5. Strings, Recursion and SDL

5.1 Anagrams

An anagram is a collection of letters that when unscrambled, using all the leters, make a single word. For instance `magrana` can be rearranged to make the word `anagram`.

Exercise 5.1 Using a file of valid words, allow the user to enter an anagram, and have the answer(s) printed. For instance :

```
 % ./anagram sternaig
angriest
astringe
ganister
gantries
ingrates
rangiest
reasting
stearing
```

■

Exercise 5.2 Using a file of valid words, find all words which are anagrams of each other. Each word should appear in a maximum of one list. Output will look something like :

```
 % ./selfanagram
.
.
7 merits mister miters mitres remits smiter timers
.
.
.
6 noters stoner tenors tensor toners trones
.
.
.
6 opts post pots spot stop tops
```

```
.
.
.
6 restrain retrains strainer terrains trainers transire
.
```

■

If you wished to create "interesting" anagrams, rather than simply a random jumble of letters, you could combine together two shorter words which are an anagram of a longer one.

Exercise 5.3 Write a program which uses an exhaustive search of all the possible pairs of short words to make the target word to be computed. For instance, a few of the many pairs that can be used to make an anagram of `compiler` are :

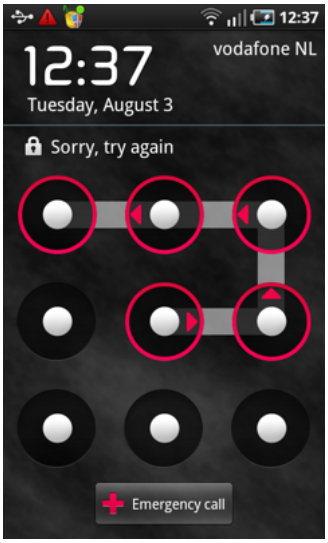
```
 % ./teabreak compiler
LiceRomp
LimeCrop
LimpCore
MileCrop
MoreClip
PermCoil
PromLice
RelicMop
```

■

The name `Campbell` comes out as `CalmPleb` which is a bit harsh. Can't **ever** remember being called calm ...

5.2 Draw to Unlock

Rather than remembering passwords or passcodes, many mobile devices now allow the user to draw a pattern on the screen to unlock them.



Here we will explore how many unique patterns are available when drawing such patterns to connect "dots", such as shown in the figure. We assume that people put their finger on one "dot" and then only ever move one position left, right, up or down (but never diagonally) at a time.

You are not allowed to return to a "dot" once it has been visited once. If we number the first position in our path as 1, the second as 2 and so on, then beginning in the top left-hand corner, some of the possible patterns of 9 moves are :

```
1 2 3      1 2 3      1 2 3
6 5 4      8 9 4      8 7 4
7 8 9      7 6 5      9 6 5
```

Exercise 5.4 Write a program that computes and outputs all the valid paths. Use **recursion** to achieve this.

- How many different patterns of length 9 are available on a 3×3 grid, if the user begins in the top left corner ?
- How many different patterns of length 9 are available on a 3×3 grid, if the user begins in the middle left ?
- How many different patterns of length 7 are available on a 3×3 grid, if the user begins in the top left corner ?
- How many different patterns of length 25 are available on a 5×5 grid, if the user begins in the top left corner ?

■

5.3 SDL - Intro

Many programming languages have no inherent graphics capabilities. To get windows to appear on the screen, or to draw lines and shapes, you need to make use of an external library. Here we use SDL[1], a cross-platform library providing the user with (amongst other things) such graphical capabilities.

www `https://www.libsdl.org/`

The use of SDL is, unsurprisingly, non-trival, so some simple wrapper files have been created (`neillsdl2.c` and `neillsdl2.h`). These give you some simple functions to initialise a window, draw rectangles, wait for the user to press a key etc.

An example program using this functionality is provided in a file `blocks.c`.

This program initialises a window, then sits in a loop, drawing randomly positioned and coloured squares, until the user presses the mouse or a key.

