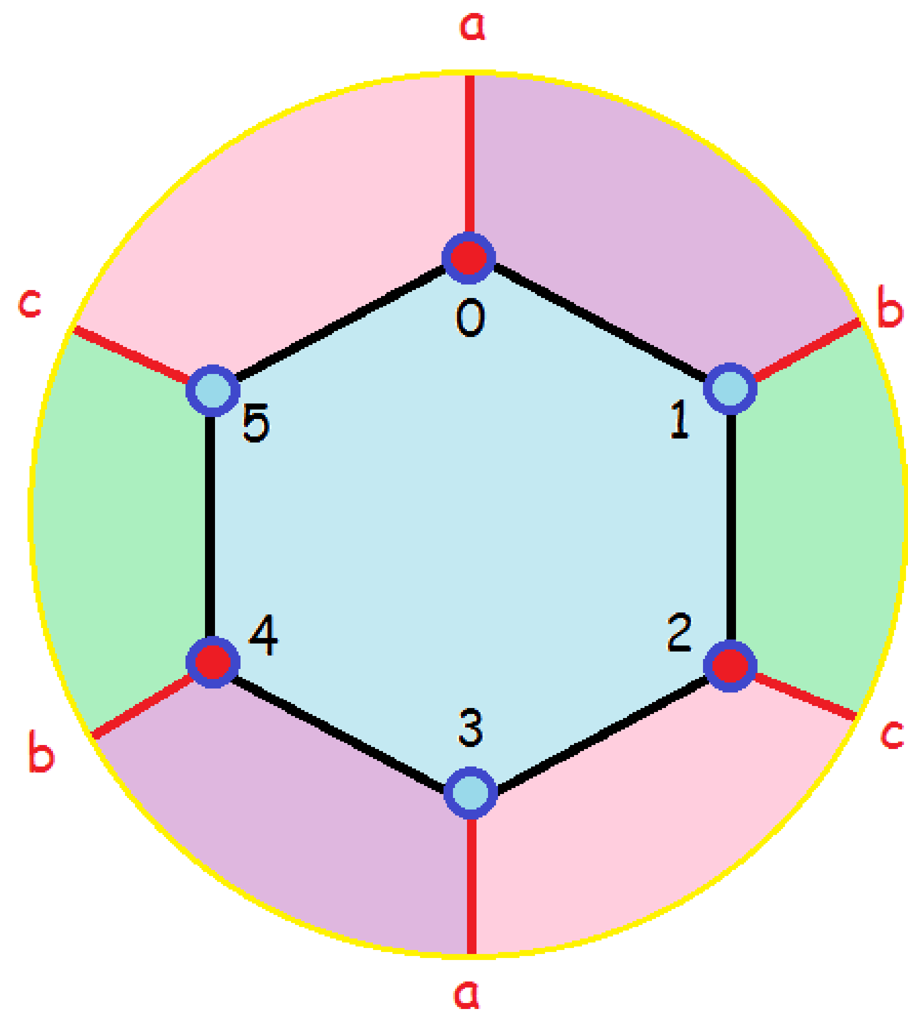
Walking the faces of non-orientable embeddings: Edges with sign -1 are red.



0: 1 5 3

1: 0 4 2

2: 1 5 3

3: 0 4 2

4: 1 5 3

5: 0 4 2

Consider a rotation system that has a vertex v with the neigbours listed in clockwise order as:

v: $u_0, u_1, u_2, \ldots, u_d$

This vertex is incident to d gaps and the gaps correspond to corners of the faces it is on.

The gaps for v are:

g_0: (u_0, v, u_1)

g_1: (u_1, v, u_2)

…

g_{d-1}: (u_{d-1}, v, u_d)

g_d: (u_d, v, u_0)

In our data structure we have made the (arbitrary) decision to store the gap information for g_i in array position i.

Each has a face number (corresponding to the face the gap is on) and a gap parity.
The gap parity is either +1 or -1.

