HOMOTOPY THEORY OF WORDS

The rather simple idea is to create a homotopy between different words by changing only one letter at a time where each intermediate word has to exist in a given 'word-universe'.
It is useful to adapt to the following convention of vocabulary. Two words are *conected* if they differ by exactly one letter. Later I will moreover distinguish the property of being *ordered-connected* and *order-less-connected*. The difference between those properties is simply that the latter one consideres two words being equal up to anagrams.

Creating A Thesaurus

To proceed, we obviously first need to create a thesaurus, which in our case is a sorted list of lower case words without multiples. After some technical but self-explanatory functions to add words from a string to a given thesaurus, the question at hand really is, what is the most efficient way to obtain new words.

```
LearnWords[known_, s_String] := Module[{thesaurus1, n, words},

    words = Append[StringCases[s, WordCharacter .. ], "a"];
                                n = ToLowerCase /@ words;

    thesaurus1 = Sort[DeleteDuplicates[known ~ Join ~ n]];
                                Drop[thesaurus1,
     Det[Position[thesaurus1, "a"]] - 1]
                          ];

LearnWords[{"mathematics"}, "Hello, this is a beautiful day"]
```

```
{a, beautiful, day, hello, is, mathematics, this}
```

```
JoinThesauri[t1_, t2_] :=
 Drop[#, Det[Position[#, "a"] - 1]] & @ Sort[DeleteDuplicates[t1 ~ Join ~ t2]]

JoinThesauri[{"a", "day", "hello", "is", "lennart", "nice", "this"}, {"test"}]
```

```
{a, day, hello, is, lennart, nice, test, this}
```

Stealing words from *RandomWord[]*

Learning new words by using wolframs built-in *RandomWord* function this is kind of cheating, but as it turns out, it is anyway more efficient to use random wikipedia articles. One could argue that wikipedia uses in certain articles very specific words, but this argument pales after adding about 80 000 words or

SO.

```
RandomThesaurus[n_] := Module[{thesaurus = {}, a},
                        a = While[Length[thesaurus] < n,
                                  w = RandomWord[];
      thesaurus = LearnWords[thesaurus, w];
                                ];
                        thesaurus
                        ];
```

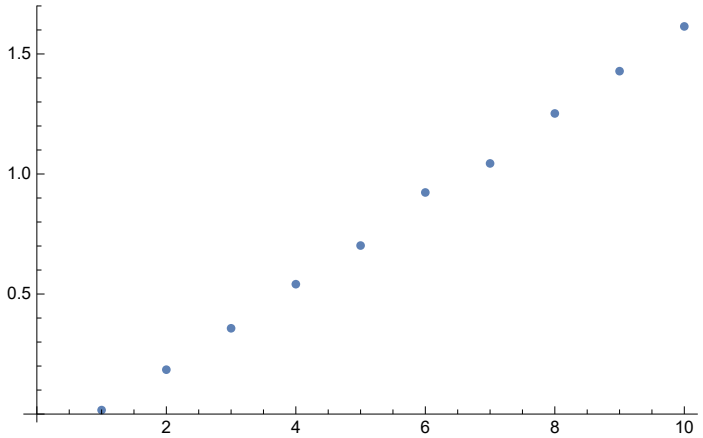
```
RandomThesaurus[100] // Timing
```

```
{1.64063, {a, affection, allege, amigo, apt, array, asphyxiation, aspirin, atavistic,
  bingo, blustery, bottomed, bugged, carious, cheese, chosen, comfort,
  concernedly, congealed, coral, cortex, debriefing, decidedly, delightfully,
  demob, devilry, devoted, discomfited, disconnected, diverse, doff, dribbler,
  effendi, exemplifying, explanatory, fashionable, fecund, fluffy, foppishness,
  gaolbreak, governing, gropingly, headless, heartlessly, hysterically,
  impoliteness, impressionist, infertile, latex, lavatory, lilting, loveless,
  luminescence, menstrual, midstream, misinformation, mohair, moonstone,
  moroseness, moulting, mountainous, mutinous, nincompoop, northwestward,
  obstreperous, outhouse, packsaddle, peewit, piccalilli, plagiarizer, plummy,
  presbyopia, prion, prisoner, pyramiding, quantization, reactant, realization,
  regatta, retch, sailing, sandblast, scion, selective, self, signing, smelter,
  stimulate, stonework, storyteller, streptococci, superordinate, swami,
  thrusting, tippler, toughie, tuba, unilateralism, vibrator, voicelessness}}
```

```
l = Table[Timing[RandomThesaurus[i]][[1]], {i, 1, 100, 10}]
```

```
{0.015625, 0.125, 0.25, 0.375, 0.453125,
 0.640625, 0.71875, 0.828125, 0.984375, 1.07813}
```

```
ListPlot[l]
```



