeilen bzw. Spalten sortiert und der nächste Sortierschritt wird vorgenommen.

2.2 `main`-Methode

In der `main`-Methode wird die Methode `lesen` aufgerufen. Diese ließt das Feld, gegebenenfalls gespiegelt, aus und speichert im Dictionary `feld` für jede Position als Wert das Ziel des Pakets, das dort zu Beginn ist. Nun wird das Dictionary `log` initialisiert, dass für jede Position eine Liste enthält, in der später die Tauschanweisungen eingetragen werden. Dann wird beim ersten Mal, also beim richtig orientierten Feld, das Feld einmal auf die Konsole ausgegeben. Dies funktioniert bei großen Feldern naturgemäß nicht so gut. Daraufhin wird solange das Feld unsortiert ist, also solange die Bewertung größer als $(0, n^2)$ ist, je ein Durchlauf von Höhensortierung, Odd- und Even-Durchgang auf eine Kopie von `feld` angewandt. Dazu werden die Methoden `hor` und `oddeven` aufgerufen, wobei `oddeven` je einmal 0 und 1 als `start` übergeben werden. Das beste Ergebnis ersetzt dann das alte `feld` und die entsprechenden Tauschanweisungen in den Log übernommen. Danach wird die Methode `square` aufgerufen, die die Quadratsortierung auf eine Kopie von `feld` ausübt. Falls das Ergebnis besser ist als das aktuelle Feld, wird das Feld ersetzt und der Log entsprechend geändert. Dabei wird ständig ein Fortschrittsbalken aktualisiert, der die aktuelle Bewertung im Vergleich zur ersten Bewertung anzeigt. Da die Bewertung zu Beginn stärker abnimmt als zu Ende ähnelt dieser Balken aber eher dem Windows Kopierdialog [4].

2.3 Odd-Even-Sort

Die Odd- und Even-Durchgänge werden in der Methode `oddeven` durchgeführt. Dabei gibt der Parameter `start` an, ob man bei der nullten oder der ersten Position beginnt und entscheidet somit, ob es ein Even- oder ein Odd-Durchgang ist. Das übergebene Dictionary `tosort` wird nach `ret` kopiert. In der Liste `keys` werden die nach ihrem Wert sortierten Schlüssel von `tosort` gespeichert. Deswegen kann man dann einfach diese Liste per for-Schleife durchgehen, je nach `start` beginnend bei 0 oder 1, und für jede Position der Wert des Pakets mit dem des Pakets des Nachfolgers verglichen. Falls der Wert beim Nachfolger kleiner ist, tauschen die beiden Positionen die Pakete, also ihre Werte in `ret`, aus und es wird in `ausg` ein entsprechender Vermerk für den Log eingefügt. Zum Schluss wird das Paar aus `ret` und `ausg` zurückgegeben.

2.4 Spaltensortierung

Die Spaltensortierung findet in der Methode `hor` statt. Dabei werden alle Spalten durchgegangen und für jede je einmal `oddeven` mit dem `start`-Parameter 0 und 1 aufgerufen, was, wie schon erwähnt, einem Even- bzw. einem Odd-Durchgang entspricht. Das jeweils bessere Ergebnis wird `endfeld` hinzugefügt und der entsprechende Log an `ausg` angefügt. Zum Schluss wird dann das Paar aus `endfeld` und `ausg` zurückgegeben.

2.5 Quadratsortierung

Für die Quadratsortierung wird `square` sowohl das aktuelle Feld als auch der Log übergeben. Aus dem Log wird für jede Position der letzte Eintrag extrahiert, dieser muss keine Veränderung sein, sondern kann auch ein "_" sein. Dann werden alle Positionen durchgegangen, die nicht am rechten oder unteren Rand liegen. Wenn man dann zu jeder Position das 2×2-Quadrat nimmt, dass diese Position als linke, obere Ecke hat, findet man somit alle 2×2-Quadrate. In diesen wird geprüft, ob für jede der 4 Positionen die letzte Veränderung in diesem Quadrat liegt. Das ist der Fall, wenn die nordwestliche Ecke ihr letztes Paket in den Süden oder Osten abgegeben hat oder zuletzt ihr Paket behalten hatte, usw. für alle Ecken. Dann ist es der Quadratsortierung möglich, dieses Quadrat zu bearbeiten. Falls jedoch das Quadrat schon optimal sortiert ist, d.h. die Position mit dem kleinsten Wert hat bereits das Paket mit dem kleinsten Wert des Zielhauses etc., ist es sinnlos daran zu arbeiten, und es wird weitergegangen. Andernfalls werden daraufhin für alle Positionen die letzten Änderungen rückgängig gemacht, da nur so alle erlaubten Züge in diesem Quadrat gefunden werden können. Danach werden in `allsq` alle möglichen Züge für diese Position erzeugt. Das schließt die Veränderungen der Pakete und die Aufzählung der Positionsveränderungen mit ein. Um diese Züge zu erzeugen wurde im Vorhinein überlegt, dass es für das 2×2-Quadrat im äußersten Nordwesten genau 9 gültige Züge gibt:

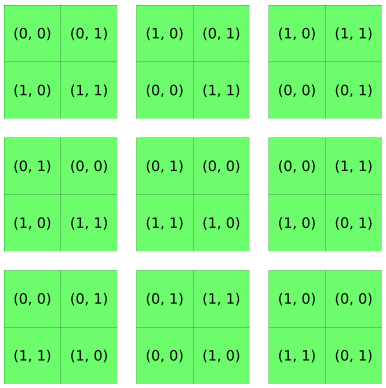

Abbildung 3: Alle legalen Züge auf einem 2×2-Quadrat.