Exercise 5.5 Using the `Makefile` provided, compile and run this program, using something like : `make -f sdl_makefile`

SDL is already installed on lab machines. At home, if you're using a ubuntu-style linux machine, use: `sudo apt install libsdl2-dev` to install it. ■

5.4 Word Ladders

In this game, made famous by the author Lewis Caroll, and investigated by many Computer Scientists including Donald Knuth, you find missing words to complete a sequence. For instance, you might be asked how go from "WILD" to "TAME" by only changing one character at a time:

[1] actually, we are using the most recent version SDL2, which is installed on all the lab machines

W I L D
W I L E
T I L E
T A L E
T A M E

A useful concept here is that of the `edit distance`. Here, the edit distance is a count of the number of characters which are different between two words. For words of n characters, the edit distance will be in the range $0 \ldots n$. For 'aboard' and 'canape' the edit distance is 5. An edit distance of zero means that the words are identical.

In a heavily constrained version of this game we make some simplifying assumptions:

- Words are always four letters long.
- We only seek ladders of five words in total.
- Only one letter is changed at a time.
- A letter is only changed from its initial state, to its target state. This is important, since if you decide to change the second letter then you will always know what it's changing from, to what it's changing to.

So, in the example above, it is enough to give the first word, the last word, and the position of the character which changed on each line. On line one, the fourth letter 'D' was changed to an 'E', on the next line the first character 'W' was changed to a 'T' and so on. The whole ladder can be defined by "WILD", "TAME" and the sequence $4, 1, 2, 3$.

W I L (D)
(W) I L E
T (I) L E
T A (L) E
T A M E

Since each letter changes exactly once, the order in which this happens is a *permutation* of the numbers $1, 2, 3, 4$, which we have looked at elsewhere..

We'll also need another function:

```
int edit_distance(char *s, char *t);
```

which returns the number of characters which are different between two strings of the same length. For our strings of length four (excluding the null character) this will be either $0, 1, 2, 3$ or 4. Here, we can check that the first and last word in our search are distance 4 apart.

Exercise 5.6 For the constrained version of the game, given a file of valid four letter words, write a program which when given two words on the command line (`argv[1]` and `argv[2]`) outputs the correct solution, if available. Use an exhaustive search over all 24 permutations until one leads to no invalid words being required. Make sure your program works, with amongst others, the following:

C O (L) D
C (O) R D
(C) A R D
W A R (D)
W A R M

P O (K) E
P O L (E)
(P) O L L
M (O) L L
M A L L

C U B (E)
(C) U B S
T U (B) S
T (U) N S
T O N S

■

Exercise 5.7 Adapt the program above so that if the first and last words share a letter (the edit distance is less than 4), you can find the word ladder required, as in:

W A S P
W A S H
W I S H
F I S H

■

For the "full" version of Wordladder, you make no assumptions about the number of words that are needed to make the ladder, although we do assume that all the words in the ladder are the same size.

Exercise 5.8 To achieve this, you could make a list of all the words, and for all words an `edit distance` of 1 away from the initial word, mark these and store their 'parent'. Now, go through this list, and for all words marked, find words which are distance 1 from these, and hence distance 2 from the initial word. Mark these and retain their parent. Be careful you don't use words already marked. If the word ladder is possible, you'll eventually find the solution, and via the record of the parents, have the correct route. This is shown for the word ladder CAT to DOG in Figure 5.1 using a very small subset of the possible three letter words.
■

5.5 The Devil's Dartboard

In the traditional 'pub' game, darts, there are 62 different possible scores : single 1 - 20 (the white and black areas), double 1 - 20 (the outer red and green segments) (i.e. 2, 4, 6, 8 ...), treble 1 - 20 (i.e. 3, 6, 9, 12 ...) (the inner red or green segments), 25 (small green circle) and 50 (the small red inner circle).

It's not obvious, if you were inventing darts from scratch, how best to lay out the numbers. The London board shown seems to have small numbers near high numbers, so that if you just miss the 20 for example, you'll hit a small number instead.

Here we look at a measure for the 'difficulty' of a dartboard. One approach is to simply sum up the values of adjacent triples, squaring this number. So for the London board shown, this would be: $(20+1+18)^2+(1+18+4)^2+(18+4+13)^2 \ldots (5+20+1)^2 = 20478$

For our purposes a **lower** number is better[2]. For more details see :

www `http://www.mathpages.com/home/kmath025.htm`

Exercise 5.9 Write a program that repeatedly chooses two positions on the board and swaps their numbers. If this leads to a lower cost, keep the board. If not, unswap them. Repeat this *greedy search* 5000000 times, and print out the best board found. Begin with the trivial monotonic sequence. The output may look something like :

```
Total = 19966 :  3 19 11  2 18 12  1 20 10  4 16  8
```

[2]It's beyond the scope here to explain why!