Harvesting words from Wikipedia

The following function starts with a random word generated by Mathematica and searches Wikipedia for any results for this word. By looping through this process, one can assure that eventually the search for some word gives back a list of associated articles. Subsequently the function converts each of these suggested articles in a string and *learns* words in these articles by using the previously written function *LearnWords[thesaurus_ , s_String]*.

```
AddRandomWikipediaArticles[] :=
  Module[{thesaurus = {}, title, article, word, ass = {}, a},
                                        While[Length[ass] == 0,
                                              word = RandomWord[];
                                              ass = WikipediaSearch[word];
                                         ];
                                        Do[
                                              title = "Title" /. a;
                                              article = WikipediaData[title];

    (*Print["learning words from ",title];*)

    thesaurus = LearnWords[thesaurus, article];
                                               , {a, ass}];
                                        thesaurus
                                   ];
```

By uncommenting out the *Print* function above one can see which article is being learned. And a direct comparison shows below that going through Wikipedia articles is about 10-20 times faster than using the built-in function to aquire new words.

```
AbsoluteTiming[Length[AddRandomWikipediaArticles[]]]
```

```
{17.4815, 11 410}
```

```
AbsoluteTiming[RandomThesaurus[100]]
```

```
{1.44435, {a, abducting, acrobat, affected, affidavit, afflict, annealing, armband,
  armorer, attractable, belling, bright, chine, clanging, commercialized,
  commissary, conscientiously, coronation, creepiness, cribbage, crutch,
  cylindrical, dais, declassified, digestibility, elope, exception, fieldworker,
  flatulent, foliage, foolishly, freckled, funny, fusion, geomorphology,
  gunner, heartening, heavyset, henna, hint, humming, ichthyologist, icon,
  impiously, insulator, interwoven, jitterbug, jutting, lakeside, layabout,
  legs, loudspeaker, lunch, madness, magnetosphere, mangle, mosaic, nihilist,
  nineteen, nonalignment, oath, oft, pewter, pianoforte, pilferer, pin, preordain,
  prosciutto, punnet, pyrimidine, rearrangement, recitalist, reducible, referable,
  refuse, rehabilitative, relapsing, requisition, rhomboidal, ridiculous,
  righteousness, salutatory, shoreward, shortage, slit, slog, spokesperson,
  stratosphere, strengthened, supervising, termite, thoughtless, touchiness,
  underfoot, unfeigned, unruly, unwittingly, usurpation, westerly, ytterbium}}
```

This function uses everything from above together to build a thesaurus for the next part of this project. Without a limit it would run forever, and maybe this would be an interesting thing to do on the math servers, but for the sake of this project the functions takes a time limit.

```
FillThesuarus[timeLimit_] := Module[{t = {}, time, saturation = {}},
                              time = AbsoluteTime[];
                              While[AbsoluteTime[] - time < timeLimit,
                                    t = JoinThesauri[t, AddRandomWikipediaArticles[]];
                                    Print[Length[t], "  ", DuplicateFreeQ[t]];

    AppendTo[saturation, {AbsoluteTime[] - time, Length[t]}];
                               ];
                              {t, saturation}
                              ];

thesaurus = FillThesuarus[60][[1]];
```

```
15 028  True
18 974  True
19 458  True
23 067  True
23 067  True
23 092  True
28 132  True
28 161  True
28 277  True
28 447  True
35 504  True
```

```
ContainsAll[thesaurus, {"math", "is", "great"}]
```