Dabei wird von dem sortierten Feld links oben ausgegangen. Jedes andere 2×2-Quadrat lässt sich durch Verschiebung auf diese Quadrat abbilden, daher lassen sich auch die Züge leicht abbilden. Daher werden in `allsq` die, in `twoxtwo` im Quellcode gespeicherten Züge für das nordwestliche Quadrat auf die aktuelle Position angepasst und so alle legalen Züge für das aktuelle Quadrat erzeugt. Danach wird die Bewertungsmethode `bewertefeld` auf alle Züge angewandt und der Zug mit der besten Bewertung übernommen. [1]

[1] Am 1. April habe ich einen Programmierfehler in dieser Methode gefunden. Seit der Ausbesserung bekomme ich für Amacity keine Sortierung mehr zustande die so gut ist wie mein bestes Ergebnis davor, das ich deshalb als *sehrkurzerlog.txt* beilege.

2.6 Ausgabe

(a) Die Animation zu Beginn (b) Die Animation zu Ende

Schlussendlich wird das Ergebnis im vorgegebenen Format abgespeichert und gefragt, ob der Ausführer das Ergebnis animiert haben will. Die Animation wird in *animation.py* durchgeführt. Diese kann auch eigenständig aufgerufen werden. Dabei werden aus dem Log alle Bewegungen ausgelesen und vom Ausgangsfeld aus nachgespielt. Wegen begrenzten Bildschirmgrößen sind die Animationen vorallem bei kleineren Feldern sinnvoll. Während der Animation sind die Felder bezüglich der Manhatten-Distanz von ihrem Paket zu dessem Ziel von grün nach rot im Vergleich zur aktuell größten solchen Distanz farbmarkiert. Da jeweils die aktuell größte Distanz genommen wird, gibt es stets mindestens ein rotes Feld.

3 Laufzeit und Korrektheit

Der Sortieralgorithmus terminiert auf jeden Fall. Dazu reicht es zu zeigen, dass bei jedem möglichen Feld mindestens einer der Teilalgorithmen die Bewertung erniedrigt. Da die Bewertung ein Minimum hat, wenn das Feld sortiert ist, muss dieses dann irgendwann erreicht werden. Weder Odd- noch Even-Durchgang erhöhen die Bewertung einer Lösung, dazu müsste sich nämlich ein Paket mit maximaler Distanz entgegen seiner gewünschten Bewegungsrichtung bewegen. Das ist aber nur der Fall wenn das Nachbarpaket in der ungewünschten Richtung in die gleiche Richtung will und einen Zielwert mit höherer Priorität hat, dann wäre aber auch die Distanz dieses Pakets größer, was nicht sein kann, da ja das ursprüngliche Paket maximale Distanz hat. Auch können keine zwei Pakete benachbart sein und in die gleiche Richtung wollen, wenn sich ihre Distanzen von den jeweiligen Zielen nur um 1 unterscheiden und das Paket mit der größeren Distanz weiter von seinem Ziel entfernt ist, da sie dann auf das gleiche Ziel wollen würden. Daher erniedrigt jeder Tausch, in den ein Paket mit maximaler Distanz involviert ist, die Bewertung. Es gibt auch stets so einen Tausch, da es entweder ein Paket mit maximaler Distanz gibt, dass in der Richtung, in die es will, ein Paket mit kleinerer Distanz und somit auch einem Zielwert mit niedrigerer Priorität hat, oder eine zusammenhängende Reihe von Paketen mit maximaler Distanz, sodass die Pakete an ihren Enden in unterschiedliche Richtungen wollen. Dann muss es zwischendurch einen

Wechsel der gewünschten Richtung geben, was dazu führt, dass zwei Pakete mit maximaler Distanz einen für beide vorteilhaften Tausch durchführen.

Folglich nimmt die Bewertung stets ab. Leider lässt sich über die Bewertung wenig über die maximale Schrittzahl aussagen. Da es genau n^2 Pakete gibt und die Bewertung stets um mindestens 1 in der zweiten Stelle abnimmt, lässt sich sehr leicht zeigen, dass der Sortieralgorithmus maximal n^4 Schritte benötigt. Der Algorithmus ohne Quadratsortierung ist wohl nicht schlechter als der normale Shearsort Algorithmus, allerdings kann ich das nicht beweisen. So kann die lokale Optimierung, die in jedem Schritt das gerade beste Feld nimmt, unter Umständen kontraproduktiv sein. Allerdings zeigt sich bei zufälligen Feldern, dass die Sortierung sehr gut im Vergleich zum Shearsort-Algorithmus abschneidet und auch die Quadratsortierung einen durchschnittlichen Vorteil von 10% gegenüber meinem Algorithmus ohne Quadratsortierung bringt. Sowohl die Sortierung mit als auch ohne Quadratsortierung scheinen folglich ein lineares Average-Case-Ergebnis an Schritten zu haben, was besser ist als der $\mathcal{O}(n \log(n))$ Verhalten von Shearsort ist.

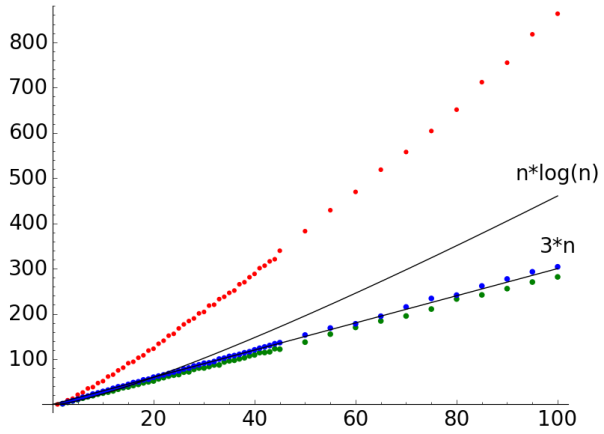

Abbildung 5: Die Schrittzahlen im Vergleich: Shearsort (rot), Sortieralgorithmus ohne (blau) und mit (grün) Quadratsortierung

Es gibt jedoch auch Felder, in denen der Algorithmus mit Quadratsortierung sowohl schlechter als der Algorithmus ohne Quadratsortierung als auch schlechter als der Shearsort-Algorithmus ist. Das Programm *worst.py* erzeugt solche Felder, die dem sortierten Feld gespiegelt an der Südwest-Nordost Diagonalen entsprechen, mit beliebiger Größe. Allerdings ist bei diesen Feldern der Algorithmus ohne Quadratsortierung auch besser als Shearsort. Ich führe daher als ein Teilalgorithmus Shearsort durch um so ein garantiertes Worst-Case-Verhalten von $\mathcal{O}(n \log n)$ zu haben. Auch hat der Nutzer die Wahl ob er die Quadratsortierung benutzen will oder nicht.

