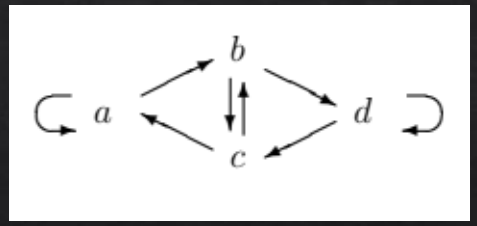
Robby Silvey

Tile Fitting Problem

The Problem

- Recall the graph which encodes the information about how our sequences of tiles can be fitted snuggly on top of each other (after aligning them to fit of course).



- In particular, the “a” sequence fits on top of itself to form “aa”. This can be continued, forming the pattern “aaa” of three sequences fitting together snuggly:

a	1	2	8	6	4	3	5	7
a	4	3	5	7	1	2	8	6
a	1	2	8	6	4	3	5	7

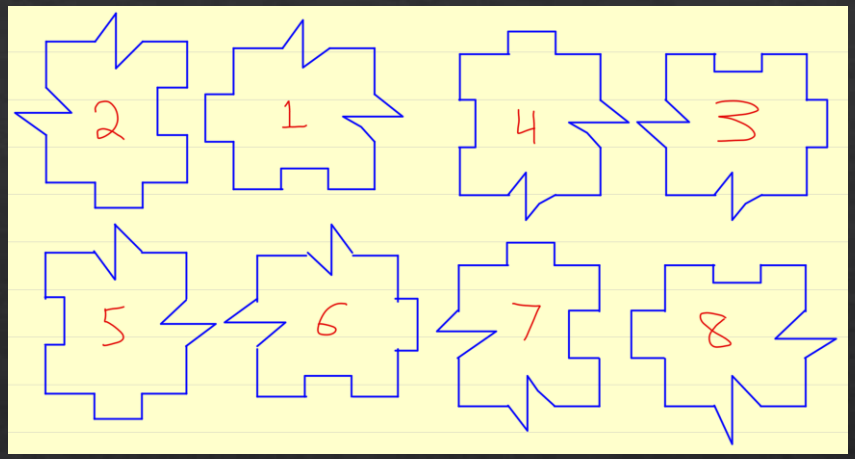
The Problem

- Find a 3-term pattern in which all the columns contain distinct orientations of the tile.
- I.E. none of the numbers repeat in any given column.

- Extend this to find an 8-term pattern.

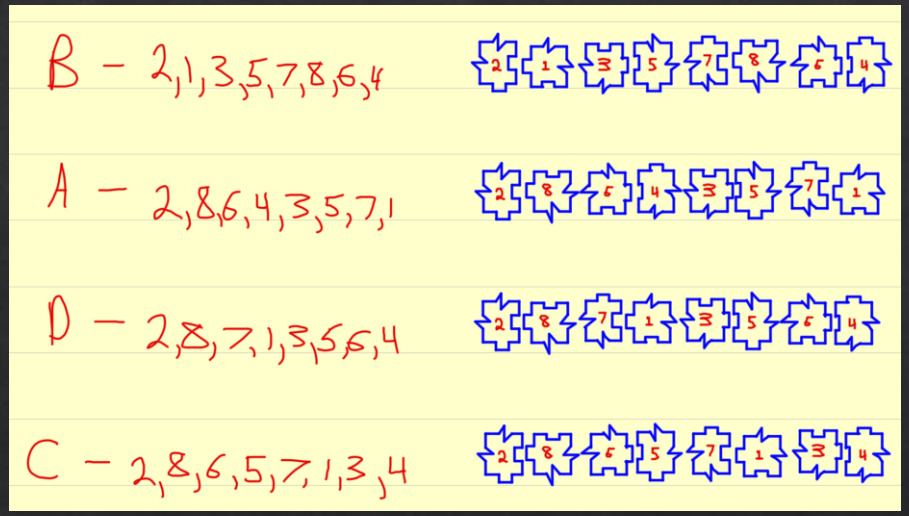
Refresher

- We started with one tile and created these 8.



Refresher

- From these tiles we have 4 sequences.



