```
(* Die Erstellung der folgenden Listen hab ich exportiert,
weil ich sie so häufig gebraucht habe. *)
germanICPartL = Import["D:\\Muh\\Mathematica\\Wörterlisten\\germanICPartL.txt",
   "Table", CharacterEncoding → "UTF8"];
maxLänge = Length@germanICPartL;
neighboursList =
  Join @@ Table[Import["D:\\Muh\\Mathematica\\Wörterlisten\\neighbours_" <>
      ToString[i] <> ".wdx", "WDX"], {i, 2, maxLänge}];


(*Für Testzwecke kann diese Liste hier genommen
   werden. Es kann eine beliebige "wörter"-liste definiert werden.
    germanICPartL und neighboursList werden dann ineffizient
   erstellt. Das geht also nicht für längere Listen.

    wörter={"ab","aus","maus","haus","kaue","kauen","alter","alte","trinken","nehmen",
  "nehme","nehmer","nahmst","nahm","nahmt","nehmt","remis","bruder","martin",
  "martini","ohrstöpsel","klaus","klause","klausel","klauseln","altern"};
maxLänge=Max[StringLength/@wörter];
germanICPartL=Table[Cases[wörter,x_/;StringLength[x]==i],{i,1,maxLänge+1}];
neighboursList=Association[
  Table[w→Cases[wörter,x_/;DamerauLevenshteinDistance[w,x]==1],{w,wörter}]];
Graph[Union@Flatten@Table[Sort[w←→n],{w,Flatten@germanICPartL},
    {n,neighboursList[w]}],VertexLabels→Automatic]
*)

(* Wir nutzen germanICPartL, also die Wörter partitioniert nach Länge,
damit nicht immer die ganze Liste durchsucht werden muss. *)
(* Man kann beobachten,
dass über die Hälfte der Einträge in "germanIC" entweder isolierte Knoten
 oder isolierte Paare sind. Diese können wir zuerst mal aussortieren,
um deutlich mehr als Faktor 4 in der Hauptschlefie zu
 sparen. Faktor 4 wegen quadratischer Laufzeit,
außerdem deutlich mehr, da die Komponenten da schneller wachsen,
als wenn in vielen Durchläufen einfach isolierte Paare betrachtet werden. *)

(* Eine leere Liste an germanICPartL anhängen,
damit ..[[maxLänge+1]] keinen Fehler ausgibt. *)
AppendTo[germanICPartL, {}];
(* isolierteKnoten sind offenbar alle isolierte Knoten, aber sortiert nach Länge. *)
isolierteKnoten =
  Table[Cases[germanICPartL[[i]], x_ /; neighboursList[x] == {}], {i, 1, maxLänge}];

(* Diese isolierten Knoten müssen wir nicht mehr betrachten. Wir exportieren
 sie also separat und entfernen sie aus neighboursList und germanICPartL: *)
(*Export["D:\\Muh\\Mathematica\\Wörterlisten\\Levenshtein_isolierte_Knoten.list",
  Flatten@isolierteKnoten,"List"];*)
Do[germanICPartL[[i]] = Complement[germanICPartL[[i]], isolierteKnoten[[i]]],
 {i, 1, maxLänge}]
Do[neighboursList[w] =., {w, Flatten@isolierteKnoten}]
(* So sparen wir schon mal viele Wörter: *)
Print["Es gibt ", Length /@ isolierteKnoten // Total, " isolierte Wörter."]