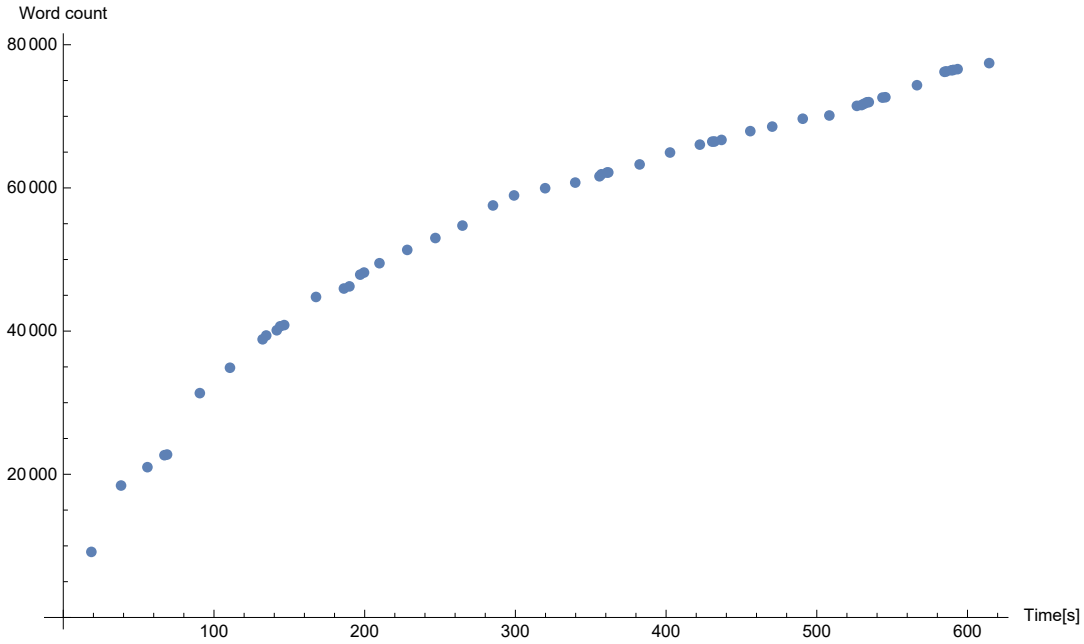
```
True
```

Saturation

Let's see if we can fund some kind of saturation. (The following data comes from a 10 min long collection of words, you may repeat the process if you are curious.)

```
saturation = {{18.632884`7.721825073746445, 9159},
   {38.333733`8.035126107098352, 18429}, {55.828945`8.198404414557801, 20993},
   {67.131084`8.278468653512972, 22666}, {68.860866`8.289517473622634, 22758},
   {90.609751`8.408719930440114, 31335}, {110.608019`8.495331607636478, 34882},
   {132.280264`8.573040046487176, 38855}, {134.666401`8.580804247121954, 39387},
   {141.648897`8.602758190575663, 40116}, {143.919576`8.609664864364792, 40669},
   {146.59818`8.617673572117157, 40841}, {167.697352`8.676071198494022, 44768},
   {186.210599`8.721549390636508, 45948}, {189.87403`8.730010561713891, 46247},
   {197.040631`8.746100783107469, 47889}, {199.553687`8.751604749815272, 48179},
   {209.809489`8.773370119522946, 49483}, {228.241883`8.809940335097746, 51342},
   {246.913404`8.84408966028179, 53002}, {264.876947`8.874589155544218, 54739},
   {285.132869`8.906592277457559, 57551}, {299.10968`8.927375461701338, 58950},
   {319.753251`8.956359962234414, 59960}, {339.720095`8.982666230355688, 60745},
   {355.808364`9.002761146339864, 61625}, {357.081758`9.004312657740005, 61906},
   {360.816974`9.008831953466593, 62150}, {361.64176`9.009823567651079, 62159},
   {382.445323`9.034114347713073, 63282}, {402.608188`9.056427596554961, 64944},
   {422.402885`9.077271869251136, 66046}, {430.659854`9.085679382326376, 66467},
   {432.105843`9.087135132442013, 66484}, {436.7718`9.091799584044882, 66704},
   {455.882618`9.110398027131154, 67931}, {470.434135`9.12404382039119, 68565},
   {490.685754`9.142348442858584, 69665}, {508.338625`9.157698103369944, 70118},
   {526.583281`9.173012059650926, 71462}, {529.718598`9.175590214434184, 71573},
   {531.380225`9.176950381859317, 71753}, {533.326138`9.178537862217283, 71962},
   {534.505384`9.179497077652021, 71980}, {543.586761`9.186813864487515, 72615},
   {545.513094`9.188350172955065, 72665}, {566.479432`9.204729139414601, 74347},
   {584.749826`9.218515094747147, 76215}, {585.938439`9.219396983222213, 76278},
   {589.448325`9.221990731424047, 76431}, {590.674931`9.22289353273127, 76481},
   {593.448347`9.224927917984392, 76591}, {614.394302`9.23999217281527, 77433}};
```

```
ListPlot[saturation, AxesLabel → {"Time[s]", "Word count"}]
```



To get an estiamate of the amount of words we are using here, this might be the right moment to quote not only single words, but an entire sentence from the wikipedia article on vocabulary:
"*A 1995 study shows that junior-high students would be able to recognize the meanings of about 10,000–12,000 words, whereas for college students this number grows up to about 12,000–17,000 and for elderly adults up to about 17,000 or more.*"

Connecting Words And Building Homotopies

This function takes subsets of words with a specific length, since by our definition of homotopic this is the subset in which the homotopies will live anyway.

```
SpecificLength[thesaurus_, n_] := Module[{result = {}},

   Do[If[Length[Characters[e]] == n, AppendTo[result, e]], {e, thesaurus}];
                                    result
                                ];
```

```
test = SpecificLength[thesaurus, 3]
```

```
{a16, abc, abs, ace, ach, acm, act, add, ads, aet, aft, age, åge, agm, ago, aid, aim, air,
 åke, all, alo, ama, amr, ams, amt, and, ane, ann, ant, anu, any, api, app, apt, arc,
 ard, are, arm, årø, art, arx, ash, ask, ata, atp, aug, aus, avr, awe, ax4, ax6, bac,
 bad, bar, bay, bbc, bcd, bce, bch, bdb, bdo, bed, bee, bel, ben, bet, bf3, bid, big,
 bio, bit, bkm, bma, bmi, bob, bow, box, boy, bre, btu, bud, bus, but, buy, bwl, c3v,
 caa, cab, cad, can, cap, car, cat, caw, cbs, cbt, ccl, cct, cen, ceo, ch4, cha,
 chi, chn, clm, cm², cnc, co2, cod, com, con, cos, cox, cp1, cpl, crc, cri, cs1,
```

