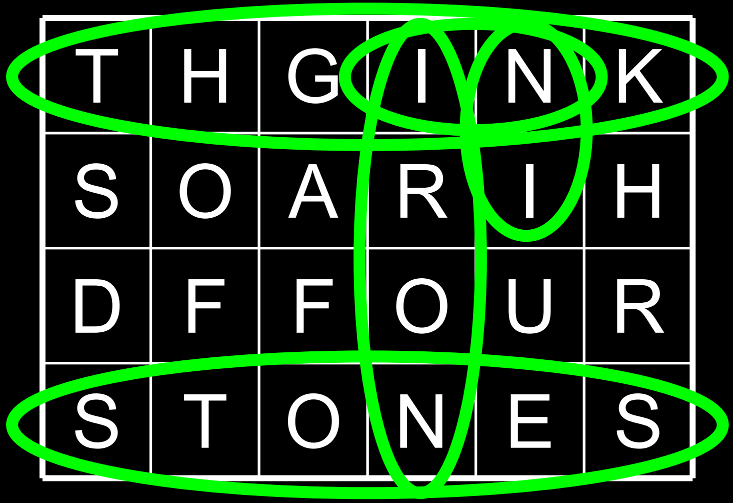
The French Taunter // Le Taunter français

#define end lol

Test Case

taunter0.in

```
4 6 5
THGINK
SOARIH
DFFOUR
STONES
NI
IRON
KNIGHT
HAMSTER
STONES
```



T	H	G	I	N	K
S	O	A	R	I	H
D	F	F	O	U	R
S	T	O	N	E	S

taunter0.out

```
BOTH
VERTICAL
HORIZONTAL
NEITHER
HORIZONTAL
```

Brute [or ‘in’] method

```
fin = file('taunter.in','r')
fout = file('taunter.out','w')
n , m ,r = fin.readline().split()
chars = []
vert_string , hor_string = "",""
for x in xrange(int(n)):
    currentline = fin.readline()
    ar_line = [e for e in currentline.strip()]
    chars.append(ar_line)

for x in xrange(int(n)):
    for y in xrange(int(m)):
        hor_string+= chars[x][y]
    hor_string+= "#"

for x in xrange(int(m)):
    for y in xrange(int(n)):
        vert_string+= chars[y][x]
    vert_string+= "#"

for word in xrange(int(r)):
    wd = fin.readline().strip()
    backwd= ""
    for i in xrange(len(wd)-1,-1,-1):
        backwd+=wd[i]

    vr  = (vert_string.count(wd))  or (vert_string.count(backwd))
    hr  = (hor_string.count(wd)) or (hor_string.count(backwd))
    if (hr and vr): print >> fout, "BOTH"
    elif (hr and not(vr)): print >> fout, "HORIZONTAL"
    elif (not(hr) and vr): print >> fout, "VERTICAL"
    elif not(hr and vr): print >> fout, "NEITHER"
fin.close()
fout.close()
```

← Horizontal Strings

← # Stops runover lines or line spilling

← Vertical strings.

← Reverse strings.

Dictionary

```
fin = file('taunter.in','r')
fout = file('taunter.out','w')
n , m ,r = fin.readline().split()
dicti = {}

def rev(s):
    backwd= ""
    for i in xrange(len(s)-1,-1,-1):
        backwd+=s[i]
    return backwd

def words(s,dicti,ori):
    for sv in xrange(len(s)):
        for ev in xrange(sv,len(s)+1):
            if dicti.has_key(s[sv:ev]):
                if ori in dicti[s[sv:ev]]:
                    pass
                else:
                    dicti[s[sv:ev]] = str(dicti[s[sv:ev]]) + str(ori)
            else:
                dicti[s[sv:ev]]=ori
    return dicti

chars = []
for x in xrange(int(n)):
    currentline = fin.readline()
    ar_line = [e for e in currentline.strip()]
    chars.append(ar_line)

for x in xrange(int(n)):
    st = ""
    for y in xrange(int(m)):
        st+=chars[x][y]
    dicti = words(st,dicti,"V")
    dicti = words(rev(st),dicti,"V")
for x in xrange(int(m)):
    st = ""
    for y in xrange(int(n)):
        st+=chars[y][x]
    dicti = words(st,dicti,"H")
    dicti = words(rev(st),dicti,"H")

for x in xrange(int(r)):
    e= fin.readline()
    if dicti.has_key(e.strip()):
        if ("H" in dicti[e.strip()]) and ("V" in dicti[e.strip()]):
            print >> fout, "BOTH"
        elif "H" in dicti[e.strip()]:
            print >> fout, "HORIZONTAL"
        elif "V" in dicti[e.strip()]:
            print >> fout, "VERTICAL"
    else:
        print >> fout, "NEITHER"

fin.close()
fout.close()
```

Trie – Marco’s solution

```
fin = open("taunter.in", "r")
fout = open("taunter.out", "w")

N, M, W = map(int, fin.readline().split(" "))
h = [fin.readline().strip() for i in xrange(N)]
v = ["".join([h[j][i] for j in xrange(N)]) for i in xrange(M)]
h += [a[::-1] for a in h]
v += [a[::-1] for a in v]
t = tri()
for i in xrange(W):
    t.insert(fin.readline().strip(), 0, i)
found = [[False] * W for i in xrange(2)]
for a in h:
    for i in xrange(len(a)):
        for w in t.search(a, i):
            found[0][w] = True
for a in v:
    for i in xrange(len(a)):
        for w in t.search(a, i):
            found[1][w] = True
for i in xrange(W):
    fout.write("BOTH\n" if found[0][i] and found[1][i] else
            "HORIZONTAL\n" if found[0][i] else
            "VERTICAL\n" if found[1][i] else
            "NEITHER\n")

fin.close()
fout.close()
```

```
class tri:
    def __init__(self):
        self.c = {}                #children of node
        self.t = []                # index of words that term.
    def insert(self, w, i, n):
        if i == len(w):
            self.t.append(n)
        else:
            if not w[i] in self.c:
                self.c[w[i]] = tri()
            self.c[w[i]].insert(w, i+1, n)
    def search(self, w, i):
        if i == len(w):
            return self.t
        if w[i] in self.c:
            return self.t + self.c[w[i]].search(w, i+1)
        return self.t
```

Hashtable

Too long for here.. Just one quote:
“//By Charles Bradshaw (shamelessly edited from Carls cpp solution) “

Running times

	Brute force			Dictionary/ Hashtable			Trie		
Case	Python	C++	Java	Python	C++	Java	Python	C++	Java
1	0.024	0.089	0.003	0.024	0.004	0.063	0.025	0.004	n/a
2	0.055	0.092	0.005	0.031	0.007	0.092	0.033	0.005	n/a
3	0.318	0.098	0.011	0.041	0.011	0.144	0.084	0.009	n/a
4	0.605	0.111	0.016	0.053	0.020	0.151	0.110	0.010	n/a
5	3.038	0.149	0.080	0.134	0.056	0.285	0.492	0.043	n/a
6	fail	fail	fail	1.077	0.703	0.836	4.490	0.219	n/a
7	fail	fail	fail	1.086	0.783	0.914	5.103	0.251	n/a
8	fail	fail	fail	1.041	0.692	0.838	4.474	0.211	n/a
9	fail	fail	fail	1.147	0.839	0.981	crash	0.271	n/a
10	fail	fail	fail	1.158	0.823	0.994	crash	0.265	n/a

LOL;

return 0;