(* Alle Wörter, die nur einen Nachbarn haben,
können auch separat behandelt werden. *)
```

```
komponentenListe = {};

(* Erklärung hierfür siehe unten bei "nachAußen". *)
nachAußenList = {};

(* Der Counter c1N zählt, wie oft wir Wörter mit genau einem Nachbarn entfernen *)
c1N = 0;
While[True,
 (* Hier erzeugen wir alle Wörter mit genau einem Nachbarn. *)
 einNachbar = Table[
   Cases[germanICPartL[[i]], x_ /; Length[neighboursList[x]] == 1], {i, 1, maxLänge}];
 (* Hier lassen sich wieder viele Wörter sparen: *)
 Print["Es gibt nun ", Length /@ einNachbar // Total,
  " Wörter mit genau einem Nachbarn."];
 (* Nach einigen Durchgängen ist einNachbar klein und wir geben es aus: *)
 If[Total[Length /@ einNachbar] < 50,
  Print["Flatten@einNachbar: ", Flatten@einNachbar]];
 If[c1N == 100, Break[]];
 (* Falls es kein Wort mit genau einem Nachbarn mehr gibt, brechen wir ab: *)
 If[Total[Length /@ einNachbar] == 0, Break[]];

 (* Hier betrachten wir alle einNachbarn, mit entsprechenden Nachbar. Alle Einträge,
 die dann doppelt vorkommen, müssen EinNachbar-EinNachbar Paare sein,
 also isolierte Paare. Diese können also direkt als Zusammenhangskomponente
  exportiert werden und müssen nicht mehr betrachtet werden.
 *)
 isoliertePaare = Cases[Tally[
    (* Hier fügen wir den entsprechenden Nachbarn hinzu:  ... {wort, nachbar}...
       Flatten, weil neighboursList immer eine Liste ist.
      Sort, damit wir danach Tally anwenden können *)
    (x ↦ Sort@Flatten[{x, neighboursList[x]}]) /@ Flatten[einNachbar]
   ],
   (* In diesem Tally suchen wir alle Paare,
   die doppelt vorkommen, also deren zweiter Tally-Eintrag 2 ist.
      Die 2 wird dann aber nicht mehr benötigt,
   also können wir gleich aus {{v,w},2} → {v,w} machen. Dafür das "->x[[1]]"
   *)
   x_ /; x[[2]] == 2 → x[[1]]];
 Print["Es gibt nun ", Length@isoliertePaare,
  " isolierte Paare, die nun eine eigene Zusammenhangskomponenten bilden."];
 If[c1N++ > 0,
  (* Die isolierten Paare bilden jeweils eine eigene Zusammenhansgkomponente:*)
  komponentenListe = Join[komponentenListe, isoliertePaare]
  (*
  Behandele den ersten Durchlauf unterschiedlich,
  weil die isolierten Paar im Ursprungsgraphen separat exportiert werden sollen.
  ,Export["D:\\Muh\\Mathematica\\Wörterlisten\\Levenshtein_isolierte_Paare.table",
    isoliertePaare,"Table"];
  *)
 ];

 (* Das hier sind die restlichen EinNachbarn, also die wörter,
 die nur einen Nachbar haben, aber nicht Teil eines isolierten Paares sind. *)
 restlicheEinNachbarn = Complement[Flatten@einNachbar, Flatten@isoliertePaare];
```

```
 (* nach Außen zeigt von den Wörter,
 die mehr als ein Nachbar haben ("innere Wörter") auf die Wörter,
 die nur ein Nachbarn haben ("äußere Wörter"). Da wir nach Wegnehmen der
  äußeren Wörter neue äußere Wörter bekommen können, ist "nachAußen" im
  jeden Schritt anders und wird in der "nachAußenList" gespeichert. *)
 nachAußen = <||>;
 Do[
  (* Das Wort "innen" ist der eindeutige Nachbar des Wortes außen: *)
  innen = neighboursList[außen][[1]];
  (* Die Wörter innen sollen nicht mehr nach außen zeigen: *)
  neighboursList[innen] = Complement[neighboursList[innen], {außen}];
  (* Dafür sollen die Wörter innen durch eine spezielle Liste nach außen zeigen,
  die wir anwenden können,
  nachdem wir die Zusammenhangskomponente berechnet haben. Da ein Wort innen
   auf mehrere Wörter außen zeigen kann, müssen wir "Join" anwenden.*)
  nachAußen[innen] = Join[Lookup[nachAußen, innen, {}], {außen}],
  {außen, restlicheEinNachbarn}
 ];
 AppendTo[nachAußenList, nachAußen];

 (* Die EinNachbarn haben wir nun fertig behandelt. Wir können sie also
  nun komplett aus germanPartL und auch aus der neighboursList streichen.