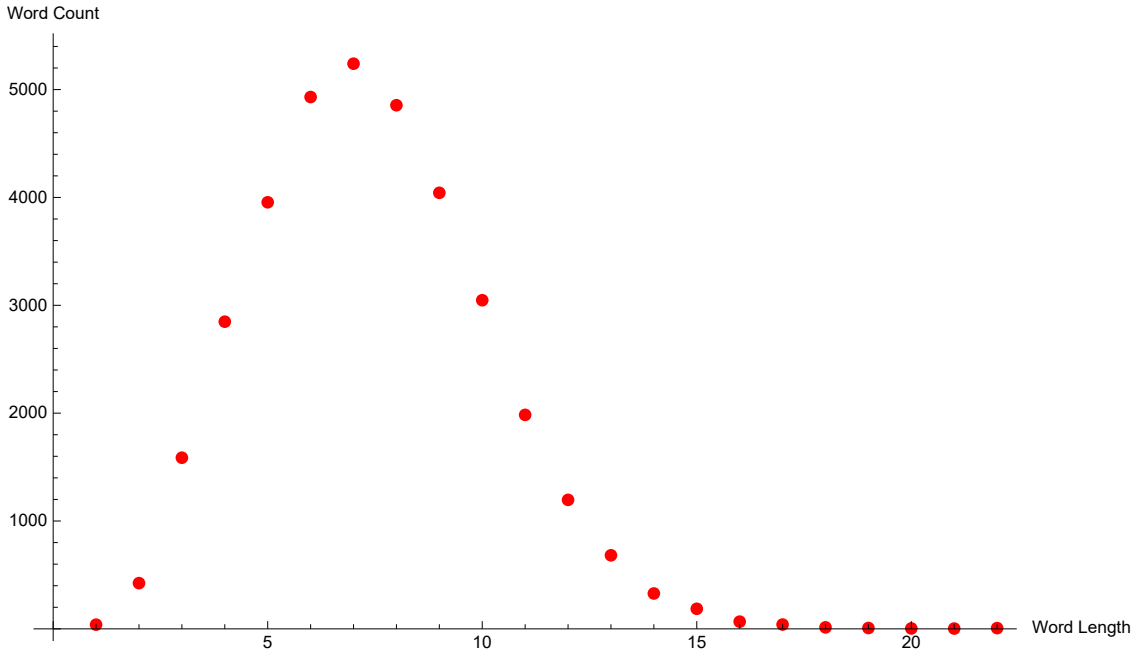
csi, cta, cue, cup, cut, cvm, cyr, d2m, d3h, d6h, dag, dai, dam, dan, das, dax,
day, dbt, dcm, ddg, deb, dec, deg, del, dem, der, des, det, dev, did, die, din,
dip, diu, div, djk, dog, doi, don, dor, dos, dot, drw, dry, dsp, dsr, dsw, dsx, dtl,
due, dvd, e2t, e39, ean, ear, eat, ebu, ecm, edd, edn, eds, egg, eiθ, elm, ely,
emg, end, enl, eos, eps, era, erg, esa, eta, etc, eve, evi, exp, eye, faa, fan,
faq, far, fat, fay, fbr, fda, fed, few, fft, fig, fil, fit, fix, flo, fly, fog,
for, fps, fpu, fry, ft2, fun, für, fxv, gad, gal, gap, gas, gay, gen, geo, get, ghz,
gis, gln, gmp, gnu, god, got, gov, gps, gta, gtm, gui, gun, guy, g$\alpha\beta$, h2n, hab,
had, hal, has, hat, hay, hbo, hcb, hch, hcl, hcn, hen, her, het, hex, him, his,
hit, hms, hoe, hoh, hot, how, hpl, hub, hug, i12, ian, iau, ibm, ibn, icb, ice, ida,
ide, ign, iii, iij, iji, ill, ils, imp, imu, inc, ine, inn, ion, ios, ipo, ipr,
ism, iss, ist, ith, its, ivd, ixx, ixy, jan, jay, jct, jet, jfi, jim, job, joe,
jon, joy, jpg, jth, kai, kei, kek, ken, ker, kew, key, kid, kim, kit, km2, knn, lab,
lar, las, law, lax, lay, lb2, lbf, lbm, lbp, lbs, lcb, lcf, lda, lds, led, lee,
leg, leo, lep, les, let, lev, lgm, lie, lin, lip, lit, llc, lms, loa, loc, log,
loo, los, lot, low, lpo, lpp, lrs, ltd, lwl, lxx, lyn, mac, mad, mal, man, map, max,
may, maz, mdf, mdt, mei, men, meo, met, mhz, mia, mid, min, mix, mkx, mod, mol,
mph, mpn, mr2, mra, mri, mrs, msc, mse, mtm, mtv, mud, muk, mul, mvp, myr, nås,
nat, ncs, ned, nef, neo, nes, net, new, ngr, nh3, nhs, nmr, nno, nok, non, nop,
nor, not, nov, now, npc, npi, nrk, nsm, nth, nut, oak, ocs, oda, odd, off, ohr, oil,
ola, old, ole, ona, one, ono, ons, ook, oph, orb, ore, org, ots, our, out, oux,
ove, own, oyz, p28, p91, pad, pål, par, pat, pde, pdf, pen, per, pet, pex, phd,
pic, pit, ply, pmo, pod, pom, pop, pot, pov, ppi, ppm, pre, pro, psi, psr, pub,
put, pvc, q36, q37, qld, qtq, qua, que, qvq, qxx, qxy, qxz, qyx, qyy, qyz, qzx, qzy,
qzz, rad, raf, ran, rar, raw, ray, rca, re1, re2, re3, rec, red, ref, req, rer,
rev, rhs, ria, rib, rig, rim, rlc, rly, rms, rmv, rna, rob, rod, rom, rot, row,
roy, rp3, rpm, run, rur, rye, s3l, s3r, s43, sab, sac, sam, san, sap, sas, saw, sax,
say, sb2, sbc, scd, sci, sde, sdk, sea, see, ser, set, seu, sex, sga, sgr, sgv,
she, shy, sih, sin, sir, sit, siv, six, sjø, ski, sky, slr, soc, sod, som, son,
sow, sp1, spy, sst, ssx, stc, stv, sub, sum, sun, sur, svm, syn, tal, tan, tap,
tau, tax, ted, ten, tet, tfc, the, tie, til, tip, tiw, tms, tnf, tno, toe, tom, too,
top, tor, toy, try, tuf, two, ubu, uci, und, ups, uri, usa, usb, use, ut1, ut4,
ut6, uwe, ux2, uy2, uz2, v2i, v2m, vak, val, van, vcb, vcr, veb, via, vib, vip,
vol, von, vox, vpk, vq3, vye, vδt, war, was, wax, way, web, wed, wen, wet, who,
why, wie, wii, win, wit, won, wrt, www, xi1, xip, xiv, xmm, xvx, xxy, xyz, yam,
yaw, yeo, yet, yew, yin, you, yℓm, zag, zig, zyz, βtx, δri, $\delta\xi$2, μ12, ρsℓ, ωij, ωnu}