Dies waren Betrachtungen zur Schrittzahl der Ergebnisse. Die Laufzeit der Planerzeugung ist abhängig von der Schrittzahl der Pläne. Für jeden Schritt ist jedoch eine Laufzeit von $\mathcal{O}(n^2)$ erforderlich, da bei jedem der Teilalgorithmen alle Felder mehrfach abgegangen werden. Die Anzahl, wie oft ein Feld abgegangen wird hat eine konstante Obergrenze, da jedes Feld in den Algorithmen maximal 4 mal besucht wird (bei der Quadratsortierung) und zusätzlich

nur ca noch 4 mal bei den Feldbewertungen besucht wird. Daher vermute ich eine Laufzeit der Planerzeugung zwischen $\mathcal{O}(n^3)$ und $\mathcal{O}(n^4)$. Bei Shearsort ist die Laufzeit $\mathcal{O}(n^3 \log(n))$ da bei jedem Schritt bis zu n^2 Tausche vollführt werden, die alle nacheinander berechnet werden müssen. Jedoch ist Shearsort bei der Laufzeit den anderen Algorithmen stark überlegen, da bei diesen, durch die vielen Algorithmen, die jeweils durchgeführt werden, die Konstanten der höchsten Potenzen sehr groß werden. Deswegen ist auch der Sortieralgorithmus mit Quadratsortierung wesentlich langsamer als der ohne, obwohl sie wohl nominal die gleiche Laufzeit haben. Ein Test der durchschnittlichen Ausführzeiten, gemessen auf einem i3-5010U, ergibt:

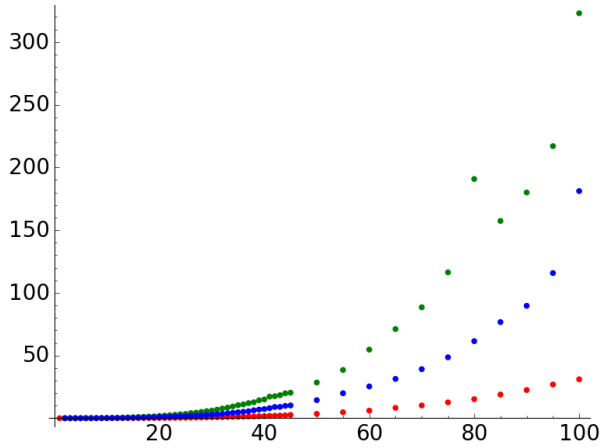

Abbildung 6: Die durchschnittlichen Ausführzeiten, Farbcodierung wie oben

Dabei ist bei einer Feldgröße von 25 die Ausführzeit noch sehr gering. Bei allen größeren Feldern gibt es mehr als 640 Häuser und 640 Häuser sollten eigentlich genug für jeden sein. Zusätzlich wurden auf 5000 zufälligen Feldern der Größe 10×10 Tests der Sortierung mit oder ohne Quadratsortierung durchgeführt. Dabei zeigt sich, dass die Sortierung mit Quadratsortierung im Durchschnitt etwa eine 1.95-mal solange Ausführzeit hat wie die ohne Quadratsortierung auf dem gleichen Feld. Dabei schwankten die Werte zwischen 1.01 und 3.79. Allerdings erzielte die Sortierung mit Quadratsortierung eine durchschnittliche Schrittzahl von 24.96, was 3.78 Schritte unter dem Durchschnitt von 28.74 ohne Quadratsortierung liegt. Das entspricht einer Verbesserung von 13%. Jedoch ist die Sortierung mit Quadratsortierung unter Umständen bis zu 3 Schritte besser, dafür manchmal auch 13 Schritte schlechter. Auch zeigt es sich, dass es gut war, beide Orientierungen des Feldes zu sortieren, da beide Orientierungen etwa gleich oft das bessere Ergebnis liefern. Die untere Schranke der Schrittanzahl lag im Durchschnitt bei 15.39 Schritten, weshalb ich mit meinem Ergebnis von 24.96 Schritten sehr zufrieden bin. Am nächsten kam ein Sortierung bis auf 3 Schritte an die untere Schran