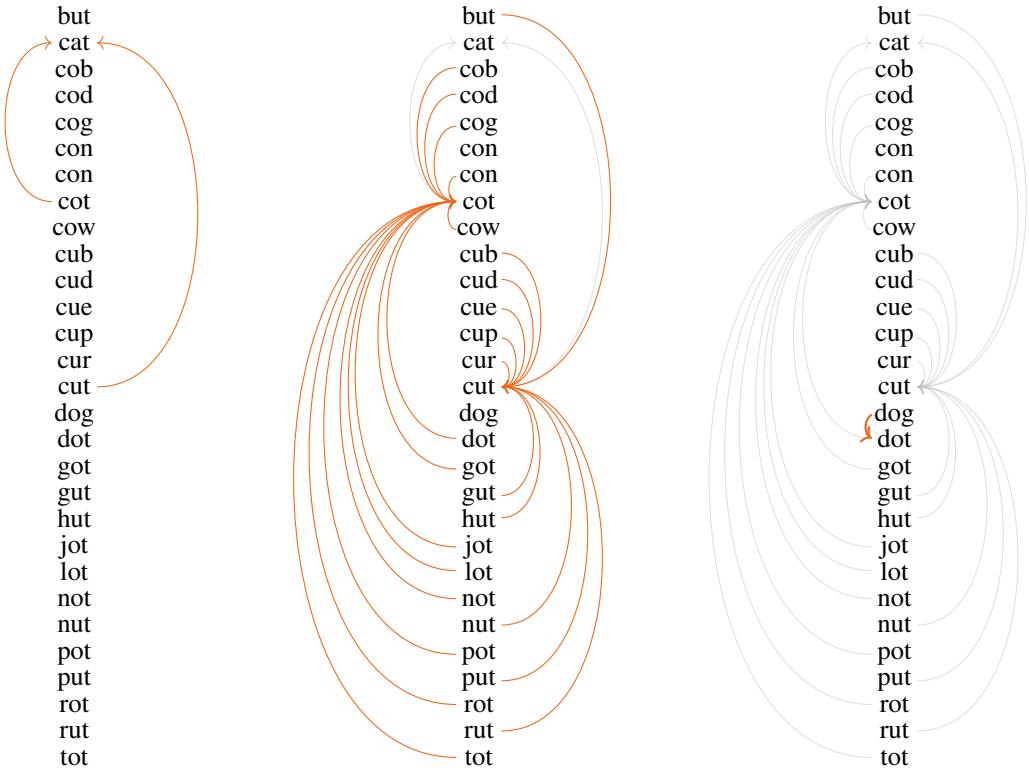
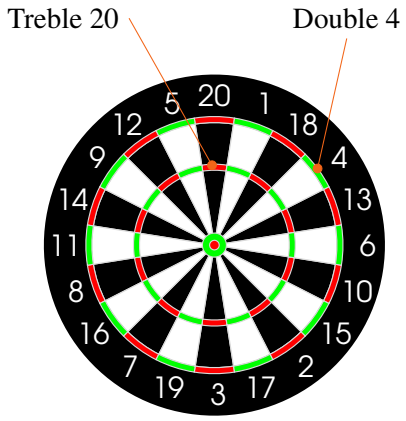


Figure 5.1: Word Ladder from CAT to DOG. (Left) Words which are distance one from CAT are COT and CUT. (Middle) Words which are distance one from CUT and COT, which includes amongst others, DOT. (Right) DOG is distance one from DOT. We now have our route, via the pointers, back to CAT.




```
  14   5  13  15   6   7  17   9
```

or

```
  Total = 19910 :   3  18  10   5  16   9   8  14  11   4  19   6
  7  20   2  13  15   1  17  12
```

The score of 19874 seems to be the lowest possible that may be obtained via this technique.

■

5.6 Maze

Escaping from a maze can be done in several ways (ink-blotting, righthand-on-wall etc.) but here we look at recursion.

Exercise 5.10 Write a program to read in a maze typed by a user via the filename passed to `argv[1]`. You can assume the maze will be no larger than 20×20, walls are designated by with a # and the rest are spaces. The entrance can be assumed to be the gap in the wall closest to (but not necessarilty exactly at) the top lefthand corner. The sizes of the maze are given on the first line of the file (width,height). Write a program that finds the route through a maze, read from this file, and prints out the solution (if one exists) using full stops. If the program succeeds it should exit with a status of `0`, or if no route exists it should exit with a status of `1`.

■