Interestingly the number of words with a specific length obeys a normal distribution.

```
d = Table[Length[SpecificLength[thesaurus, i]], {i, 22}]
```

```
{39, 424, 1587, 2848, 3955, 4931, 5240, 4855,
 4043, 3047, 1984, 1196, 682, 328, 186, 67, 40, 14, 8, 4, 3, 7}
```

```
ListPlot[d, PlotStyle → Red, AxesLabel → {"Word Length", "Word Count"}]
```



Connected Words

ConnectedToOrdered and *ConnectedToOrderless* are the implementations of what was described in the very beginning.

```
ConnectedToOrderless[thesaurus_, word_] :=
  Module[{result = {}, l = Length[Characters[word]], pool},

   pool = DeleteCases[SpecificLength[thesaurus, l], word];

   Do[If[Length[Characters@word⋂Characters@w] == l - 1,
     AppendTo[result, w]], {w, pool}];
                                    result
                                ];
```

```
ConnectedToOrderless[thesaurus, "lee"]
```

```
{ale, bel, cle, del, eel, elf, eli, elk, elm, ely, fel, île, lae, lea,
 led, leg, lei, len, leo, les, let, lev, lex, ley, lie, lue, tel, vel, yle}
```

```
ConnectedToOrdered[thesaurus_, word_] :=
  Module[{candidates = {}, result = {}, l = Length[Characters[word]], pool},

   pool = DeleteCases[SpecificLength[thesaurus, l], word];

   Do[If[Length[DeleteCases[Characters@word - Characters@w, 0]] == 1,
     AppendTo[result, w]], {w, pool}];
                                        result
                                    ];
```

```
ConnectedToOrdered[thesaurus, "lee"]
```

```
{bee, cee, fee, gee, hee, kee, lae, lea, led, leg, lei, len,
 leo, les, let, lev, lex, ley, lie, lue, nee, see, tee, vee, wee, yee}
```

As the following example shows, this word “see” does not get matched with “lee” in the ConnectedToOrderless, because the intersection deletes duplicates. Not even the union of the results of ConnectedToOrdered and ConnectedToOrderless will catch every word, as the following example shows. Either one finds a way to fix this issue or one defines ones expectations by the results. It’s more interesting to use this data to create a graph showing homotopy relations between all words in a given thesaurus. Personally I like the notion of ConnectedToOrdered better, so that’s what I’ll use from now on.

```
Characters@"see"⋂Characters@"lee"
```

```
{e}
```

```
ConnectedToOrdered[{"ese"}, "lee"]
```

```
{}
```

