Kernel: SageMath 10.4

Groupe symétrique

In [1]:
```
[[s,[s(i) for i in [1..4]]] for s in SymmetricGroup(3)]
```

Out[1]:
```
[[(), [1, 2, 3, 4]],
 [(1,3,2), [3, 1, 2, 4]],
 [(1,2,3), [2, 3, 1, 4]],
 [(2,3), [1, 3, 2, 4]],
 [(1,3), [3, 2, 1, 4]],
 [(1,2), [2, 1, 3, 4]]]
```

Bataille

In [2]:
```
def reglej1(c1,c2):
    return [c1,c2]

def reglewin(c1,c2):
    return [max(c1,c2),min(c1,c2)]

def regleloose(c1,c2):
    return [min(c1,c2),max(c1,c2)]
```

In [3]:
```
def tour(j1,j2,regle):
    add = regle(j1[0], j2[0])
    if j1[0] < j2[0]:
        return j1[1:], j2[1:]+add
    return j1[1:]+add, j2[1:]
```

In [4]:
```
tour([1,2,5],[3,4,6],reglej1)
```

Out[4]:
```
([2, 5], [4, 6, 1, 3])
```

In [5]:
```
tour([1,2,5],[3,4,6],reglewin)
```

Out[5]:
```
([2, 5], [4, 6, 3, 1])
```

In [6]:
```
tour([1,2,5],[3,4,6],regleloose)
```

Out[6]:
```
([2, 5], [4, 6, 1, 3])
```

In [7]:
```
def joue(j1,j2,regle,lim):
    compt = 0
    while compt < lim:
        if len(j1) == 0 or len(j2) == 0:
            return compt
        j1,j2 = tour(j1,j2,regle)
        compt += 1
    return lim
```

In [8]:
```
def joueprint(j1,j2,regle,lim):
    compt = 0
    while compt <= lim:
        print(j1,len(j1),"\n",j2,len(j2),"\n\n")
        if len(j1) == 0 or len(j2) == 0:
            return compt
        j1,j2 = tour(j1,j2,regle)
        compt += 1
    return lim
```

In [9]:
```
joueprint([1,3,6],[2,4,5],reglej1,20)
```

Out[9]: [1, 3, 6] 3
 [2, 4, 5] 3

[3, 6] 2
 [4, 5, 1, 2] 4

[6] 1
 [5, 1, 2, 3, 4] 5

[6, 5] 2
 [1, 2, 3, 4] 4

[5, 6, 1] 3
 [2, 3, 4] 3

[6, 1, 5, 2] 4
 [3, 4] 2

[1, 5, 2, 6, 3] 5
 [4] 1

[5, 2, 6, 3] 4
 [1, 4] 2

[2, 6, 3, 5, 1] 5
 [4] 1

[6, 3, 5, 1] 4
 [2, 4] 2

[3, 5, 1, 6, 2] 5
 [4] 1

[5, 1, 6, 2] 4
 [3, 4] 2

[1, 6, 2, 5, 3] 5
 [4] 1

[6, 2, 5, 3] 4
 [1, 4] 2

[2, 5, 3, 6, 1] 5
 [4] 1

[5, 3, 6, 1] 4

```
 [2, 4] 2


[3, 6, 1, 5, 2] 5
 [4] 1


[6, 1, 5, 2] 4
 [3, 4] 2


[1, 5, 2, 6, 3] 5
 [4] 1


[5, 2, 6, 3] 4
 [1, 4] 2


[2, 6, 3, 5, 1] 5
 [4] 1



20
```

In [10]:

```
joueprint([3,1],[2,4],reglej1,6)
```

Out[10]:
```
[3, 1] 2
 [2, 4] 2


[1, 3, 2] 3
 [4] 1


[3, 2] 2
 [1, 4] 2


[2, 3, 1] 3
 [4] 1


[3, 1] 2
 [2, 4] 2


[1, 3, 2] 3
 [4] 1


[3, 2] 2
 [1, 4] 2



6
```

In [14]:
```
joue([1,4,6],[2,3,5],reglej1,100)
```

Out[14]:
```
5
```

In [15]:
```
joueprint([1,4,6],[2,3,5],reglej1,100)
```

Out[15]:
```
[1, 4, 6] 3
 [2, 3, 5] 3


[4, 6] 2
 [3, 5, 1, 2] 4


[6, 4, 3] 3
 [5, 1, 2] 3


[4, 3, 6, 5] 4
 [1, 2] 2


[3, 6, 5, 4, 1] 5
 [2] 1


[6, 5, 4, 1, 3, 2] 6
 [] 0



5
```

In [16]:
```
joueprint([1,4,6],[2,3,5],reglewin,100)
```

Out[16]:
```
[1, 4, 6] 3
 [2, 3, 5] 3


[4, 6] 2
 [3, 5, 2, 1] 4


[6, 4, 3] 3
 [5, 2, 1] 3


[4, 3, 6, 5] 4
 [2, 1] 2


[3, 6, 5, 4, 2] 5
 [1] 1


[6, 5, 4, 2, 3, 1] 6
 [] 0



5
```

In [17]:
```
joueprint([1,4,6],[2,3,5],regleloose,100)
```

Out[17]:
```
[1, 4, 6] 3
 [2, 3, 5] 3


[4, 6] 2
 [3, 5, 1, 2] 4


[6, 3, 4] 3
 [5, 1, 2] 3


[3, 4, 5, 6] 4
 [1, 2] 2


[4, 5, 6, 1, 3] 5
 [2] 1


[5, 6, 1, 3, 2, 4] 6
 [] 0



5
```

