1 Question 1

1.1

See code.

1.2

```
                      Device Bandwidth GB/sec
---------------------------------------------------------------------
                          char           uint          uint2
Problem Size MB
        1.23515        24.3216        79.9968        93.1205
         2.4703        48.0977        106.773        114.706
         4.9406        54.0027        125.523        133.964
         9.8812        57.7983         137.33        146.867
        19.7624        60.5199        143.689        154.703
        39.5248        61.4533        146.068        159.539
        79.0496        61.8805        149.951        162.067
        158.099        62.1447        151.075        163.309
        316.198        62.2718        144.983         163.82
```

1.3

Considering the theoretical peak bandwith is 240GB/s, but that for the char kernel it is estimated to be at most 25%, this gives us an expected bandwidth of about 60GB/s. We see roughly this bandwidth for the char kernel once the problem size exceeds 19.7624 MB. For the smaller problem sizes the execution time is likely being dominated by read and write duration and the capacity of the GPU is being underutilized.

The speed up from switching to uint happens because the number of reads and writes is reduced. For every 4 chars we only need to read one uint, the execution time for the threads when using chars and using uints is most likely very similar since we are only doing a single addition. This is why for the smaller problem sizes we see a roughly 4x speedup. For the larger problem sizes we cannot load all the text data into the registers on the GPU at once, so we start to run against the memory bandwidth limits.

For the unit2, at smaller sizes, we get a further speedup but still are dominated by the read and write times. However for larger problems since we have 8 times fewer read and writes, we approach the benchmark speed of the K80 on Google cloud.

2 Question 2

2.1

See code.

2.2

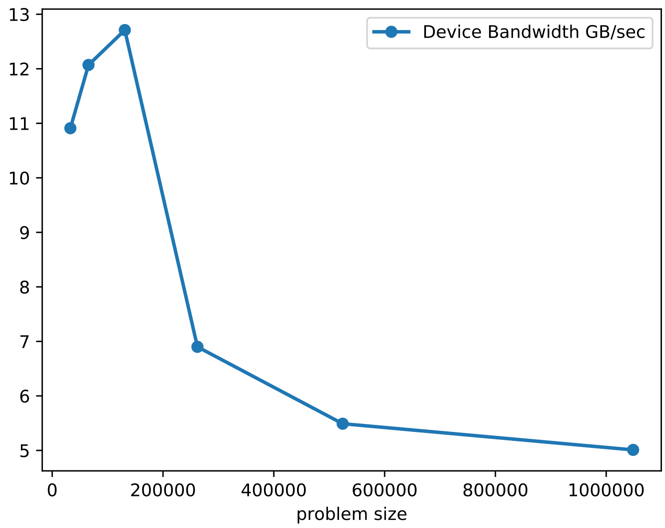
For every call to device_graph_propagate we write one float and read 1 unsigned integer and 2 floats for every edge of the current node.

Therefore the total number of bytes read is

$$total\_bytes = node(f + edge(u + 2f))NUM\_ITERATIONS \tag{1}$$

where f is the size of a float and u is the size of an unsigned integer in bytes.

2.3



2.4

For this problem, compared to problem 2, we are doing many more reads for each kernel call. The number of reads increases as the average edge size increases. This is reflected in the bandwidth measurements, where bandwidth decreases as average number of edges increases.

For example, with an average number of edges we are doing 31 reads and we get slightly more than 1/31 of the theoretical peak performance.

```
                                   Device Bandwidth GB/sec

                                                 Number of nodes
                        32768          65536         131072         262144         524288        1048576
Avg. no. edges
             2          12.12          14.27          15.22           9.59           7.09           6.31
             3          11.91          13.61          14.67           9.25           6.80           6.03
             4          12.23          13.44          14.45           8.77           6.54           5.81
             5          12.05          13.22          14.12           8.39           6.31           5.64
             6          10.08          11.10          11.60           8.11           5.78           5.19
             7          10.03          10.87          11.39           7.89           5.58           5.08
             8          10.95          11.68          12.57           7.45           5.68           5.18
             9          10.71          11.72          12.13           7.17           5.59           5.06
            10          10.91          12.07          12.71           6.90           5.49           5.01
            11          10.84          11.86          12.43           6.68           5.42           4.93
            12          10.55          11.54          12.15           6.39           5.28           4.82
            13          10.13          10.98          11.83           6.20           5.16           4.74
            14          10.21          11.01          11.55           6.07           5.06           4.67
            15          10.02          11.00          11.27           5.89           4.97           4.59
            16           9.95          10.87          11.00           5.75           4.87           4.51
            17           9.43          10.42          10.72           5.61           4.80           4.47
            18           9.52          10.36          10.54           5.46           4.72           4.39
            19           9.28          10.36          10.28           5.37           4.65           4.34
```