4 Beispiele

Da ein Beispiel der Größe n eine Ausgabe mit n^2 Zeilen erzeugt, schreibe ich nur die Ausgabe von Amacity mit Quadratsortierung in die Dokumentation und lege die anderen Beispiele bei. Sie sind im Ordner “Aufgabe2/Beispiele/” und haben die Endung *.out*.

```
0 0 _________S________________
0 1 O_O_O_____O____SO_____SO__
0 2 WOWOW____SW_OS__WS_____W__
0 3 _WOW______O_WSO__SO_SO_O__
0 4 _OWO_____SW__SW___W_SWSW__
0 5 _W_WO____SO______SO_SO_OS_
0 6 ____WO____W_______W_SWSWS_
0 7 _____W___SO__SOS__O_SO____
0 8 __________W__SWS_SW_SW____
0 9 _________S___S_S____S_____
1 0 O___S___ONO_____S_________
1 1 W____O_OW_WS__SNS__S_SN___
1 2 ____SWOW_N_S_N___N________
1 3 _____OWO_____NS_SN_SN_____
1 4 _____WOWON_S_N__S___NSN___
1 5 _____OWOWN____S_SN_SN___N_
1 6 O____WOWO_OS________NSNSN_
1 7 W___SOWOWNWS_NSNS___NS____
1 8 O____W_W__O__NSNSN_SNS____
1 9 W___S____NWS_NSN___SN_____
2 0 O___N____S__O__SNS________
2 1 WO_O_____SONWSNSNSONSN__S_
2 2 OWOWNO___SWNO____SW_S_____
2 3 WOWO_W____O_W_NSNS_N______
2 4 _WOW_O___SWNOS_SN___SN____
2 5 __WO_WO_OSO_WSNSNSONS_____
2 6 ___W__WOW_WN_S___SW_SNSN__
2 7 _O__N_OWOS_N__NSNS___NS___
2 8 _WO__OW_WS__O_NSNS_NSN____
2 9 __W_NW___S_NWSN__S_N______
3 0 O___S_O__N____SNSN________
3 1 W____OW__N_S_NSNSN_SN__SN_
3 2 ____SWO_ON______SN_SNS____
3 3 _____OWOW__SO_SNSN________
3 4 ____SWOWONO_WNSN___SN_____
3 5 ____SOWOWNWS_NSNSN_SN_____
3 6 O____WOW__OS_NS_SN_SNSNS__
3 7 W___SOWO_NW_O_SNSN____N___
3 8 _____W_WONOSW_SNSN_SNS____
3 9 ________WNWS_NS_SN________
4 0 _O_ONO___S__OSNSN_________
4 1 OWOW_WO___ONWSNSNS_NS_SN__
4 2 WOWONOW__SW____SNS_NSN____
4 3 _WOW_W__OSON_SNSN_________
4 4 _OWON__OWSW_O_NS_S_N______
```

```
4 5 _WOWN_OWOS_NWSNSNS_N______
4 6 __WO_OWOWS_NOSNSNS_NSNSNS_
4 7 O_OWNW_WO___W_N_NS________
4 8 WOW_____W__N_SNSNS_NSN____
4 9 _W_______S_N_SNSN_________
5 0 ______O__NOS_NSN__O__S____
5 1 __O__OWO__WSONSNSNWSNSN___
5 2 _OWO_WOW_NO_W_SNSN_SN_____
5 3 _WOW_OW__NW_ONSNS_________
5 4 _OWO_W__ONOSW_SNSN________
5 5 OWOW___OWNWSONSNSN_S______
5 6 WOWO__OW_N__WNSNSN_SNSN_NS
5 7 OWOW_OW____SO___SN________
5 8 W_WOSW____O_WNSNSN_SNS____
5 9 ___W_____NWS_NSNS_________
6 0 O_O__O___S_N_SN_____SN____
6 1 WOWO_WO_OSON__NSNS_NSN____
6 2 OWOW_OWOW_W__SNSNS_N______
6 3 WOWO_WOWOS____N_NS________
6 4 OWOW_OWOWS_N_SNSNS________
6 5 W_WO_WOWO_ON__NSNS_N______
6 6 __OW_OWOWSW_OSNSNS_NSN___N
6 7 _OW__WOWO_ONW__SN_________
6 8 OW__NOWOWSW___NSNS_NSNS___
6 9 W____W_W___N_SNSNS________
7 0 __O_S____NOS_NS___O_N____S
7 1 _OWO_O___NW_O_SNSNWSNS____
7 2 OWOWSW____OSWNSNSN________
7 3 WOWO_____NW___S__N________
7 4 _WOWS____N_S_NSNSN_S______
7 5 __WOSO______O__NSN________
7 6 __OW_W___NOSWNSNSN_SNS____
7 7 _OWOS_____W_O_SN__________
7 8 OWOW_____NOSW_SN_N__NSNS__
7 9 W_W_S_____W__NSNSN_S______
8 0 _O__NO___S_N_SNS______S_SN
8 1 _W___WO_O____SN_NS_NSNS___
8 2 ____NOWOWS_NO_NSN_________
8 3 O____WOW__O_WSN___________
8 4 WO_ONOWO_SWN_SNSNS_NS_____
8 5 OWOWNWOW________N___S_____
8 6 WOWO_OW__S_NOSNSNS_NSN____
8 7 OWOWNW______W_N__S__S_S___
8 8 WOW__O___S_N_SN______NSNS_
8 9 _W__NW___S____N_N__NS_____
```

```
9 0 O_O___O__NO__N_NO_O__ONON_
9 1 WOWO_OWO__W_ON__WNW_NWNW__
9 2 OWOWOWOW_NO_W__NO_________
9 3 W_WOWOW___W_ON__W_________
9 4 O_OWOWO__NO_WN_NONO_N_____
9 5 WOWOWOWO__W_____W_W_N_____
9 6 OWOWOWOW_N___NONONO_NO____
9 7 W_WOW_WO______W_WNW_NWN___
9 8 ___WO__W_N___NO_______N_N_
9 9 ____W____N____W_____N_____
```

5 Quellen

Literatur

[1] https://en.wikipedia.org/wiki/Odd%E2%80%93even_sort

[2] https://en.wikipedia.org/wiki/Taxicab_geometry

[3] http://www.iti.fh-flensburg.de/lang/algorithmen/sortieren/twodim/shear/shearsorten.htm

