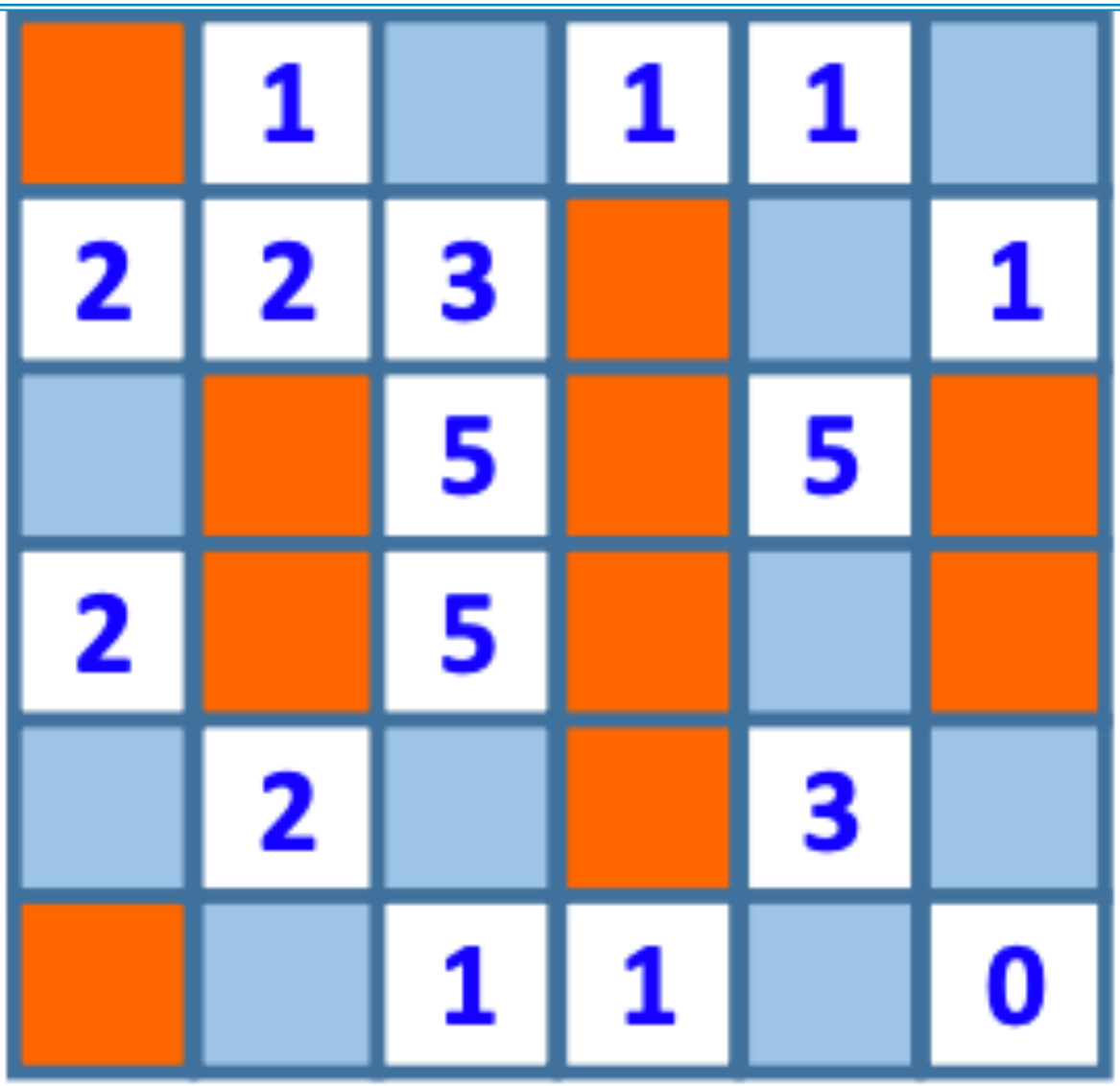
CSP - Minesweeper

Introduction to Artificial Intelligence Project 2

0712238 Yan-Tong Lin - May 1, 2020



Task Description

Solve a classic game - “MineSweeper” by seeing it as a CSP problem.

Design a backtracking algorithm for it.