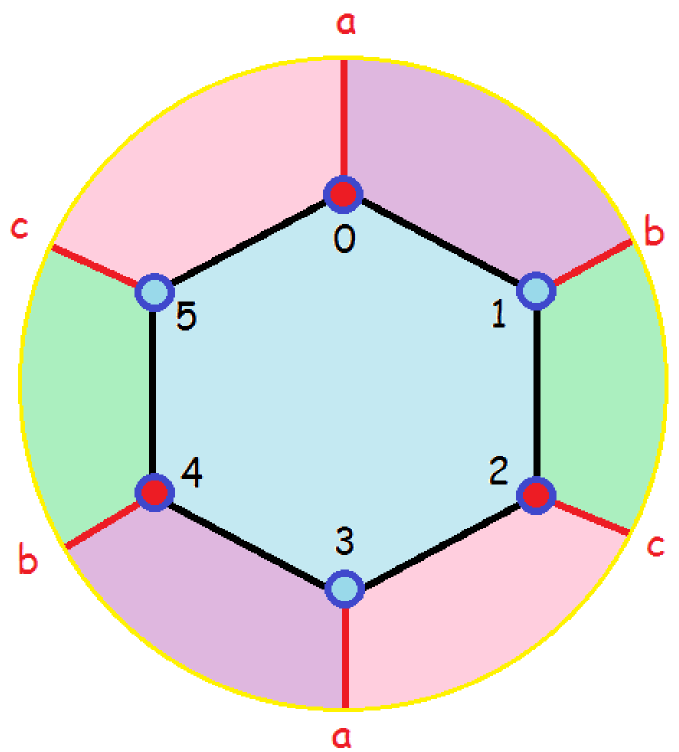
For each face, the gap where the face traversal starts is assigned gap parity +1.

As the face is traversed, the current gap parity is:

-1 : if an odd number of -1 edges have been traversed so far

+1: if an even number of -1 edges have been traversed so far.

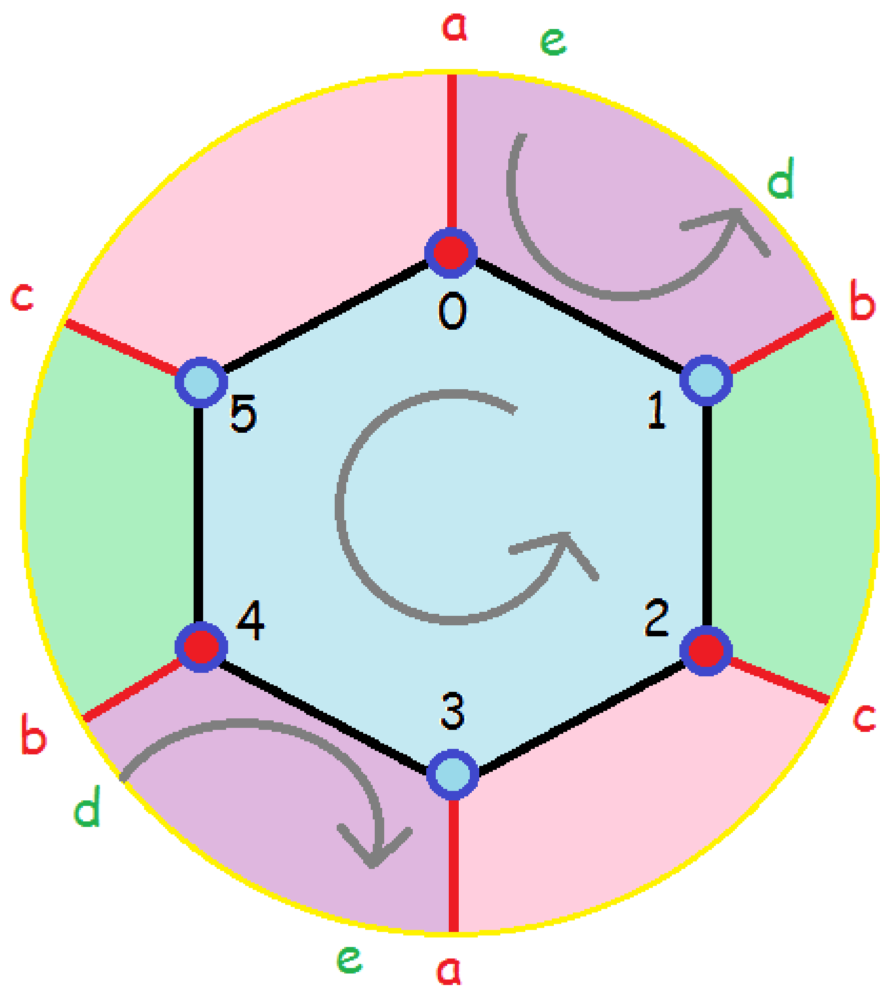
Black gaps: parity +1, traversing face ccw by choosing the next neighbour in cw order.
Red gaps: parity -1, traversing cw by choosing the next neighbour in ccw order.