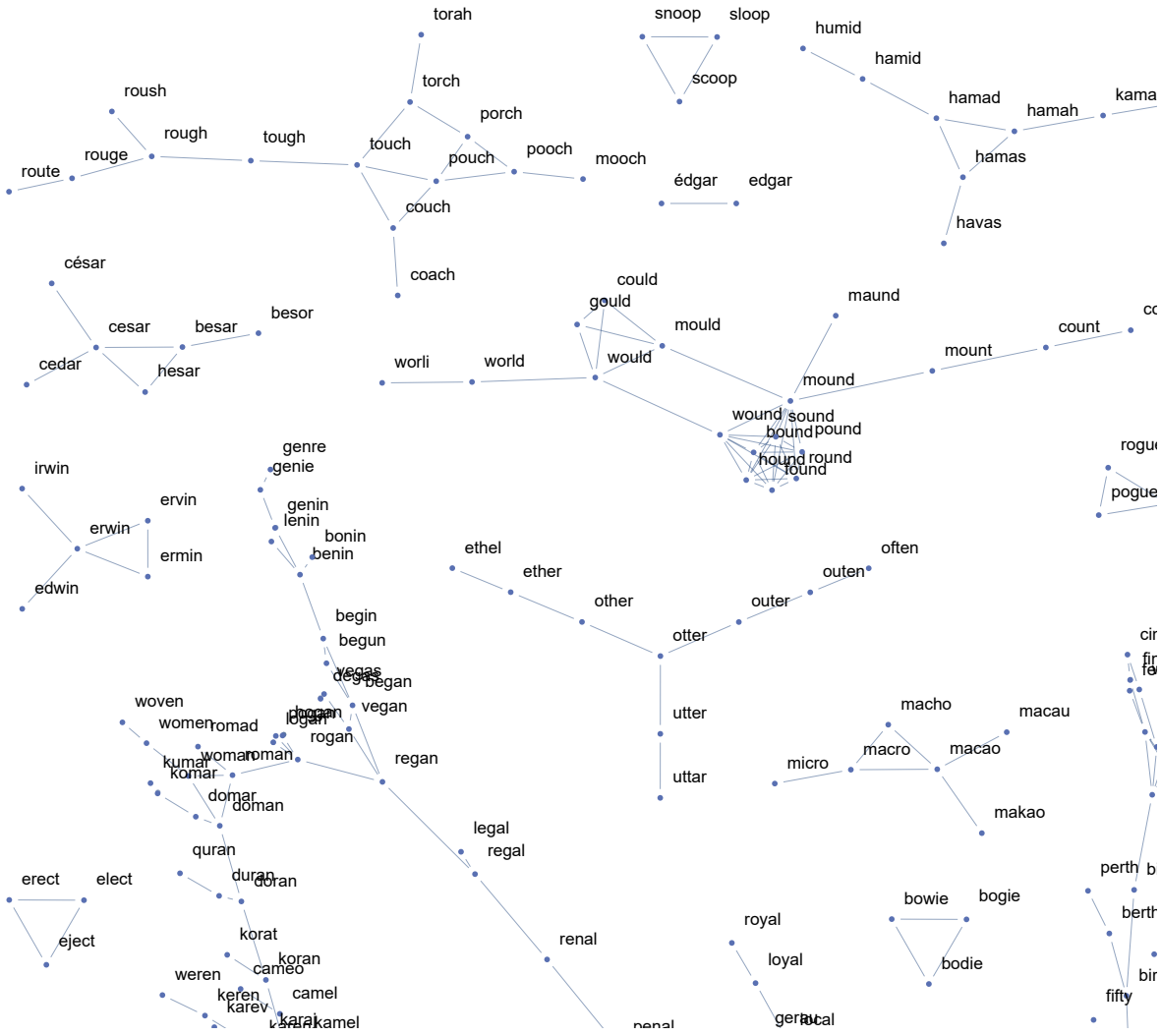
```
ConnectedToOrderless[{"ese"}, "lee"]
```