[4] https://xkcd.com/612/

6 Quellcode

Die Abgabe enthält die Programmdateien *aufgabe2.py*, *animation.py*, *gui.py*, *fortschritt.py*, *worst.py* und *randfeld.py*. Dabei ist *aufgabe2.py* das Hauptprogramm, das *animation.py* und *fortschritt.py* benötigt. In *animation.py* wird die Animation aufgerufen und benötigt *gui.py*. Die anderen beiden Dateien erzeugen zum Einen sehr schwere und zum Anderen zufällige Felder beliebiger Größe. interessant genug für die Dokumentation sind meines Erachtens nur die Dateien *aufgabe2.py*, *animation.py* und *gui.py* interessant:

aufgabe2.py

```
1 #!/bin/env python3
2 from datetime import datetime, timedelta
3 import fortschritt
4
5 DirToStr = {(0, 1): 'O', (1, 0): 'S', (0, -1): 'W', (-1, 0): 'N', (0, 0):
  '_'}
6 StrToDir = {'N': (-1, 0), 'W': (0, -1), 'S': (1, 0), 'O': (0, 1), '_': (0,
  0)}
7
8 twoxtwo = [{(0, 1): (0, 1), (1, 0): (1, 1), (0, 0): (0, 0), (1, 1): (1, 0)},
```

```
9             {(0, 1): (0, 0), (1, 0): (1, 0), (0, 0): (0, 1), (1, 1): (1, 1)},
10            {(0, 1): (1, 1), (1, 0): (0, 0), (0, 0): (1, 0), (1, 1): (0, 1)},
11            {(0, 1): (1, 1), (1, 0): (0, 0), (0, 0): (0, 1), (1, 1): (1, 0)},
12            {(0, 1): (0, 1), (1, 0): (1, 0), (0, 0): (0, 0), (1, 1): (1, 1)},
13            {(0, 1): (1, 1), (1, 0): (1, 0), (0, 0): (0, 0), (1, 1): (0, 1)},
14            {(0, 1): (0, 0), (1, 0): (1, 1), (0, 0): (1, 0), (1, 1): (0, 1)},
15            {(0, 1): (0, 1), (1, 0): (0, 0), (0, 0): (1, 0), (1, 1): (1, 1)},
16            {(0, 1): (0, 0), (1, 0): (1, 1), (0, 0): (0, 1), (1, 1): (1, 0)}]
17
18 diff = lambda y, x: (x[0] - y[0], x[1] - y[1])
19
20
21 def lese(filename, turned=False):
22     ''' Liest das Feld aus einer Datei, gegebenenfalls auch die gespiegelte
   Variante '''
23     with open(filename) as f:
24         lines = f.readlines()
25     global size
26     size = int(lines.pop(0).strip())
27     feld = {}
28     for i in lines:
29         sp = list(map(int, i.split()))
30         if len(sp) < 4:
31             # Falls am Ende eine Leerzeile ist
32             break
33         if turned:
34             feld.update({(sp[1], sp[0]): (sp[3], sp[2])})
35         else:
36             feld.update({(sp[0], sp[1]): (sp[2], sp[3])})
37     return feld
38
39
40 def schreibe(feld):
41     ''' Gibt das Feld in die Konsole aus '''
42     out = ["\t(y, %d)" % i for i in range(size)]
43     # Die obere Koordinatenleiste
44     out.append("\n")
45     for i in range(size):
46         out.append("(%d, x)\t" % i)
47         # Die Koordinate an der linken Seite
48         for j in range(size):
49             out.append("%s\t" % str(feld[(i, j)]))
50         out.append("\n")
51     print("".join(out))
52
```

```
53
54 def wert(t):
55     ''' Berechnet den Wert einer Position '''
56     ret = t[0] * size
57     ret += t[1] if t[0] % 2 == 0 else (size - t[1] - 1)
58     return ret
59
60
61 def oddeven(tosort, start):
62     '''
63     Führt auf tosort eine Odd- oder einen Even-Durchgang aus.
64     Dabei ist start = 0 ein Even-Durchgang und
65     start = 1 ein Odd-Durchgang
66     '''
67     ret = tosort.copy()
68     keys = [i for i in sortedkeys if i in tosort]
69     # Dadurch hat man die Schlüssel in O(n) sortiert und muss
70     # nur einmal die O(n*log(n)) sorted-Methode benutzen
71     ausg = {}
72     for i in range(start, len(keys) - 1, 2):
73         if wert(tosort[keys[i]]) > wert(tosort[keys[i + 1]]):
74             ret[keys[i]], ret[keys[i + 1]] = tosort[keys[i + 1]],
tosort[keys[i]]
75             ausg.update({keys[i]: DirToStr[diff(keys[i], keys[i + 1])]})
76             ausg.update({keys[i + 1]: DirToStr[diff(keys[i + 1], keys[i])]})
77     return ret, ausg
78
79
80 def hor(feld):
81     ''' Führt einen Spaltensortiervorgang aus. '''
82     ausg = {}  # Hier wird der Log gespeichert
83     endfeld = {}  # Hier wird das Feld gespeichert
84     for i in range(size):
85         column = dict(((j, i), feld[j, i]) for j in range(size))
86         # column enthält nun alle Positionen aus der i-ten Spalte von feld
87         a = oddeven(column, 0)  # Der Even-Durchgang
88         b = oddeven(column, 1)  # Der Odd-Durchgang
89         m = min(a, b, key=lambda x: bewertefeld(x[0]))
90         ausg.update(m[1])
91         endfeld.update(m[0])
92     return endfeld, ausg
93
94
95 def square(fe, lo):
96     ''' Hier wird die Quadratsortierung durchgeführt '''
```

```
97      f = fe.copy()
98      log = dict((i, lo[i][-1]) for i in lo)
99      # Die jeweils letzten Änderungen wurden aus dem Log extrahiert
100     for x in range(0, size - 1):
101         for y in range(0, size - 1):
102             doit = log[x, y] in ['S', 'O', '_']
103             doit = doit and log[x + 1, y] in ['N', 'O', '_']
104             doit = doit and log[x, y + 1] in ['S', 'W', '_']
105             doit = doit and log[x + 1, y + 1] in ['N', 'W', '_']
106             # doit ist genau dann wahr, wenn alle Veränderungen in diesem
107             # 2x2 Quadrat innerhalb von ihm durchgeführt wurden
108             if not doit:
109                 # Wenn es auch Veränderungen nach Außen gab lässt sich
110                 # der Log nicht sicher manipulieren
111                 continue
112             if x % 2 == 0:
113                 # So sind die Positionen nach ihrem Wert sortiert
114                 bereich = [(x, y), (x, y + 1), (x + 1, y + 1), (x + 1, y)]
115             else:
116                 bereich = [(x, y + 1), (x, y), (x + 1, y), (x + 1, y + 1)]
117             if wert(f[bereich[0]]) <= wert(f[bereich[1]]) <= wert(
118                     f[bereich[2]]) <= wert(f[bereich[3]]):
119                 # Dann ist alles schon optimal sortiert
120                 continue
121             # Hier werden die letzten Änderungen rückgängig gemacht
122             # Da die anderen Algorithmen ausschließlich Paartausche
    ausführen,
123             # impliziert ein 'S' der nördlichen Position ein 'N' der
    südlichen
124             # und deswegen ist klar, dass ein Tausch zwischen diesen
    beiden
125             # stattgefunden hat
126             if log[x, y] == 'S':
127                 f[x, y], f[x + 1, y] = f[x + 1, y], f[x, y]
128             elif log[x, y] == 'O':
129                 f[x, y], f[x, y + 1] = f[x, y + 1], f[x, y]
130             if log[x + 1, y + 1] == 'W':
131                 f[x + 1, y], f[x + 1, y + 1] = f[x + 1, y + 1], f[x + 1, y]
132             elif log[x + 1, y + 1] == 'N':
133                 f[x, y + 1], f[x + 1, y + 1] = f[x + 1, y + 1], f[x, y + 1]
134             allsq = allsquares(f, x, y)
135             m = min(allsq, key=lambda t: bewertefeld(t[0]))
136             # m ist die der Bewertung nach beste Umsortierung
137             for i in m[1]:
138                 # Der Log wird angepasst
```

```
                log[i] = DirToStr[(m[1][i][0] - i[0], m[1][i][1] - i[1])]
            f.update(m[0])  # Das Feld wird auf Vordermann gebracht
    return f, log, bewertefeld(f)


def allsquares(f, x, y):
    '''
    Hier werden alle gültigen Züge für ein 2x2-Quadrat mit
    der linken, oberen Ecke (x, y) berechnet.
    '''
    allsq = []
    for m in twoxtwo:
        logtmp = {}
        ftmp = {}
        for n in m:
            k = m[n]  # eckige Klammern sind nervig zu tippen
            logtmp.update({(n[0] + x, n[1] + y): (k[0] + x, k[1] + y)})
            ftmp.update({(k[0] + x, k[1] + y): f[(n[0] + x, n[1] + y)]})
        allsq.append((ftmp, logtmp))
    return allsq


def shearsort(filename):
    '''
    Der Shearsort-Algorithmus wird ausgeführt
    '''
    feld = lese(filename)
    print("Das ist das Feld:")
    schreibe(feld)
    print("Der Shearsort-Algorithmus wird durchgeführt:\n")
    global sortedkeys
    sortedkeys = sorted(feld, key=lambda x: wert(x))