The Solution

- To solve this problem I decided to write a program in C++ to brute force the answer.
- https://github.com/rsilvey2/Tile-Fitting-Problem

The Solution

- Class of tiles with information being the name of the sequence and the sequence itself.

```
10    class tiles {
11
12    public:
13
14        tiles(std::string x, std::vector<int> y);
15        ~tiles();
16
17        std::vector<int> numbers;
18        std::string name;
```

```
int main() {

    tiles A("A",{2,8,6,4,3,5,7,1});
    tiles B("B",{2,1,3,5,7,8,6,4});
    tiles C("C",{2,8,6,5,7,1,3,4});
    tiles D("D",{2,8,7,1,3,5,6,4});

    std::vector<tiles> input = { A,B,C,D };
    multifullprintscan(compareanswers(computeanswer(input)));


    return 0;
}
```

The Solution

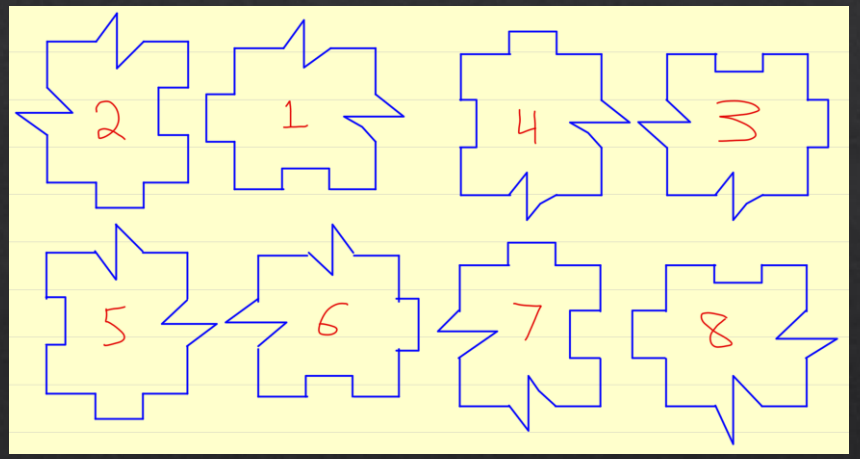
- multifullprintscan
- compareanswers
- computeanswer
- scan
- stackSequences
- createshifted
- shift
- doesfit

```
multifullprintscan(compareanswers(computeanswer(scan
(stackSequences(createshifted(shift(doesfit(((((((((;
```

doesfit

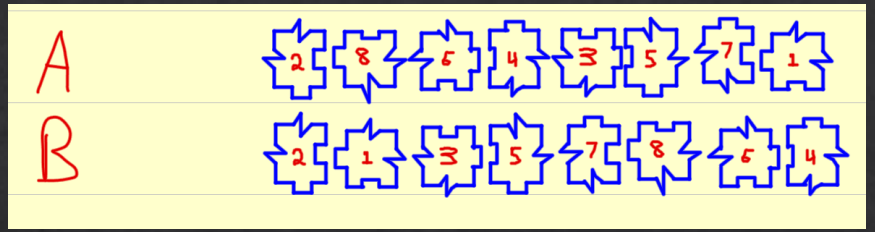
- Checks if tile a fits on top of tile b.

```
21  bool doesfit(int a, int b) {
22      if ((a == 1) && ((b == 4) || (b == 7))) {
23          return true;
24      }
25      if ((a == 2) && ((b == 3) || (b == 8))) {
26          return true;
27      }
28      if ((a == 3) && ((b == 1) || (b == 2))) {
29          return true;
30      }
31      if ((a == 4) && ((b == 1) || (b == 2))) {
32          return true;
33      }
34      if ((a == 5) && ((b == 3) || (b == 8))) {
35          return true;
36      }
37      if ((a == 6) && ((b == 4) || (b == 7))) {
38          return true;
39      }
40      if ((a == 7) && ((b == 5) || (b == 6))) {
41          return true;
42      }
43      if ((a == 8) && ((b == 5) || (b == 6))) {
44          return true;
45      }
46      return false;
47  }
```

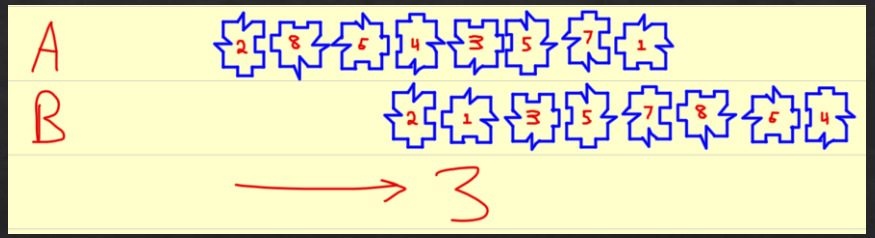


shift

- Takes two sequences of tiles and tries to shift them to fit.