Discuss difference when applying different strategy listed below:

Forward Checking, MRV heuristic, Degree heuristic, LCV heuristic

I also proposed a cut named CUTA,

if a variable cannot be assign to any variables in the node, all siblings need not to be explored

Implementation Details

Language and Tools:

1. Python 3
2. Editor: Jupyter Notebook
3. Version Control: git
4. Report: Mac OS Pages

Interactive programming and experimentations

Use flag to control usage of heuristic

An independent solver is available(“solver.py”)

Usage:

1. python3 solver.py
2. Use STD in to input the constraints like in the HW spec
   - A string in one line in format “n m #mine board(n*m)”

Experiments

Performance Evaluation

For evaluation of performance of algorithms,

I choose three features described as follow:

- **Node_pushed**
  - Compare with node_popped can see how heuristics do in the sense of “variable visit order”
- **Node_popped**
  - # node popped, **~ time_elapsed**

- **Node_cut**
  - **estimation of # cut tree nodes, by pow(branching factor, # unassigned VAR)**
- **Time_elapsed**
  - Real time elapsed during the algorithm
  - Since I thought time elapsed may not be ~ node popped, the two measurement is taken at the same time

Experiment Designs

- EXP1: Compare **performance of different heuristic combination** in 6*6 board with 18 variables
  - 100 random generated test cases corresponding to each n, m , # variables
- EXP2: Compare **growth of statistics** with all hyper params ON **respect to different board size / # variable**

Experiment Results

1. EXP1 - Combinations of Heuristics(ALL-, and indivisual)

test data = testcase_6_6_18_100.txt generated randomly
TLE means time limit exceeds with time limit set to 2 seconds
In all, if DEG+MRV is both used, the heuristic sort consider MRV first

ALGO/(avg)	Node Popped	Node Pushed	Node Cut	Time Elapsed	TLE
Without Forward	TLE	TLE	TLE	TLE	All
MRV	42.03	247.48	890175.08	0.17	2
Degree	135.85	320.17	1242165.37	0.26	5
LCV	-	-	-	-	All
CUTA	-	-	-	-	All
ALL - MRV	141.26	326.71	861950.91	0.2375	5
ALL - Degree	44.29	251.23	836149.90	0.15	1
ALL - LCV	59.07	265.1	825385.13	0.157	1
ALL -CUTA	66.81	272.96	828476.58	0.18	2
ALL	58.05	264.3	825352.79	0.162	1

The experiment result shows that

1. Without Forward Checking => Doomed to TLE
2. Need at least one "variable choosing heuristic"(MRV or degree)
3. However Degree does not sync with MRV well(by node popped, ALL-degree < ALL)
4. MRV better than degree with ALL- test and Individual test
5. CUTA is actually a better supporting feature than LCV
6. Also, I would like to say CSP problems solutions are not easy to be compare, since it is extremely task/test cases dependent !

2. EXP2 - #Variable (single test case for each)

BOARD/VAR/Popped	20	30	50
6x6	19(~0.1sec)	TLE	NOCASE
10x10	21(0.4sec)	31(0.85sec)	~51(2sec)
15x15	21(0.97)	31(2sec)	~51(5.48sec)

- REALLY surprisingly, node popped ~ #VAR and is the same number with randomly generated test case!!
- And since the processing of node in my algorithm is O(n^2), the time elapsed values' relation is no surprise
- I would guess that there is a threshold of ratio of #VAR/SIZE(GRID) that the problem become unsolvable in practical time.

Remaining Questions and Future Work

1. Find out the "critical ratio" in this task
2. More experiment on various test cases
3. But I think solving CSP problems in this way is essentially Backtracking Search + Heuristic + Cutting Techniques, spending too much time would be a waste of time, since the result is not generalizable in the big picture :(