 *)
 Do[germanICPartL[[i]] = Complement[germanICPartL[[i]], einNachbar[[i]]], {i, 1, maxLänge}];
 Do[neighboursList[w] =., {w, Flatten@einNachbar}]
]

(* Bereinige hier nochmal die isolierten Knoten. Hier steht fast das Gleiche wie
 am Anfang. Allerdings werden hier die isolierten Knoten nicht exportiert,
sondern in die komponentenListe gesteckt.*)
isolierteKnoten =
  Table[Cases[germanICPartL[[i]], x_ /; neighboursList[x] == {}], {i, 1, maxLänge}];
komponentenListe = Join[komponentenListe, Partition[Flatten@isolierteKnoten, 1]];
Do[germanICPartL[[i]] = Complement[germanICPartL[[i]], isolierteKnoten[[i]]],
  {i, 1, maxLänge}];
Do[neighboursList[w] =., {w, Flatten@isolierteKnoten}];
(* So sparen wir wieder einige viele Wörter: *)
Print["Es gibt ", Length /@ isolierteKnoten // Total,
 " isolierte Wörter nach Entfernen der \"EinNachbarn\"."]

(* Graphen ausgeben. Das geht natürlich nur im Testmodus, wenn der Graph klein ist.
  Graph[
   Union@Flatten@Table[Sort[w↔n],{w,Flatten@germanICPartL},{n,neighboursList[w]}],
   VertexLabels→Automatic]*)


(* Nun haben wir nur noch Wörter mit mindestens zwei Nachbarn.
   lä=1;
c=0;
While[True,
 (* Falls es keinen mit Lange lä mehr gibt, erhöre lä um eins. *)
 While[lä≤maxLänge&&Length[germanICPartL[[lä]]==0,lä++];
 (* Wenn die Länge das Maximum überschritten hat, sind wir fertig. *)
 If[lä>maxLänge,Break[]];
 (* Nun nehme das erste Wort in germanICPartL. *)
```

```
 aktuell={germanICPartL[[lä,1]]};
 If[Mod[c++,1000]==0||(c<1000&&Mod[c,10]==0),
  Print["aktuell: ",aktuell,". Noch ",Total[Length/@germanICPartL]," Wörter"]];
 (* Hier wird die Zusammenhangskomponente des aktuellen Wortes berechnet. *)
 komponente=aktuell;
 germanICPartL[[lä]]=Rest[germanICPartL[[lä]]];
 While[True,
  (* aktuell sind alle Nachbarn von den davor betrachteten. Die,
  die wir schon betrachtet haben (komponente), werden rausgeschmissen. *)
  aktuell=Complement[Union@Flatten[neighboursList/@aktuell],komponente];
  (* Wenn aktuell leer ist, haben wir die ganze Komponente also erreicht. *)
  If[Length[aktuell]==0,Break[]];
  (* Alle aktuellen Wörter werden in die komponente aufgenommen*)
  komponente=Join[komponente,aktuell];
  (* germanICPartL wird angepasst. *)
  Do[germanICPartL[[i]]=
    Complement[germanICPartL[[i]],Cases[aktuell,x_/;StringLength[x]==i]],
   {i,1,maxLänge}];
 ];
 AppendTo[komponentenListe,komponente]
]
*)
```

```
Es gibt 446894 isolierte Wörter.

Es gibt nun 595409 Wörter mit genau einem Nachbarn.

... Part: Part specification x[[1]] is longer than depth of object.

Es gibt nun 194464 isolierte Paare, die nun eine eigene Zusammenhangskomponenten bilden.

Es gibt nun 16927 Wörter mit genau einem Nachbarn.

Es gibt nun 2839 isolierte Paare, die nun eine eigene Zusammenhangskomponenten bilden.

Es gibt nun 3063 Wörter mit genau einem Nachbarn.

Es gibt nun 209 isolierte Paare, die nun eine eigene Zusammenhangskomponenten bilden.

Es gibt nun 916 Wörter mit genau einem Nachbarn.

Es gibt nun 25 isolierte Paare, die nun eine eigene Zusammenhangskomponenten bilden.

Es gibt nun 388 Wörter mit genau einem Nachbarn.

Es gibt nun 1 isolierte Paare, die nun eine eigene Zusammenhangskomponenten bilden.

Es gibt nun 86 Wörter mit genau einem Nachbarn.