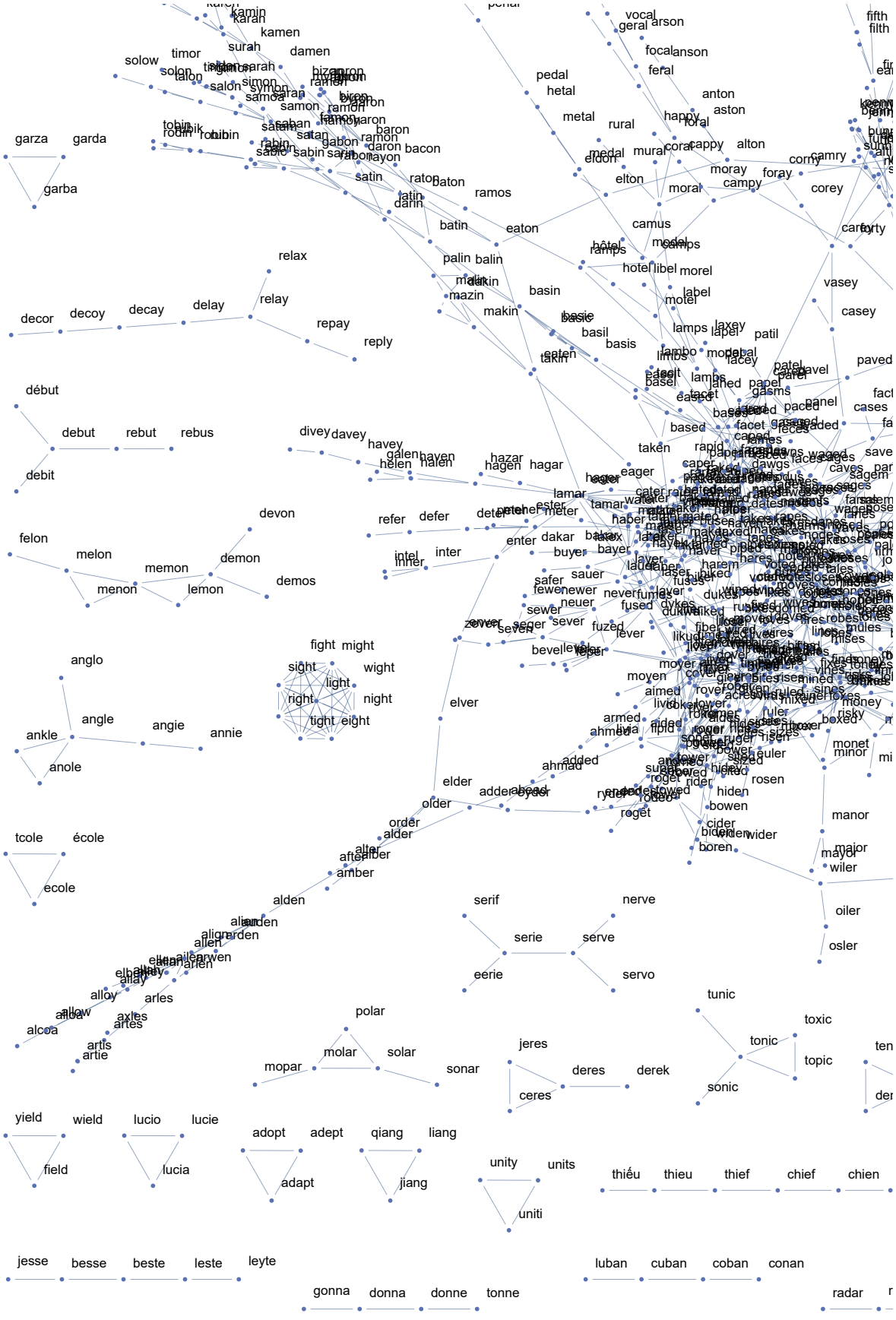
```
{}
```

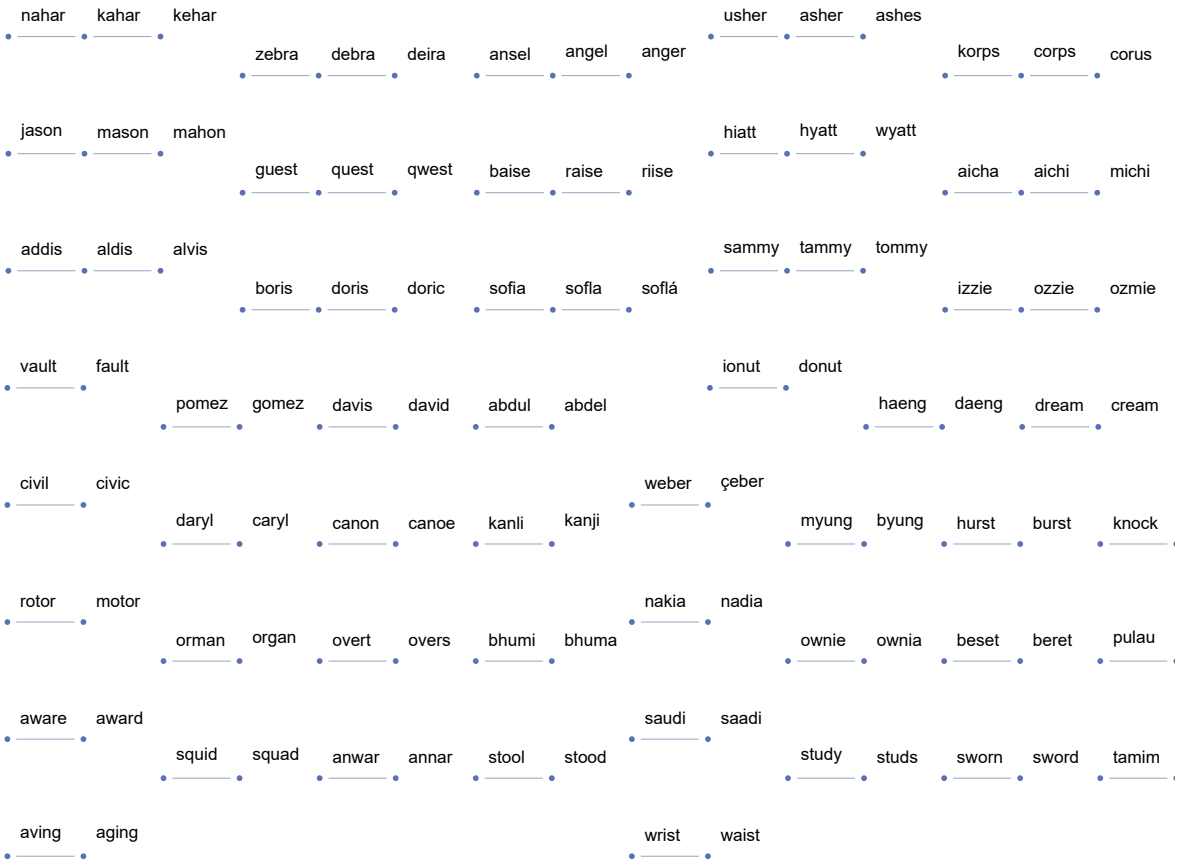
Graph Of Connected Words

This function looks at every words in a given thesaurus and outputs links to plot a graph in which every word shares an edge with all words to which it is *ordered-connected*. Below is an example of the graph of the 6 letter long word only in our thesaurus.

```
HomotopyGraph[thesaurus_] := Module[{links = {}, p, left},
                                    Do[
                                          p = Det[Position[thesaurus, w]];
                                          left = Drop[thesaurus, p];
                                          Do[
                                                AppendTo[links, w ⟷ q];
                                           , {q, ConnectedToOrdered[left, w]}];
                                     , {w, thesaurus}];
                                    links
                                ];

sample = SpecificLength[thesaurus, 5];

l = HomotopyGraph[sample];

g = Graph[l, VertexLabels → "Name",
   GraphLayout → {"PackingLayout" → "ClosestPacking"}]
```