Gaps for blue face:
(1, 0, 5)
(0, 5, 4)
(5, 4, 3)
(4, 3, 2)
(3, 2, 1)
(2, 1, 0)

Gaps for purple face:
(0, 1, 4)
(1, 4, 3)
(4, 3, 0)
(3, 0, 1)

Black gaps: parity +1, traversing face ccw by choosing the next neighbour in cw order.
Red gaps: parity -1, traversing cw by choosing the next neighbour in ccw order.



Theorem: Each face of an embedding has an even number of -1 edges.

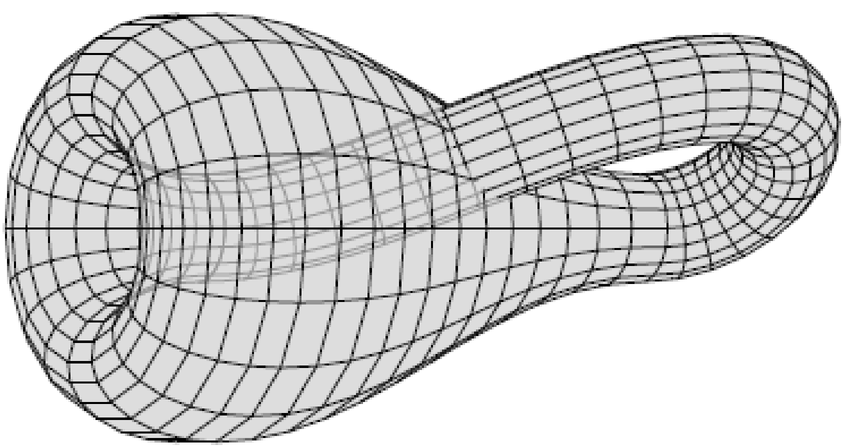
Proof:

The face traversal starts with a record [(u, v), +1] and does not end until revisiting [(u, v), +1].

Since the final sign is +1, an even number of -1 edges were traversed.

Euler genus $g = (2 - n + m - f)$
0 plane, 1 projective plane, 2 torus if orientable and Klein bottle if non-orientable

http://www.map.mpim-bonn.mpg.de/2-manifolds



Klein bottle:
non-orientable genus 2

The face number and gap parity information enables us to compute the change to the Euler genus (if the surface is actually orientable, the Euler genus is 2 times the non-orientable genus) that results from adding an edge into two gaps in O(1) time:

If the two gaps are on the same face:
If the gap parities are the same: no change +1 edge genus increases by +1 with a -1 edge.
If the gap parities are different: -1 edge, genus is the same, +1 edge, genus increases by 1.

If the two gaps are on different faces:
the change is +2.