Es gibt nun 2 isolierte Paare, die nun eine eigene Zusammenhangskomponenten bilden.

Es gibt nun 21 Wörter mit genau einem Nachbarn.

Flatten@einNachbar: {erzbach, sattsam, einstand, hochmuts, textraum, feierraum, neulingen,
  psychogen, spielgeld, wehrbaues, wendlands, aufzuhaben, blockbaues, brandseite, doppelwall,
  metallbaus, weltraumes, anzuzettelnd, umherwerfend, herausquellend, herunterbrachen}

Es gibt nun 0 isolierte Paare, die nun eine eigene Zusammenhangskomponenten bilden.

Es gibt nun 7 Wörter mit genau einem Nachbarn.

Flatten@einNachbar: {erbach, sittsam, hochmuss, spielgel, feuerraum, weltraums, doppelfall}

Es gibt nun 0 isolierte Paare, die nun eine eigene Zusammenhangskomponenten bilden.
```

```
Es gibt nun 2 Wörter mit genau einem Nachbarn.
Flatten@einNachbar: {weltraum, feuerbaum}
Es gibt nun 0 isolierte Paare, die nun eine eigene Zusammenhangskomponenten bilden.
Es gibt nun 1 Wörter mit genau einem Nachbarn.
Flatten@einNachbar: {weltbaum}
Es gibt nun 0 isolierte Paare, die nun eine eigene Zusammenhangskomponenten bilden.
Es gibt nun 1 Wörter mit genau einem Nachbarn.
Flatten@einNachbar: {wellbaum}
Es gibt nun 0 isolierte Paare, die nun eine eigene Zusammenhangskomponenten bilden.
Es gibt nun 1 Wörter mit genau einem Nachbarn.
Flatten@einNachbar: {fellbaum}
Es gibt nun 0 isolierte Paare, die nun eine eigene Zusammenhangskomponenten bilden.
Es gibt nun 1 Wörter mit genau einem Nachbarn.
Flatten@einNachbar: {fallbaum}
Es gibt nun 0 isolierte Paare, die nun eine eigene Zusammenhangskomponenten bilden.
Es gibt nun 1 Wörter mit genau einem Nachbarn.
Flatten@einNachbar: {fallraum}
Es gibt nun 0 isolierte Paare, die nun eine eigene Zusammenhangskomponenten bilden.
Es gibt nun 1 Wörter mit genau einem Nachbarn.
Flatten@einNachbar: {hallraum}
Es gibt nun 0 isolierte Paare, die nun eine eigene Zusammenhangskomponenten bilden.
Es gibt nun 0 Wörter mit genau einem Nachbarn.
Flatten@einNachbar: {}
Es gibt 41686 isolierte Wörter nach Entfernen der "EinNachbarn".
```

```
(* Das hier ist ineffiziert programmiert,
weil wir c1N-mal die gesamte Wörterliste durchgehen.
   Das ist aber egal, weil es auch so nicht lange dauert. *)
Do[
 Do[
  (* Wir vergrößern die Komponente i:*)
  komponentenListe[[i]] = Join[
    (* Wir wenden die "nachAußen[[cc]]"-Funktion auf jedes Element an. *)
    Flatten[(x ↦ Lookup[nachAußenList[[cc]], x, {}]) /@ komponentenListe[[i]]],
    (* und fügen es zu den Wörtern hinzu, die schon in der Komponente sind. *)
    komponentenListe[[i]]],
  {i, Length@komponentenListe}
 ],
 (* Wir müssen zuerste die innersten Wörter hinzufügen,
 daher gehen wir von c1N nach 1 *)
 {cc, c1N, 1, -1}
]
```

```
SortBy[komponentenListe, Length]
```

```
{{aalbestand, aalbestandes, aalbestands}, ··· 44 760 ···,
 {zweittanks, zweitbrandes, zweibundes, zweilandes, zweitbrand,
  zweibund, zweihand, zweittank, zweitbandes, zweitbank, zweibandes,
  zweitband, zweibades, zweiband, zweirades, zweibad, zweirads, zweirad}}
```

large output | **show less** | **show more** | **show all** | **set size limit...**

```
Total[Length /@ germanICPartL]
```

876 239