    maxdiff = max(abs(diff(i, feld[i])[0]) +
                  abs(diff(i, feld[i])[1]) for i in feld)
    from math import log2
    log = dict((i, []) for i in feld)
    balken = fortschritt.balken(2 * int(log2(size) + 1) + 1, size=60)
    for n in range(int(log2(size)) + 1):
        while True:
            newlog = {}
            newlog1 = {}
            for j in range(size):
                row = dict(((j, k), feld[j, k]) for k in range(size))
                a, b = oddeven(row, 0)
                feld.update(a)
```

```
                newlog.update(b)
                row = dict(((j, k), feld[j, k]) for k in range(size))
                c, d = oddeven(row, 1)
                feld.update(c)
                newlog1.update(d)
            if len(newlog) + len(newlog1) == 0:
                break
            if len(newlog) > 0:
                for i in log:
                    log[i].append(newlog.get(i, '_'))
            if len(newlog1) > 0:
                for i in log:
                    log[i].append(newlog1.get(i, '_'))
        balken.update(2 * n + 1)
        while True:
            newlog = {}
            newlog1 = {}
            for j in range(size):
                col = dict(((k, j), feld[k, j]) for k in range(size))
                a, b = oddeven(col, 0)
                feld.update(a)
                newlog.update(b)
                col = dict(((k, j), feld[k, j]) for k in range(size))
                a, b = oddeven(col, 1)
                feld.update(a)
                newlog1.update(b)
            if len(newlog) + len(newlog1) == 0:
                break
            if len(newlog) > 0:
                for i in log:
                    log[i].append(newlog.get(i, '_'))
            if len(newlog1) > 0:
                for i in log:
                    log[i].append(newlog1.get(i, '_'))
        balken.update(2 * n + 2)
    while True:
        newlog = {}
        newlog1 = {}
        for j in range(size):
            row = dict(((j, k), feld[j, k]) for k in range(size))
            a, b = oddeven(row, 0)
            feld.update(a)
            newlog.update(b)
            row = dict(((j, k), feld[j, k]) for k in range(size))
            c, d = oddeven(row, 1)
```

```
229             feld.update(c)
230             newlog1.update(d)
231         if len(newlog) + len(newlog1) == 0:
232             break
233         if len(newlog) > 0:
234             for i in log:
235                 log[i].append(newlog.get(i, '_'))
236         if len(newlog1) > 0:
237             for i in log:
238                 log[i].append(newlog1.get(i, '_'))
239     balken.end()
240     print()
241     print("Sortiert in %s Schritten." % len(log[0, 0]))
242     return log, maxdiff
243 
244 
245 def bewertefeld(f):
246     '''
247     Das übergebene Feld wird bewertet. Diese ist eine laufzeitkritische
248     Methode und wurde deshalb vollständig per Hand programmiert.
249     '''
250     m = 0
251     anz = 0
252     for i in f:
253         t = (i[0] - f[i][0]) * size
254         t += i[1] if i[0] % 2 == 0 else (size - 1 - i[1])
255         t -= f[i][1] if f[i][0] % 2 == 0 else (size - 1 - f[i][1])
256         t = abs(t)
257         # t ist die Distanz von dem Paket an Position i zu seinem Ziel
258         if t > m:
259             anz = 1
260             m = t
261         elif t == m:
262             anz += 1
263     return (m, anz)
264 
265 
266 def main(filename, maxlen, turned=False, quadrate=False):
267     feld = lese(filename, turned=turned)
268     global sortedkeys
269     sortedkeys = sorted(feld, key=lambda x: wert(x))
270     log = dict((i, []) for i in feld)
271     if not turned:
272         print("Es wird das richtig orientierte Feld sortiert:\n")
273     else:
```

```
274         print("Es wird das gespiegelte Feld sortiert:\n")
275     maxdiff = max(abs(diff(i, feld[i])[0]) +
276                   abs(diff(i, feld[i])[1]) for i in feld)
277     print(
278         "Die minimale Anzahl an Schritten, dieses Tohuwabohu zu sortieren,
    beträgt %s." %
279         maxdiff)
280     print("Beginne mit der Sortierung...\n")
281     bewertung = bewertefeld(feld)
282     balken = fortschritt.balken(bewertung[0], size=60, turned=True)
283     while bewertung[0] > 0:
284         if len(log[0, 0]) > maxlen:
285             # So wird keine Zeit mehr für offensichtlich schlechtere
    Ansätze
286             # mehr verschwendet
287             balken.cancel()
288             print("Dieser Algorithmus ist schlechter als ein
    vorhergehender.")
289             return log, maxdiff
290         odd = oddeven(feld, 1)
291         even = oddeven(feld, 0)
292         horiz = hor(feld)
293         m = min(horiz, odd, even, key=lambda x: bewertefeld(x[0]))
294         for i in log:
295             log[i].append(m[1].get(i, '_'))
296         feld = m[0]
297         bewertung = bewertefeld(feld)
298         balken.update(bewertung[0])
299         if not quadrate:
300             continue
301         m = square(feld, log)
302         if m[2] < bewertung:
303             feld.update(m[0])
304             for i in log:
305                 log[i][-1] = m[1][i]
306             bewertung = m[2]
307             balken.update(bewertung[0])
308     out = []
309     print("\nSortiert in %d Schritten." % len(log[0, 0]))
310     swap = {'S': 'O', 'N': 'W', 'O': 'S', 'W': 'N', '_': '_'}
311     if turned:
312         newlog = {}
313         for i in log:
314             newlog.update({(i[1], i[0]): [swap[j] for j in log[i]]})
315         log = newlog
```

```
316     return log, maxdiff
317
318
319 def main1(filename, quadrate):
320     c = shearsort(filename) + ('der Shearsort-Algorithmus als der',)
321     print("_____________________________________________\n")
322     maxlen = len(c[0][0, 0])
323     a = main(filename, maxlen, False, quadrate) + \
324         ('die Sortierung mit normaler Orientierung als die',)
325     maxlen = min(maxlen, len(a[0][0, 0]))
326     print("_____________________________________________\n")
327     b = main(filename, maxlen, True, quadrate) + \
328         ('die Sortierung mit gespiegelter Orientierung als die',)
329     out = min(a, b, c, key=lambda x: len(x[0][0, 0]))
330     print(
331         "\nNachdem alle Algorithmen evaluiert wurden, erscheint " +
332         "%s Beste mit einer Länge von %s." %
333         (out[2], len(out[0][0, 0])))
334     return out
335
336
337 if __name__ == "__main__":
338     filename = input("Eingabefeld?\n")
339     quadrate = input(
340         "Soll die Quadratsortierung angwandt werden? (y,n)\n") == 'y'
341     begin = datetime.now()
342     out = main1(filename, quadrate)
343     end = datetime.now()
344     seconds = (end - begin).total_seconds()
345     if seconds > 60:
346         zeit = "%d Minuten und %f Sekunden" % (seconds // 60, seconds % 60)
347     else:
348         zeit = "%f Sekunden" % (seconds)
349     print("Das Sortieren hat %s gedauert" % zeit)
350     print("\n")
351     aus = dict()
352     for i in out[0]:
353         aus.update({i: ''.join(out[0][i])})
354     output = '\n'.join('%s %s %s' % (i, j, aus[i, j]) for i in range(size)
355                        for j in range(size))
356     if quadrate:
357         # Um die Ausgaben zu trennen
358         sq = "(mitsquares)"
359     else:
360         sq = ""
```

```
361     with open(filename + sq + '.out', 'w') as f:
362         f.write(output)
363     print("Die Ausgabe wurde nach %s%s.out geschrieben.\n" % (filename, sq))
364     animiert = input(
365         "Wollen Sie den Ablauf animiert haben? " +
366         "Dies ist bei Feldgrößen > 20 weniger sinnvoll. (y, n)\n") == 'y'
367     if animiert:
368         import animation
369         animation.main(filename, dirlog=out[0])
```