Détection de parties infinies

In [18]:
```
def detecte_infini(j1,j2,regle):
    compt = 0
    comp = set()
    res = true
    while res:
        if len(j1) == 0 or len(j2) == 0:
            return false,compt
        if (tuple(j1),tuple(j2)) in comp:
            return true,compt
        comp.add((tuple(j1),tuple(j2)))
        j1,j2 = tour(j1,j2,regle)
        compt += 1
    raise ValueError
```

In [19]:
```
detecte_infini([2,3],[1,4],reglej1)
```

Out[19]: (False, 4)

In [20]:
```
detecte_infini([1,3,6],[2,4,5],reglej1)
```

Out[20]: (True, 17)

In [21]:
```
detecte_infini([1,4,6],[2,3,5],reglej1)
```

Out[21]: (False, 5)

Génération de toutes les parties

In [22]:
```
def gen_from_perm_even(n):
    # génère tous les débuts de parties possibles pour un nombre pair de
cartes
    assert n%2==0,"le nombre de cartes n est impair"
    res = []
    tot = factorial(n)
    compt = 0
    pc = .1
    for sigma in SymmetricGroup(n):
        compt += 1
        if compt/tot > pc / 100:
            #print(pc,"%")
            pc += .1
        sigmalist = list(sigma.tuple())
        res.append([sigmalist[:(n/2)],sigmalist[(n/2):]])
    return res
```

In [24]:
```
gen_from_perm_even(2)
```

Out[24]:
```
[[[1], [2]], [[2], [1]]]
```

In [25]:
```
gen_from_perm_even(4)
```

Out[25]:
```
[[[1, 2], [3, 4]],
 [[3, 4], [1, 2]],
 [[4, 3], [2, 1]],
 [[2, 1], [4, 3]],
 [[1, 3], [4, 2]],
 [[3, 1], [2, 4]],
 [[4, 2], [1, 3]],
 [[2, 4], [3, 1]],
 [[1, 4], [2, 3]],
 [[3, 2], [4, 1]],
 [[4, 1], [3, 2]],
 [[2, 3], [1, 4]],
 [[1, 2], [4, 3]],
 [[3, 4], [2, 1]],
 [[4, 3], [1, 2]],
 [[2, 1], [3, 4]],
 [[1, 3], [2, 4]],
 [[3, 1], [4, 2]],
 [[4, 2], [3, 1]],
 [[2, 4], [1, 3]],
 [[1, 4], [3, 2]],
 [[3, 2], [1, 4]],
 [[4, 1], [2, 3]],
 [[2, 3], [4, 1]]]
```

In [26]:
```
for cartes in gen_from_perm_even(2):
    print(cartes, detecte_infini(cartes[0],cartes[1],reglej1))
```

Out[26]:
```
[[1], [2]] (False, 1)
[[2], [1]] (False, 1)
```

In [27]:
```
for cartes in gen_from_perm_even(4):
    print(cartes, detecte_infini(cartes[0],cartes[1],reglej1))
```

Out[27]:
```
[[1, 2], [3, 4]] (False, 2)
[[3, 4], [1, 2]] (False, 2)
[[4, 3], [2, 1]] (False, 2)
[[2, 1], [4, 3]] (False, 2)
[[1, 3], [4, 2]] (True, 6)
[[3, 1], [2, 4]] (True, 4)
[[4, 2], [1, 3]] (True, 4)
[[2, 4], [3, 1]] (True, 6)
[[1, 4], [2, 3]] (False, 4)
[[3, 2], [4, 1]] (False, 4)
[[4, 1], [3, 2]] (False, 4)
[[2, 3], [1, 4]] (False, 4)
[[1, 2], [4, 3]] (False, 2)
[[3, 4], [2, 1]] (False, 2)
[[4, 3], [1, 2]] (False, 2)
[[2, 1], [3, 4]] (False, 2)
[[1, 3], [2, 4]] (False, 2)
[[3, 1], [4, 2]] (False, 2)
[[4, 2], [3, 1]] (False, 2)
[[2, 4], [1, 3]] (False, 2)
[[1, 4], [3, 2]] (True, 6)
[[3, 2], [1, 4]] (True, 4)
[[4, 1], [2, 3]] (True, 4)
[[2, 3], [4, 1]] (True, 6)
```

In [28]:
```
for cartes in gen_from_perm_even(4):
    print(cartes, detecte_infini(cartes[0],cartes[1],reglewin))
```

Out[28]:
```
[[1, 2], [3, 4]] (False, 2)
[[3, 4], [1, 2]] (False, 2)
[[4, 3], [2, 1]] (False, 2)
[[2, 1], [4, 3]] (False, 2)
[[1, 3], [4, 2]] (False, 6)
[[3, 1], [2, 4]] (False, 6)
[[4, 2], [1, 3]] (False, 6)
[[2, 4], [3, 1]] (False, 6)
[[1, 4], [2, 3]] (False, 4)
[[3, 2], [4, 1]] (False, 4)
[[4, 1], [3, 2]] (False, 4)
[[2, 3], [1, 4]] (False, 4)
[[1, 2], [4, 3]] (False, 2)
[[3, 4], [2, 1]] (False, 2)
[[4, 3], [1, 2]] (False, 2)
[[2, 1], [3, 4]] (False, 2)
[[1, 3], [2, 4]] (False, 2)
[[3, 1], [4, 2]] (False, 2)
[[4, 2], [3, 1]] (False, 2)
[[2, 4], [1, 3]] (False, 2)
[[1, 4], [3, 2]] (False, 4)
[[3, 2], [1, 4]] (False, 4)
[[4, 1], [2, 3]] (False, 4)
[[2, 3], [4, 1]] (False, 4)
```