# Java Search Project: Packing Words in Bins

Ilkka Kokkarinen, Chang School of Continuing Education, Ryerson University, Canada
Document version of **June 7, 2021**

## The Problem To Solve

To serve as a standard dataset first for *Stanford Graphbase* and then for the ever-sprawling *Volume 4* about combinatorial algorithms, Donald Knuth compiled and set in stone a collection of 5757 five-letter English words. These words are available in the file `sgb-words.txt`, sorted by frequency. The density of the five-letter words in English along with their connections to each other produces **word graphs** of just right the right amount of complexity to allow for interesting problems. Four-letter words are too dense and six- and higher-letter words are too sparse for most combinatorial search and optimization problems to be interesting for graphs of such words..

In this spirit of exploration, students may use any and all combinatorial search and optimization techniques in their arsenal to solve the following fun little combinatorial problem. Define two words to be **mutually incompatible** if they contain the same letter in any position. For example, the words `frump` and `gouts` are incompatible, since they both have the same letter `u` in the middle position, even when these words otherwise have no letters in common. On the other hand, even though the two anagrams `angle` and `glean` consist of the exact same five letters, they are still **mutually compatible**, since none of these characters is in the same position in both words.

The problem itself can now be stated while standing on one foot: Place the given words into the smallest possible number "bins" so that **all words placed into the same bin must be mutually compatible with each other**. For example, consider these randomly drawn sixty words:

```
avoid, spasm, these, prick, shunt, endow, degum, hapax, umbra, jacks,
russe, areal, modes, cinch, regal, bleak, spell, rille, glyph, jiffy,
abaft, tangy, coops, scant, cloth, hying, screw, props, grown, bravo,
nicer, facie, weeny, lords, exude, buffa, kudzu, molar, redox, elude,
lacer, beaut, skiff, adman, naked, unhip, prior, etext, gibes, rumor
```

These words can fit into six separate bins while maintaining the compatibility of all words inside the same bin. One out of exponentially many possible ways to do so would be:

```
Bin 1: [avoid, these, prick, hapax, umbra, modes, glyph, jiffy, scant,
rumor]
Bin 2: [jacks, screw, bravo, weeny, exude, kudzu, adman, unhip]
Bin 3: [shunt, endow, degum, russe, bleak, tangy, grown, nicer, lords]
Bin 4: [spasm, regal, cloth, facie, buffa, prior, etext, gibes]
Bin 5: [areal, rille, coops, lacer, beaut, skiff]
Bin 6: [cinch, spell, abaft, hying, props, molar, redox, elude, naked]
```

However, six bins turn out to be as good as it gets. Whichever way you to try twist, turn, shout and pout, these words just can't go into only five bins without at least one of these bins ending up with two mutually incompatible words.

(An equivalent way to phrase this problem using the terminology of combinatorial graph theory is to ask for a **minimum vertex colouring** in the word graph with words as nodes, and two words are connected by an edge if and only if their **Hamming distance** is less than five.)

## Automated Tester

As usual with the courses given by this instructor, an automated tester `WordPackingTest.java` will be used to grade this project. When used on the command line, it has the format

```
java WordPackingTest SEED SIZE ROUNDS VERBOSE
```