Initialization:

```
for (i=0; i < n; i++)
{
     for (j=0; j < degree[i]; j++)
     {
       face_num[i][j]= -1; // NULL
       gap_parity[i][j]= 0; // NULL
                             }
      }
}
```

To walk all the faces:

```
nf=  0;
for (i=0; i < n; i++)
{
    for (j=0; j < degree[i]; j++)
    {
        if (face_num[i][j]== -1)
        {
            walk_face(i, j, nf,
                      n, degree, G, sign,
                      face_num, gap_parity);
            nf++

        }
    }
}
```

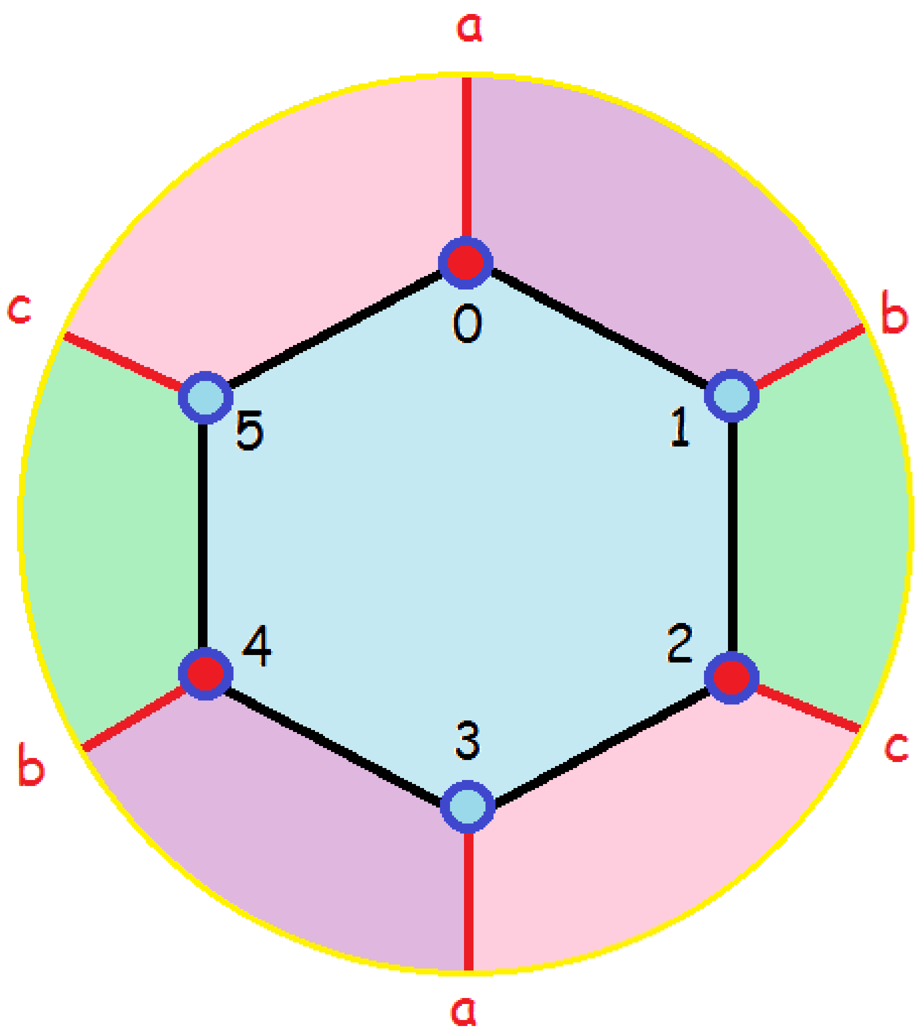
```
// Walk a face assigning my_face_num to the gaps.
// Start with gap for vertex u and jth
// neighbour of u.
int walk_face(int start_u, int start_pos,
    int my_face_num,
    int n, int degree[NMAX], int G[NMAX][NMAX],
    int sign[NMAX][NMAX],
    int face_num[NMAX][NMAX],
    int gap_parity[NMAX][NMAX])
{
   int u, v, w, first_u, first_v;
   int direction, pos;
```

```
u= G[start_u][start_pos];
v= start_u;
direction= 1;

first_u= u;
first_v= v;

#if DEBUG
        printf("Face %2d: \n",
my_face_num);
#endif
```

```
 do
 {
#if DEBUG
     printf("[(%3d, %3d), %2d]\n",
                    u, v, direction);
#endif

     for (pos=0; pos < degree[v]; pos++)
     {
       if (G[v][pos]== u) goto found;
     }
     printf("Error- neighbour %3d of %3d not found\n",
            u, v);
     exit(0);

found:
```