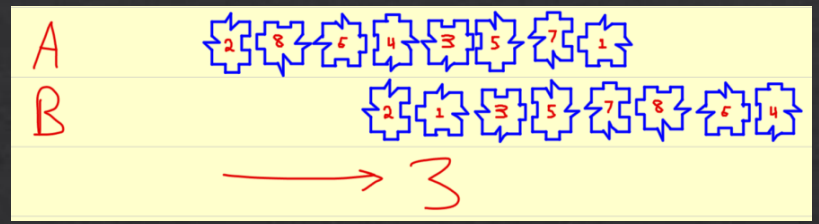

```
49  int shift(tiles a, tiles b) {
50      int shift = 0;
51      int fits;
52      while (shift < 8) {
53          fits = 0;
54          for (int i = 0; i < 8; i++) {
55              if (doesfit(a.numbers[i], b.numbers[(i + shift)%8])) {
56                  fits++;
57              }
58          }
59          if (fits == 8) {
60              return shift;
61          }
62          else {
63              shift++;
64          }
65      }
66      return -1;
67  }
```

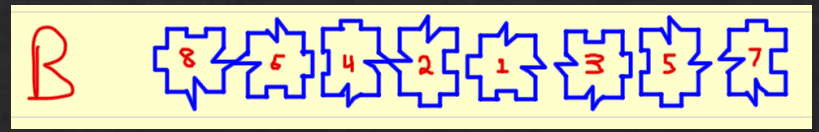


createshifted

- Creates the shifted sequence.
- Puts error in the name if no fit.

```
69  tiles createshifted(tiles a, int shift) {
70      if (shift == -1) {
71          shift = 0;
72          a.name = a.name + " ERROR";
73      }
74      tiles shifted(a.name, {0,0,0,0,0,0,0,0});
75      for (int i = 0; i < 8; i++) {
76          shifted.numbers[i] = a.numbers[(i + shift) % 8];
77      }
78      return shifted;
79  }
```

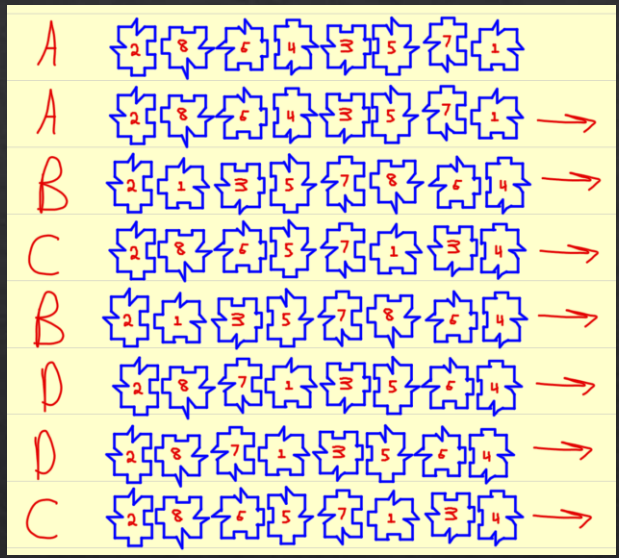




stackSequences

- Takes an order of sequences of tiles (A,A,B,C,B,D,D,C).
- Attempts to shift each sequence to fit.

```
107    std::vector<tiles> stackSequences(std::vector<tiles> tileorder) {
108        tiles Z("Z", {0,0,0,0,0,0,0,0});
109        std::vector<tiles> newtiles = {Z,Z,Z,Z,Z,Z,Z,Z};
110        newtiles[0] = tileorder[0];
111        for (int i = 1; i < 8; i++) {
112            newtiles[i] = createshifted(tileorder[i], shift(newtiles[i - 1], tileorder[i]));
113        }
114        return newtiles;
115    }
```



scan

- Scans the columns for matches.
- Counts the errors.

```
A - 28643571
A - 35712864
B - 13578642
C - 42865713
B - 13578642
D - 71356428
D - 64287135
C - 42865713
------------
E - 22222222
------------
16 Total Column Matches
-----------------------
```

```
89  std::tuple<tiles, int, std::vector<tiles>> scan(std::vector<tiles> newtiles) {
90      tiles E("E", {0,0,0,0,0,0,0,0});
91      int matches = 0;
92      for (int i = 0; i < 8; i++) {
93          std::vector<int> matched(1);
94          for (int j = 0; j < 8; j++) {
95              for (int k = j+1; k < 8; k++) {
96                  if ((newtiles[j].numbers[i] == newtiles[k].numbers[i]) && !(std::find(matched.begin(),matched.end(),k) != matched.end())){
97                      E.numbers[i] = E.numbers[i] + 1;
98                      matches++;
99                      matched.push_back(k);
100                 }
101             }
102         }
103     }
104     return {E, matches, newtiles};
105 }
```