garza garda garba
relax relay delay decay decoy decor repay reply
début debut rebut rebus debit
felon melon memon menon lemon demon devon demos
anglo angle angie annie ankle anole
fight might sight wight light night right tight eight
tcole école ecole
yield wield field
lucio lucie lucia
adopt adept adapt
qiang liang jiang
unity units uniti
jesse besse beste leste leyte
gonna donna donne tonne
serif nerve serie serve eerie servo
polar molar solar mopar sonar
jeres deres derek ceres
tunic toxic tonic topic sonic
thiếu thieu thief chief chien
luban cuban coban conan
radar
alden alder elder older order alter after amber
refer defer enter dakar buyer bayer
divey davey havey galen haven helen halen hagen hagar hazar
eager ester lamar tamar meter
basin basic basil basis
eaton batin latin darin ramos
camus camps ramps hotel libel morel motel label
vocal geral arson focal anson feral
pedal hetal metal rural happy foral anton aston
eldon medal mural coral cappy alton elton moral moray foray campy corey camry corny
vasey casey
taken rapid eager
safer sauer fewer newer never neuer fused sewer sever seger seven zever fuzed fever bevel
moven mover aimed armed livia ahmed added ahmad
rover roger ruled ruler ruger
roget rider hiden bowen cider biden boren wider
rosen monet minor manor major mayor wiler oiler osler



Homotopy Classes

Since *Mathematica* has a built-in function to obtain all connected components of a graph, so it is easy to compute the homotopy classes of a word with it. However easy does not necessarily mean that it is the most efficent way. Inf fact the second of the following two function the above computed graph as an input, because it would take too long to compute the *homotopy graph* of the entire thesaurus, even though it is *only* 17541 words long.

```
HomotopyClassCheating[thesaurus_, word_] := Module[{g, comp, class},

  g = HomotopyGraph[SpecificLength[thesaurus, Length[Characters[word]]]];
                                                  comp = ConnectedComponents[g];

Do[If[ContainsAll[c, {word}], class = c;], {c, comp}];
                                                    class
                                                ];
```

```
HomotopyClassCheating[thesaurus, "finger"]
```

```
{finger, singer}
```

```
HomotopyClassCheating[thesaurus_, word_, g_] := Module[{comp, class},
                                                  comp = ConnectedComponents[g];

   Do[If[ContainsAll[c, {word}], class = c;], {c, comp}];
                                                  class
                                               ];
```

```
HomotopyClassCheating[thesaurus, "finger", g]
```

```
class$75749
```

Since cheating is bad, here is another version of to compute the homotopy class of a word. The algorithm looks iteratively for new words connected to the known words until it can't find any new words anymore.

```
HomotopyClass[thesaurus_, word_] :=
  Module[{class = {}, length = 0, cur = {}, new = {}},
                                        AppendTo[class, word];
                                        AppendTo[cur, word];
                                        While[Length[class] ≠ length,

    Do[new = new ~Join~ ConnectedToOrdered[thesaurus, e], {e, cur}];
                                             new = DeleteDuplicates[new];

    class = DeleteDuplicates[new ~Join~ class];
                                             cur = new;
                                             new = {};
                                             length = Length[class];
                                         ];
                                        class
                                    ];

HomotopicQ[thesaurus_, word1_, word2_] :=
  ContainsAll[HomotopyClass[thesaurus, word1], {word2}];
```

```
HomotopyClass[thesaurus, "nights"]
```

```
{fights, lights, rights, sights, nights}
```

```
HomotopicQ[thesaurus, "martin", "marvin"]
```

```
True
```

It turns out that the non-cheating way of implementing is way faster and more efficient than the variant of this function that uses the graph. Still, having the graph makes it easier to visualize the homotopy classes and also to test the other functions.

The possibilities to continue this project are endless. A slightly weaker definition of two words being homotopic could for example allow to add and delete one character of the word at a time. A much stronger definition could for example involve that the word type, such as noun, verb, adjective, adverb, preposition, etc..., are homotopy invariants. Meaning two words are homotopy related if and only if they are homotopy related by the classical definition and their word type is the same.

Two last ideas that I have and might implement later are:

1) How can I find the longest “path” of words? Meaning, what is the longest sequences of “connected”, nonrepeating words in a given thesaurus. I suspect strongly that this is a very common question in graph theory, so I’ll can probablt just steal an algorithm instead of inventing one myself.
2) Since it takes quite some time to build up a large thesaurus, how can I back it up and restore it in a file? This would also make it possible to run this code on a remote controlled computer, the math server say, and come back after a week and see what it computed.