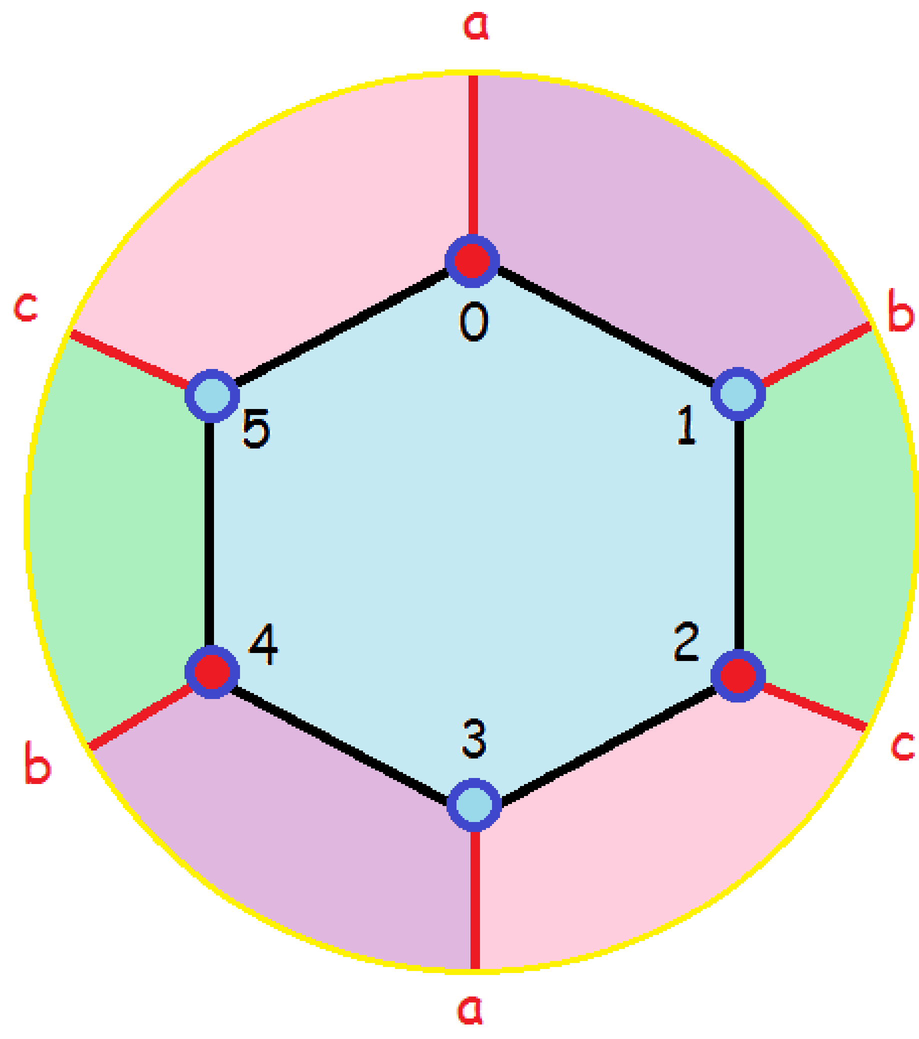
```
found:
    if (direction == 1)
    {
        face_num[v][pos]= my_face_num;
        gap_parity[v][pos]= direction;
    }

    pos+= direction;
    pos= (pos + degree[v]) % degree[v];
```

```
    if (direction == -1)
    {
      face_num[v][pos]= my_face_num;
      gap_parity[v][pos]= direction;
    }
    direction *= sign[v][pos];

    w= G[v][pos];  u=v; v=w;

    } while (first_u != u ||
             first_v != v ||
             direction  != 1);
}
```