computeanswer

- Creates all possible sequences of sequences of tiles.
- Saves the ones with 0 column errors.
- 4^8 = 65,536

```
157 std::vector<std::vector<tiles>> computeanswer(std::vector<tiles> input) {
158     std::vector<std::vector<tiles>> answers;
159     for (int i = 0; i < 4; i++) {
160         for (int j = 0; j < 4; j++) {
161             for (int k = 0; k < 4; k++) {
162                 for (int l = 0; l < 4; l++) {
163                     for (int m = 0; m < 4; m++) {
164                         for (int n = 0; n < 4; n++) {
165                             for (int o = 0; o < 4; o++) {
166                                 for (int p = 0; p < 4; p++) {
167                                     std::vector<tiles> tileorder = { input[i],input[j],input[k],input[l],input[m],input[n],input[o],input[p] };
168                                     std::tuple<tiles, int, std::vector<tiles>> tuple = scan(stackSequences(tileorder));
169                                     if (std::get<1>(tuple) == 0) {
170                                         answers.push_back(std::get<2>(tuple));
171                                     }
172                                 }
173                             }
174                         }
175                     }
176                 }
177             }
178         }
179     }
180     return answers;
181 }
```

- (A,A,A,A,A,A,A,A)
- (A,A,A,A,A,A,A,B)
- (A,A,A,A,A,A,A,C)
- (A,A,A,A,A,A,A,D)
- (A,A,A,A,A,A,B,A)
- (A,A,A,A,A,A,B,B)
- …
- (D,D,D,D,D,D,D,D)

compareanswers

- Compares all valid answers with each other.

```
A - 28643571
A - 35712864
B - 13578642
D - 71356428
C - 57134286
B - 86421357
D - 64287135
C - 42865713
```

```
B - 21357864
D - 87135642
C - 65713428
B - 78642135
D - 56428713
C - 34286571
A - 12864357
A - 43571286
```

```
183 std::tuple<std::vector<std::vector<tiles>>,std::vector<std::vector<int>>> compareanswers(std::vector<std::vector<tiles>> answers) {
184     int matches;
185     std::vector<std::vector<int>> returnmatches(16);
186     for (int i = 0; i < answers.size(); i++) {
187         for (int j = 0; j < answers.size(); j++) {
188             for (int k = 0; k < 8; k++) {
189                 matches = 0;
190                 for (int l = 0; l < 8; l++) {
191                     if (answers[i][l].name == answers[j][(l+k) % 8].name) {
192                         matches++;
193                     }
194                     if (matches == 8) {
195                         returnmatches[i].push_back(j+1);
196                     }
197                 }
198             }
199         }
200     }
201     return {answers, returnmatches};
202 }
```

multifullprintscan

- Prints the results.

```cpp
void multifullprintscan(std::tuple<std::vector<std::vector<tiles>>, std::vector<std::vector<int>>> answer_matches) {
    std::cout << "======================" << "\n";
    for (int j = 0; j < std::get<0>(answer_matches).size(); j++) {
        std::cout << "Answer " << j + 1 << ": Same as ";
        if (!std::get<1>(answer_matches)[j].empty()) {
            miniprintmatches(std::get<1>(answer_matches)[j]);
        }
        std::cout << "\n";
        std::cout << "------------" << "\n";
        for (int i = 0; i < 8; i++) {
            print(std::get<0>(answer_matches)[j][i]);
        }
        std::tuple<tiles, int, std::vector<tiles>> outs = scan(std::get<0>(answer_matches)[j]);
        std::cout << "------------" << "\n";
        print(std::get<0>(outs));
        std::cout << "------------" << "\n";
        std::cout << std::get<1>(outs) << " Total Column Matches\n";
        std::cout << "======================" << "\n";
    }
}
```

```
======================
Answer 1: Same as 1, 3, 6, 7, 9, 12, 13, 15
------------
A - 28643571
A - 35712864
B - 13578642
D - 71356428
C - 57134286
B - 86421357
D - 64287135
C - 42865713
------------
E - 00000000
------------
0 Total Column Matches
======================
Answer 2: Same as 2, 4, 5, 8, 10, 11, 14, 16
------------
A - 28643571
B - 86421357
C - 57134286
A - 35712864
B - 13578642
D - 71356428
D - 64287135
C - 42865713
------------
E - 00000000
------------
0 Total Column Matches
======================
Answer 3: Same as 1, 3, 6, 7, 9, 12, 13, 15
------------
A - 28643571
B - 86421357
D - 64287135
C - 42865713
B - 13578642
D - 71356428
C - 57134286
A - 35712864
------------
E - 00000000
------------
0 Total Column Matches
======================
Answer 4: Same as 2, 4, 5, 8, 10, 11, 14, 16
------------
A - 28643571
B - 86421357
D - 64287135
D - 71356428
C - 57134286
A - 35712864
B - 13578642
C - 42865713
------------
E - 00000000
------------
0 Total Column Matches
======================
```

Conclusion

- There are 2 unique answers.

- (A,A,B,D,C,B,D,C)
- (D,D,C,A,B,C,A,B)