animation.py

```
1  #!/bin/env python3
2  from PyQt4 import QtGui
3  import gui
4  import sys
5
6  StrToDir = {'N': (-1, 0), 'W': (0, -1), 'S': (1, 0), 'O': (0, 1), '_': (0,
   0)}
7  DirToStr = {(-1, 0): 'N', (0, -1): 'W', (1, 0): 'S',
8              (0, 1): 'O', (0, 0): '_', None: '_'}
9
10
11 def lese(filename):
12     feld = {}
13     with open(filename) as f:
14         lines = f.readlines()
15     global size
16     size = int(lines.pop(0).strip())
17     for i in lines:
18         sp = list(map(int, i.split()))
19         if len(sp) < 4:
20             break
21         feld.update({(sp[0], sp[1]): (sp[2], sp[3])})
22     return feld
23
24
25 def main(feldeingabe, filename=None, dirlog=None):
26     feld = lese(feldeingabe)
27     if dirlog is None:
28         with open(filename) as f:
29             lines = f.readlines()
30         dirlog = {}
31         for i in lines:
32             if len(i.split()) < 3:
33                 break
```

```
            tup = (int(i.split()[0]), int(i.split()[1]))
            dirlog.update({tup: list(i.split()[2].strip())})
    log = {}
    for i in dirlog:
        log.update({i: [StrToDir[j] for j in dirlog[i]]})
    app = QtGui.QApplication(sys.argv)
    app.setApplicationName("Animation")
    fenster = gui.Fenster(feld, log)
    fenster.show()
    app.exec_()

if __name__ == '__main__':
    try:
        import filechooser
    except:
        pass
    feldeingabe = input('Felddatei:\n')
    filename = input('Logdatei:\n')
    main(feldeingabe, filename)
```