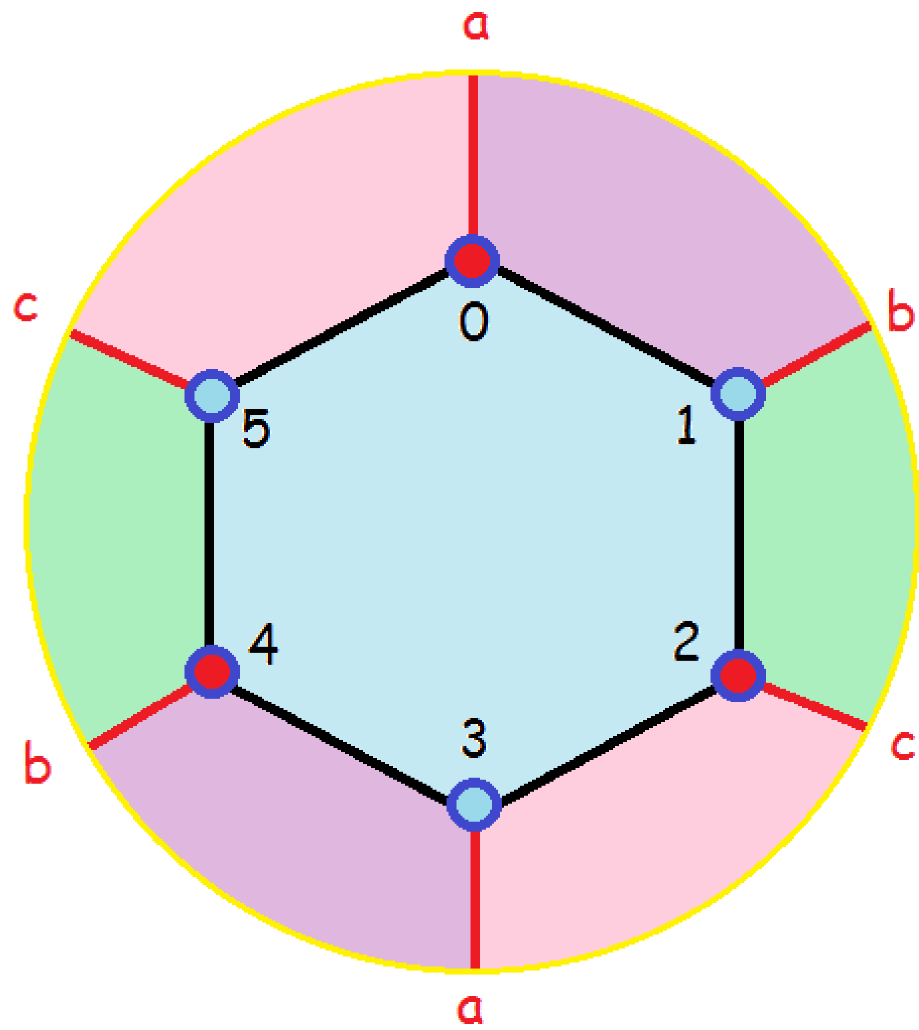


Input:

```
6
3   1   1   5   1   3  -1
3   0   1   4  -1   2   1
3   1   1   5  -1   3   1
3   0  -1   4   1   2   1
3   1  -1   5   1   3   1
3   0   1   4   1   2  -1
```




```
Face 0:
[(1, 0),  1]
[(0, 5),  1]
[(5, 4),  1]
[(4, 3),  1]
[(3, 2),  1]
[(2, 1),  1]
Face 1:
[(5, 0),  1]
[(0, 3), -1]
[(3, 2), -1]
[(2, 5),  1]
```