where `SEED` is the seed for the tester's pseudorandom number generator, `SIZE` is the number of words in each problem instance, `ROUNDS` is the number of rounds to play this game, and `VERBOSE` determines whether the tester prints out everything or only the final score line. An example run that consists of three rounds of lists that each contain 50 words should resemble the following.

```
matryximac:Java 305 ilkkakokkarinen$ java WordPackingTest 777777 50 3 true
Read 5757 words from <sgb-words.txt>.

Seed 777777: [nutsy, deter, dicot, gulch, adman, junta, skint, phone, which, marks,
ethyl, owned, mylar, unfit, scrub, skunk, myths, slots, cedar, lisle, undue, hinds,
estop, dream, jibed, impel, patch, sibyl, quark, combo, probe, ennui, polka, gamin,
daddy, atone, views, torah, asked, shush, whirr, cools, hoard, whang, ftped, harpy,
ocean, rutty, found, runty]
Solution found in 81 ms.
Bin 1: [deter, adman, junta, which, marks, sibyl, probe, found]
Bin 2: [skint, mylar, undue, impel, patch, views, hoard, runty]
Bin 3: [dicot, owned, myths, cedar, quark, polka, atone, shush, harpy]
Bin 4: [phone, ethyl, scrub, hinds, dream, combo, asked, rutty]
Bin 5: [nutsy, slots, lisle, ennui, gamin, torah, whang, ftped]
Bin 6: [gulch, unfit, skunk, estop, jibed, daddy, whirr, cools, ocean]

Seed 777778: [hexed, recap, carry, rodeo, emote, litho, pager, trick, crown, faint,
clink, typos, chino, waged, trawl, bowls, jihad, petit, kelly, axman, sloop, dogma,
elate, sutra, vitas, dials, raspy, gypsy, demon, unapt, lucky, cruse, snark, wurst,
ghoul, sweep, tenth, noddy, ghoti, mound, bride, upped, range, minas, nymph, swede,
forth, wrong, sigma, flits]
Solution found in 2966 ms.
Bin 1: [recap, carry, typos, unapt, ghoti, mound, bride, sigma]
Bin 2: [emote, pager, clink, trawl, bowls, jihad, demon, wurst]
Bin 3: [rodeo, crown, faint, gypsy, tenth, swede]
Bin 4: [waged, petit, axman, sloop, dials, lucky, cruse, forth]
Bin 5: [chino, elate, sutra, raspy, upped, minas]
Bin 6: [hexed, litho, snark, noddy, range, wrong, flits]
Bin 7: [trick, kelly, dogma, vitas, ghoul, sweep, nymph]

Seed 777779: [stash, decaf, onion, rehab, goest, armor, spark, ixnay, jives, hippy,
algin, rived, dunno, radix, swoon, noose, arise, hadda, bream, crack, klieg, agony,
furls, bombe, admit, hullo, close, rapid, bones, seams, igloo, rawly, cause, whelp,
loads, hymns, dicot, fiche, avail, chord, spoil, cruft, yerba, fauna, stung, clunk,
quick, token, umber, hertz]
Solution found in 269 ms.
Bin 1: [armor, spark, hullo, close, rapid, bones]
Bin 2: [rehab, algin, noose, hadda, crack, whelp, dicot, stung]
Bin 3: [decaf, hippy, arise, furls, igloo, spoil, clunk, token]
Bin 4: [rived, bream, agony, seams, cause, quick]
Bin 5: [ixnay, admit, loads, chord, fauna, umber]
Bin 6: [goest, swoon, klieg, rawly, hymns, fiche, avail, yerba]
Bin 7: [stash, onion, jives, dunno, radix, bombe, cruft, hertz]

20 3316 123456789 Kokkarinen, Ilkka
```

The last line of the tester output shows the total score (lower is better) followed by the total measured running time of the entire run, given in milliseconds (lower is better), followed by your student information. If the command line option `VERBOSE` is set to `false`, the tester prints only this last line. However, during your development, you will surely want to examine these verbose outputs. (Of course you may also print your own debugging outputs during development, just remember to comment them all out before submission.)

To make this problem more fun, note that the construction of the random wordlist is done with the "thumb on the scale" to avoid words that contain letters in positions that have already been used too many times in the previously selected words. In general, problems whose constraints are so hard that they essentially force you directly to the only possible solution are usually not that interesting. Same as with other NP-complete combinatorial search problems, there is a "sweet spot" for the number of random constraints that allows its chaos of latent possibilities to burst out and genuinely flower.

## Grading

Seeing that the generalization of this problem for arbitrary sets of strings is **NP-complete**, the student programs are not required to always find the smallest possible number of bins. However, the fewer the bins needed, the better the overall score. However, **to qualify for any project marks at all, submitted programs must work correctly** in the sense that for every individual test case, the method returns a legal solution that places every word into exactly one bin so that no bin contains two mutually incompatible words.

Student submissions will be tested by the instructor using the exact same secret `SEED`, with the `SIZE` of 50 for a total of ten `ROUNDS`.

Furthermore, this test run of the student solution has a total time limit of **two minutes for these ten rounds**. **Code whose execution does not terminate by that time will also receive a zero**

**mark.** Before submitting, make sure to run your code overnight for at least a thousand rounds. If no crash or error greets you when you wake up the next morning, you should be gold.