gui.py

```
from PyQt4.QtGui import *
from PyQt4.QtCore import *
from math import sqrt
import datetime


class Fenster(QMainWindow):

    fertig = pyqtSignal()

    def __init__(self, feld, log):
        super(Fenster, self).__init__()

        self.feld = feld
        self.current = feld.copy()
        self.log = log
        self.bloecke = {}
        self.add = lambda x, y: (x[0] + y[0], x[1] + y[1])

        self.breite = int(sqrt(len(feld)))
        self.slideTimer = QTimer(self)
        self.delayTimer = QTimer(self)
        self.steps = 50
        self.maxsteps = len(log[0, 0])
```

```
25          self.stepcounter = 0
26          self.slideTimer.setInterval(20)
27          self.delayTimer.setInterval(1000)
28
29          self.bestSize = 600 // self.breite
30          self.delta = self.bestSize / self.steps
31          if self.delta < 1:
32              self.delta = 1
33              self.steps = self.bestSize
34              self.slideTimer.setInterval(1000 // self.steps)
35          self.maxwert = max(map(self.wert, self.feld))
36
37          self.view = QGraphicsView()
38          self.view.setFixedSize(650, 650)
39          self.setFixedSize(self.sizeHint())
40          self.startButton = QPushButton("Start")
41
42          self.layout = QVBoxLayout()
43          self.layout.addWidget(self.view)
44          self.layout.addWidget(self.startButton)
45          self.setCentralWidget(QWidget())
46          self.centralWidget().setLayout(self.layout)
47          self.setWindowTitle("Animation")
48
49          viewRechteck = self.view.rect()
50          viewRechteck.setSize(QSize(600, 600))
51          self.szene = QGraphicsScene(QRectF(viewRechteck), self.view)
52          self.view.setScene(self.szene)
53
54          self.startButton.clicked.connect(self.main)
55          self.slideTimer.timeout.connect(self.animier)
56          self.delayTimer.timeout.connect(self.main)
57          self.fertig.connect(self.delayTimer.start)
58
59          self.erstellen()
60          self.fill()
61
62      def erstellen(self):
63          for i in range(self.breite):
64              for j in range(self.breite):
65                  block = QGraphicsRectItem(0, 0, self.bestSize,
    self.bestSize)
66                  block.setPos(i * self.bestSize, j * self.bestSize)
67                  self.szene.addItem(block)
68                  self.bloecke.update({(j, i): block})
```

```
69
70      def wert(self, pos):
71          return abs(pos[0] - self.feld[pos][0]) + \
72              abs(pos[1] - self.feld[pos][1])
73
74      def farbe(self, pos):
75          return int(120 * (1 - self.wert(pos) / self.maxwert))
76
77      def fill(self):
78          self.maxwert = max(map(self.wert, self.feld))
79          if self.maxwert == 0:
80              # Niemand hat die Absicht durch 0 zu teilen
81              self.maxwert = 1
82          for j in range(self.breite):
83              for i in range(self.breite):
84                  col = self.farbe((i, j))
85                  block = self.bloecke[i, j]
86                  bound = block.boundingRect()
87
88                  pos = (self.feld[i, j][0], self.feld[i, j][1])
89                  text = QGraphicsSimpleTextItem(str(pos), block)
90                  font = text.font()
91                  font.setPixelSize(self.bestSize // 4)
92                  text.setFont(font)
93                  textBound = text.boundingRect()
94                  text_x = bound.width() / 2 - textBound.width() / 2
95                  text_y = bound.height() / 2 - textBound.height() / 2
96                  text.setPos(text_x, text_y)
97                  block.setBrush(QColor.fromHsv(col, 0xff, 0xff, 0x90))
98
99      def ende(self):
100         self.startButton.setText("Schließen")
101         self.startButton.setEnabled(True)
102         self.startButton.clicked.connect(self.close)
103
104     @pyqtSlot()
105     def main(self):
106         self.stepcounter += 1
107         self.startButton.setEnabled(False)
108         self.delayTimer.stop()
109         self.move = {}
110         if len(self.log[0, 0]) == 0:
111             self.ende()
112             return
113         self.setWindowTitle(
```

```
114             "Frame %s von %s" %
115             (self.stepcounter, self.maxsteps))
116         newfeld = {}
117         for i in self.log:
118             self.move.update({i: self.log[i].pop(0)})
119             newfeld.update({self.add(i, self.move[i]): self.feld[i]})
120         self.feld = newfeld
121         self.counter = 0
122         self.slideTimer.start()
123 
124     def animier(self):
125         if self.counter == self.steps:
126             for i in self.bloecke:
127                 self.szene.removeItem(self.bloecke[i])
128             self.erstellen()
129             self.fill()
130             self.slideTimer.stop()
131             self.fertig.emit()
132         else:
133             self.counter += 1
134             for i in self.move:
135                 pos = self.bloecke[i].pos() + QPointF(self.delta *
self.move[i][1],
136                                                       self.delta *
self.move[i][0])
137                 self.bloecke[i].setPos(pos)
```