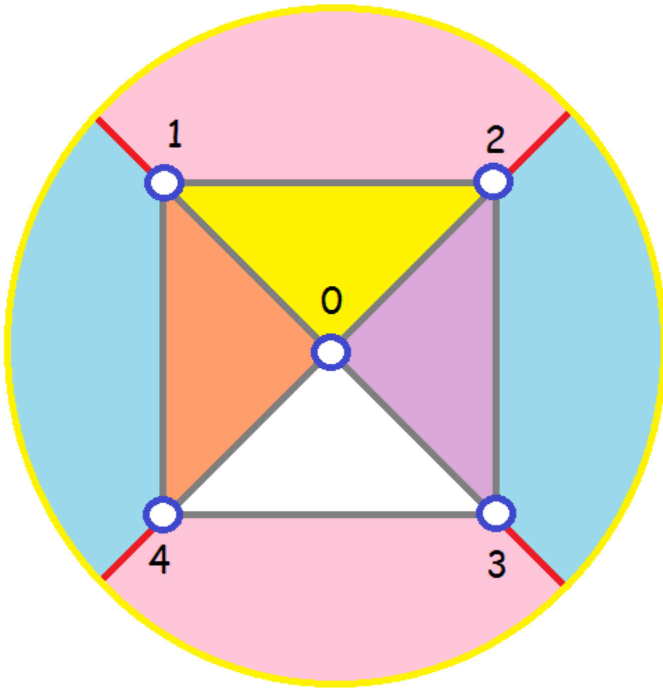


Face 2:
[(3, 0), 1]
[(0, 1), 1]
[(1, 4), -1]
[(4, 3), -1]
Face 3:
[(4, 1), 1]
[(1, 2), 1]
[(2, 5), -1]
[(5, 4), -1]

Final data structures:

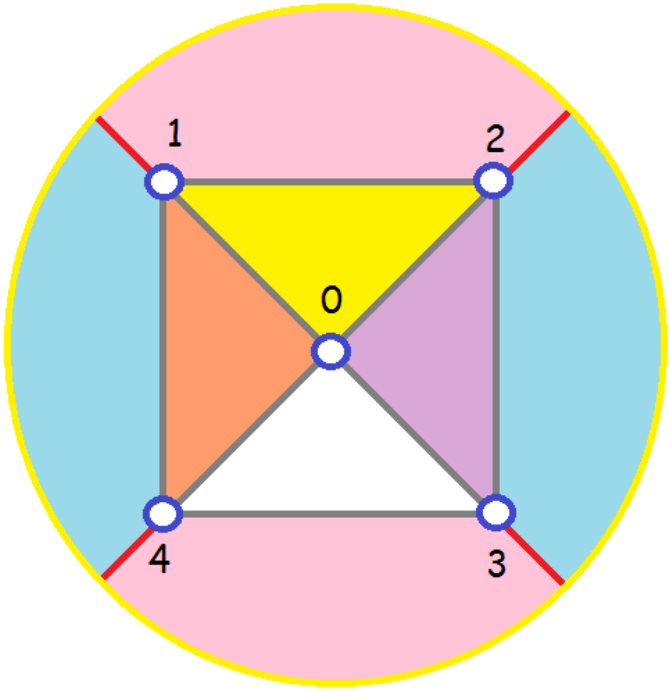
u(degree)
Then for each neighbour u:
[v, sign(u,v), face_num gap_parity]

```
0(3): [1, +, 0+] [5, +, 1+] [3, -, 2+]
1(3): [0, +, 2+] [4, -, 3+] [2, +, 0+]
2(3): [1, +, 3+] [5, -, 1-] [3, +, 0+]
3(3): [0, -, 2-] [4, +, 0+] [2, +, 1-]
4(3): [1, -, 3-] [5, +, 0+] [3, +, 2-]
5(3): [0, +, 0+] [4, +, 3-] [2, -, 1+]
```