As always with this instructor's projects, **silence is golden**. Your methods may not print anything during their execution. **A project submission that prints even one character on its own will automatically receive a zero mark.** (Only little children believe that programs must print something to actually do anything.)

Any attempt to interfere with the discipline and the measured results of the automated tester in any way, even as a prank or a proof of concept, will also forfeit all project marks.

## Required Methods

Your submission must consist of **exactly one Java 8 source code file** named `WordPacking.java` and no other files. However, you may define nested helper classes inside this class. You may use any and all data structures and algorithms provided in the Java 8 standard library. This way you don't need to reinvent any of those rusty old wheels, but can concentrate full time on the logic of your search and optimization.

Your `WordPacking` class **must have the following exact methods** so that the tester can compile and run:

```
public static String getAuthorName()
```

Returns your name in the format "`Lastname, Firstname`".

```
public static String getStudentID()
```

Returns your student ID. Please do not include any whitespace characters in your ID.

```
public static List<List<String>> wordPack(List<String> words)
```

This method is the heart and soul of this project. Given the list of `words`, create and return a `List` of bins so that each bin is a list of words placed in that bin. The tester enforces that every word in `words` appears in exactly one bin, and that no two words in the same bin contain the same letter in the same position.

You can freely choose the exact subtype of the returned list object and the lists inside it, as long as they are subtypes of `List<String>`. Most likely, simple `ArrayList<>` is best.

There are several general approaches that could be taken here. For example, you could start by placing each word into a separate bin on its own, and then keep combining bins whose contents are mutually compatible. Alternatively, you could start by putting everything in one bin, and then taking words out and placing them to other bins until all the remaining words in the bin are compatible. Or something else entirely. However, all of these techniques will lead to hard choices whose eventual consequences are not immediately obvious.

## Historical Side Contest: Maximize One Bin

This instructor first used this problem during his stint of daytime teaching back in Fall 2018. While preparing this problem, he also realized that this setup allows for an interesting sideshow tent **one-shot problem**; If allowed to freely choose the words you use from the file `sgb-words.txt`, how many words can you fit into a single bin?

Students were offered this voluntary side quest with the reward of the purely ceremonial title of "The Soldering Iron of Justice" for the duration of that term for the student or a pair of students to first submit the largest subset of words that fit together into a single bin. Theoretically this subset could contain up to 26 words, since every word consumes exactly one of the 26 possible letters from each of its five positions. However, same as with most other real world datasets, Knuth's wordlist rarely contains the items that we hope it to have. In practice we fall short of this ideal for the lack of a nail that inevitably leads to the lack of a horseshoe, and so on. Weird global stuff can emerge from complex interactions of local constraints.

The following table shows the gradual improvement of the known best solution, listed in reverse chronological order to make the best solution to be the cherry on the top.

| **Date** | **Student(s)** | **Size** | **Solution** |
|---|---|---|---|
| Nov 23 | Ryan Chan Shum Hang | 18 | `[ethyl, oxbow, injun, affix, nymph, kudzu, uvula, thwap, pssst, gizmo, skiff, flyby, wrong, bocci, hertz, dweeb, valve, czars]` |
| Nov 7 | Tyler Blanchard | 17 | `[pffft, ethyl, bally, vents, whose, litho, dweeb, using, abuzz, mujik, cramp, zombi, scram, incur, glyph, oxbow, hydra]` |
| Nov 5 | Tyler Blanchard | 16 | `[lambs, flank, trice, cowed, input, bingo, nervy, excon, abuzz, umbra, sylph, dwell, pshaw, optic, whomp, mufti]` |
| Oct 29 | Deep Patel, Calvin Yap | 15 | `[after, backs, cease, diddy, eclat, fjord, glitz, hokum, idyll, jumbo, lynch, omega, rhumb, sprig, unbox]` |

As of this writing, the lack of fully exhaustive search has not ruled out better solutions. Any student who comes up with a set of 19 mutually compatible words will instantly become the permanent holder of the purely ceremonial title of "The Soldering Iron of Heaven's Justice", at least as far as this instructor is concerned.

Motivated students can also study the same challenge in other languages. For example, how large a bin could somebody fill to the brim using the five-letter words of German, Spanish or French?