Assignment 2

Rose Novack

January 26th, 2026

Part 1

Let me begin by slapping in a bash file I call `FindGraphs.sh`.

```
#!/bin/sh
cd $(dirname $0)
echo -e "\x1b[1m\x1b[33mRamsey(K_x, K_y) Brute-Forcer\x1b[0m" >&2
echo -en "\x1b[1mx = \x1b[0m\x1b[36m" >&2
read x
echo -en "\x1b[0m\x1b[1my = \x1b[0m\x1b[36m" >&2
read y
echo -e "\x1b[0m" >&2
odir=./out/R${x}_${y}
rm -rf ${odir}/
mkdir -p ${odir}/
geng -t 1 >${odir}/n1.g6
i=1
while [ $(wc -l ${odir}/n${i}.g6 | awk '{print $1}') -gt 0 ]; do
  echo -en "\x1b[34m" >&2
  date >&2
  echo -e "\x1b[32mWorking on $((i+1))/${maxvert} ...\x1b[0m" >&2
  <${odir}/n${i}.g6 addptg -n1j: | pickg -h:$((y-1))k:$((x-1)) | labelg | sort -u >${odir}/n$((i+1)).g
  echo -e "\x1b[33mFinished $((i+1))/${maxvert}\x1b[0m" >&2
 i=$((i+1))
done
```

This handy little script brute-forces whatever $R(x, y)$ you'd like. It starts out by prompting the user for the variables x and y. After that, it creates the single vertex graph and saves it as `n1.g6`. I did this so that the graphs I'm hosting on my website are comprehensive. Sure, they're also hosted on other websites, but one can never be too redundant when it comes to backing up lists of graphs. Anyways, after creating `n1.g6`, we enter a loop of extending all previous graphs with `addptg`, filtering for the properties we want with `pickg`, finding the canonical labels with `labelg`, and only keeping unique graphs with `sort -u` where the `-u` means **u**nique. This creates the next file. This loop repeats until an empty file is created.

Now that we have this script, let's run it with $x = 3$ and $y = 5$. This will create `n4.g6` in a more long-winded way than `geng -t 4`. I will put `n4.g6` below:

n4.g6
C?
CC
CE
CF
CQ
CU
C]

We also have `n12.g6`, `n13.g6`, and `n14.g6`. Let's take a look at those:

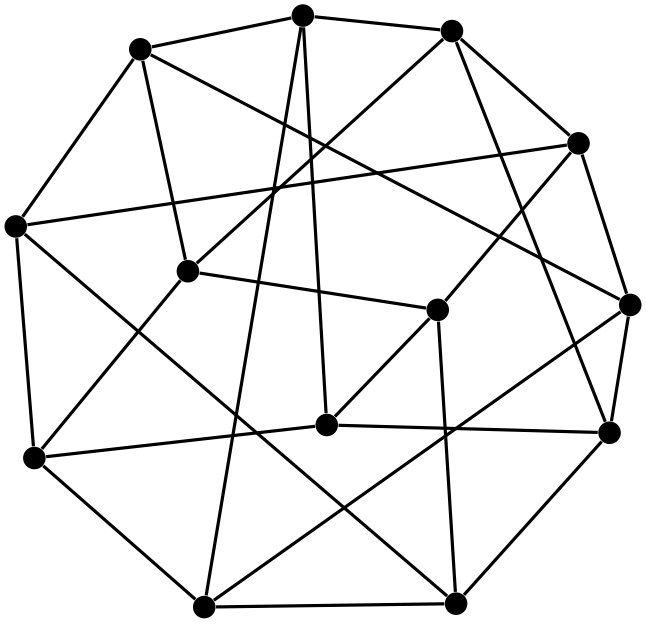
n12.g6	n13.g6	n14.g6
K?CkQMp[cgL@	Ls`?XGRQR@B`Kc	
K?GTa\cUDGrC		
K?_YPMQoPokc		
K@AAHWYoYwTO		
KQ`?pMCQ?bcU		
K`?CGtDIkwL_		
K``@OkcEICoL		
K`aAAGUEpRDo		
K`aAIOiDWsCh		
KoCIHaO@XDHB		
Ks_GagjLASko		
Ks_HIGZKQSm_		

Oh, I sorted those using `LC_ALL=C sort`, by the way. I like them better sorted even though there's no reason to do that, really.

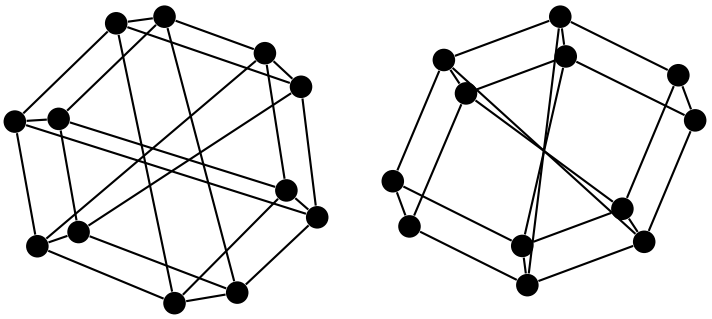
Note: I had to leave my laptop running overnight to do $R(3, 6)$, though $R(3, 5)$ did not require much waiting.

Part 2

Let's draw one of the many graphs from `n13.g6`.
(Joke: There is only one graph in `n13.g6`.)



Ok, now to draw the two most symmetric graphs among those in `n12.g6`. I'm drawing `K`aAIOiDWsCh` and `Ks_GagjLASko` respectively. (Graphs number 9 and 11 of the 12.)



On the **next page**, I will **explain the automorphism groups' generators**. I wanted to keep these big pictures on their own page because they take up a lot of space.

Generators of Automorphism Groups

Here's where we start using `dreadnaut`. Let's look at the output for our two graphs:

```
> <G9
> x
(2 3)(4 5)(6 7)(8 9)(10 11)
(2 3)(8 10)(9 11)
level 2:  6 orbits; 8 fixed; index 4
(0 1)(4 6)(5 7)
(0 2)(1 3)(4 8)(5 11)(6 10)(7 9)
level 1:  2 orbits; 0 fixed; index 4
2 orbits; grpsize=16; 4 gens; 9 nodes; maxlev=3
cpu time = 0.00 seconds

> <G11
> x
(1 2)(3 4)(5 6)(7 8)(9 10)
(1 2)(7 9)(8 10)
level 2:  6 orbits; 7 fixed; index 4
(0 1)(2 11)(3 7)(4 10)(5 9)(6 8)
(0 3)(1 7)(2 9)(4 6)(5 11)(8 10)
level 1:  1 orbit; 0 fixed; index 12
1 orbit; grpsize=48; 4 gens; 9 nodes; maxlev=3
cpu time = 0.00 seconds
```

Both graphs have four generators. They all involve swapping pairs of vertices, so nothing's happening with, like, cycling three or more vertices around. That happens for cyclic graphs, by the way.

The fact that the graphs kinda look like hexagonal prisms shows some of the symmetry, as they can be flipped over, reflected, and more.

Generators	
K`aAIOiDWsCh	Ks_GagjLASko
(2 3)(4 5)(6 7)(8 9)(10 11)	(1 2)(3 4)(5 6)(7 8)(9 10)
(2 3)(8 10)(9 11)	(1 2)(7 9)(8 10)
(0 1)(4 6)(5 7)	(0 1)(2 11)(3 7)(4 10)(5 9)(6 8)
(0 2)(1 3)(4 8)(5 11)(6 10)(7 9)	(0 3)(1 7)(2 9)(4 6)(5 11)(8 10)