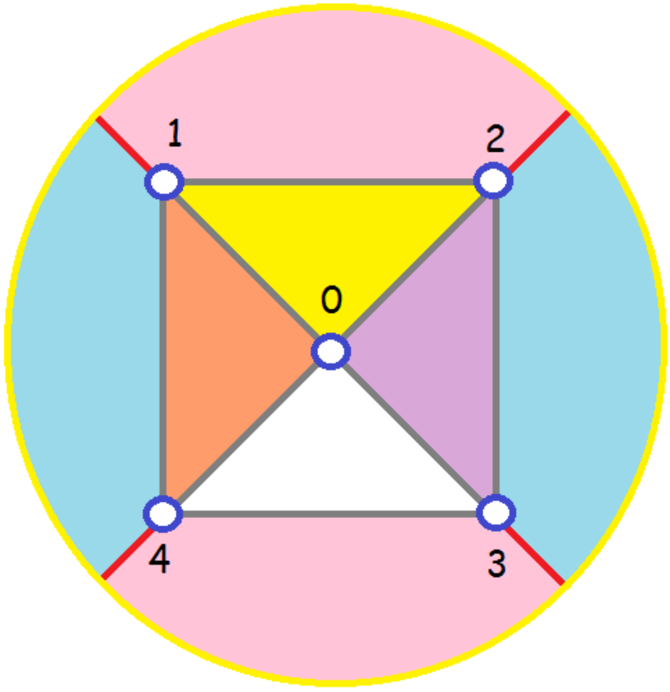
5

4	1	1	2	1	3	1	4	1
4	0	1	4	1	3	-1	2	1
4	0	1	1	1	4	-1	3	1
4	0	1	2	1	1	-1	4	1
4	0	1	3	1	2	-1	1	1



Face 1:
[(2, 0), 1]
[(0, 3), 1]
[(3, 2), 1]
Face 2:
[(3, 0), 1]
[(0, 4), 1]
[(4, 3), 1]
Face 3:
[(4, 0), 1]
[(0, 1), 1]
[(1, 4), 1]

Face 0:
[(1, 0), 1]
[(0, 2), 1]
[(2, 1), 1]

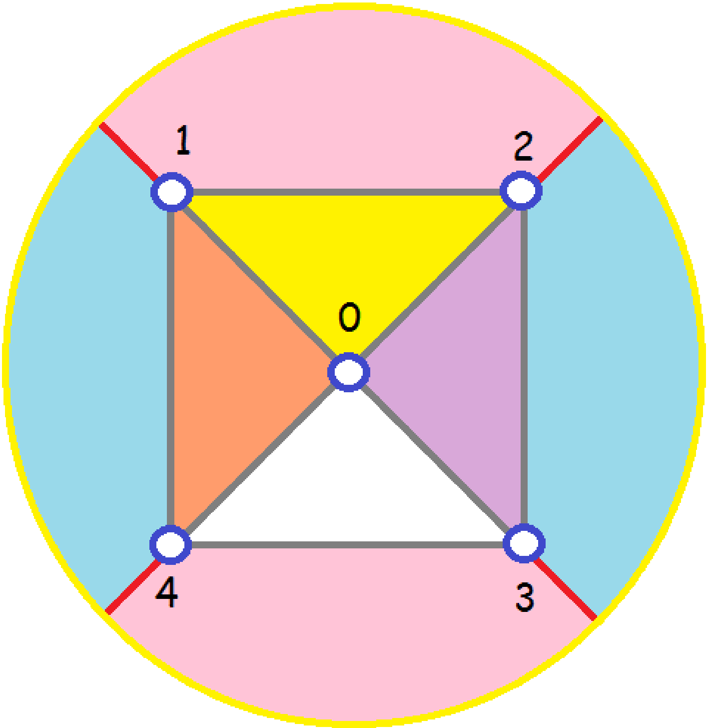


Face 4:

```
[(4, 1),  1]
[(1, 3), -1]
[(3, 2), -1]
[(2, 4),  1]
```

Face 5:

```
[(3, 1),  1]
[(1, 2),  1]
[(2, 4), -1]
[(4, 3), -1]
```



0(4):[1,+,0+][2,+,1+][3,+,2+][4,+,3+]
1(4):[0,+,3+][4,+,4+][3,-,5+][2,+,0+]
2(4):[0,+,0+][1,+,5+][4,-,4-][3,+,1+]
3(4):[0,+,1+][2,+,4-][1,-,5-][4,+,2+]
4(4):[0,+,2+][3,+,5-][2,-,4+][1,+,3+]