Code

1. LAB2-1.py

```
#!/usr/bin/env python
# coding: utf-8

# In[1]:


import os
import numpy as np
from copy import deepcopy
from time import time
from colored import fg, bg, attr


# In[2]:


# macro
UNKNOWN = -1
CONSTRAINT = -2
ASSIGNED = 1
NOMINE = 0
MINE = 1
FAIL_MSG = "leave reached but WA"


# In[3]:


#here stores hyper params
TIMELIMIT = 3 #second

FORWARD = True
# in which order should we explore variables
```

```
# MRV, domain smaller
# DEG, vertex that affect more points
MRV = True
DEG = True
# limitation added if assign this value to the variable
LCV = True
GLOBALH = False
# If a variable can not be assign, do not need to try sidlings
CUTA = True


# In[4]:


#board related

#board size
n = 0
m = 0
#number of mines
n_mine = 0
#maps
C = list() #constraint value
A = list() #ans

# dx, dy can use for loop 3*3 - 0,0


# In[5]:


def inrange(i, j):
    global n, m
    #if(((i >= 0) and (i < n) and (j >= 0) and (j < m))):
    #    print(i, j)
    return ((i >= 0) and (i < n) and (j >= 0) and (j < m))

def printA(A):
    global n, m
```

```
    for i in range(n):
        for j in range(m):
            print(str(A[i][j]), end = " ")
        print()
```

```
# In[25]:
```

```
#input the problem
# 6 6 10 -1 -1 -1 1 1 -1 -1 3 -1 -1 -1 0 2 3 -1 3 3 2 -1 -1 2 -1 -
1 -1 -1 2 2 3 -1 3 -1 1 -1 -1 -1 1
test_data = [
    "4 4 5 2 -1 -1 2 -1 3 3 -1 2 2 2 1 -1 -1 -1 0",
    "6 6 10 -1 -1 -1 1 1 -1 -1 3 -1 -1 -1 0 2 3 -1 3 3 2 -1 -1 2 -
1 -1 -1 -1 2 2 3 -1 3 -1 1 -1 -1 -1 1",
    "6 6 10 -1 -1 -1 1 1 1 3 4 -1 2 -1 -1 2 -1 -1 -1 -1 -1 -1 -1 2
2 -1 2 1 2 -1 -1 1 -1 -1 1 -1 1 0 -1",
    "6 6 10 -1 -1 -1 -1 -1 -1 -1 2 2 2 3 -1 -1 2 0 0 2 -1 -1 2 0 0
2 -1 -1 3 2 2 2 -1 -1 -1 -1 -1 -1 -1",
    "6 6 10 -1 1 -1 1 1 -1 2 2 3 -1 -1 1 -1 -1 5 -1 5 -1 2 -1 5 -1
-1 -1 -1 2 -1 -1 3 -1 -1 -1 1 1 -1 0",
    "10 10 23 -1 1 -1 0 -1 -1 0 1 -1 1 1 -1 -1 -1 -1 -1 1 -1 1 1 0
0 -1 2 -1 -1 1 1 1 1 1 1 -1 -1 -1 -1 1 2 -1 3 2 -1 4 -1 -1 -1 1 -1
-1 -1 -1 -1 4 -1 5 -1 -1 2 3 -1 -1 -1 2 -1 -1 -1 2 -1 1 1 -1 2 1 3
-1 5 2 0 -1 0 -1 -1 1 2 -1 -1 2 1 -1 0 -1 1 1 1 -1 2 2 -1 -1 0",
"10 10 24 -1 1 -1 2 -1 2 -1 2 2 1 0 1 -1 3 -1 -1 -1 -1 4 -1 -1 1 1
2 -1 -1 -1 -1 -1 2 -1 1 -1 4 4 3 2 -1 -1 -1 0 -1 4 -1 -1 -1 1 -1 0
0 0 1 -1 -1 -1 4 3 3 -1 1 -1 2 -1 3 -1 3 -1 -1 -1 1 1 -1 1 -1 -1 -
1 4 -1 3 1 -1 -1 2 1 2 -1 3 -1 -1 0 -1 2 -1 1 1 -1 2 1 -1 0"
]
```

```
# In[7]:
```

```
class Node:
    def __init__(self, remain=None, UB=None, LB=None, VAR=None,
DOM=None):
```

```
        #2D list
        self.remain = None
        self.TYPE = None
        self.VAL = None
        self.DOM = None

    # forward calculations
    # will calclulate UpperBound and LowerBound and change Domain
    def forward_domain(self):
        newDOM = [[[] for j in range(m)] for i in range(n)]
        takemin = [[False for j in range(m)] for i in range(n)]
        takemax = [[False for j in range(m)] for i in range(n)]
        cnt = 0
        for x in range(n):
            for y in range(m):
                if(self.TYPE[x][y] == UNKNOWN):
                    cnt += 1
                if(self.TYPE[x][y] == CONSTRAINT):
                    lb = 0
                    rb = 0
                    for dx in range(-1, 2, 1):
                        for dy in range(-1, 2, 1):
                            nx, ny = x + dx, y + dy
                            if((dx == 0 and dy == 0) or not
inrange(nx,ny)):
                                continue
                            if (self.TYPE[nx][ny] == UNKNOWN):
                                if(len(self.DOM[nx][ny]) == 0):
                                    #print("should not print this")
                                    return False
                                lb += min(self.DOM[nx][ny])
                                rb += max(self.DOM[nx][ny])
                            if (self.TYPE[nx][ny] == ASSIGNED):
                                lb += self.VAL[nx][ny]
                                rb += self.VAL[nx][ny]
                    if(lb > C[x][y] or rb < C[x][y]):
                        #print("bound error@" + str(x) + " " +
str(y))
                        #printA(self.TYPE)
```

```
                        #printA(self.VAL)
                        #print(C[x][y], lb, rb)
                        return False
                    if(lb == C[x][y]):
                        for dx in range(-1, 2, 1):
                            for dy in range(-1, 2, 1):
                                nx, ny = x + dx, y + dy
                                if((dx == 0 and dy == 0) or not
inrange(nx,ny)):
                                    continue
                                if(self.TYPE[nx][ny] == UNKNOWN):
                                    takemin[nx][ny] = True
                    if(rb == C[x][y]):
                        for dx in range(-1, 2, 1):
                            for dy in range(-1, 2, 1):
                                nx, ny = x + dx, y + dy
                                if((dx == 0 and dy == 0) or not
inrange(nx,ny)):
                                    continue
                                if (self.TYPE[nx][ny] == UNKNOWN):
                                    takemax[nx][ny] = True
        for i in range(n):
            for j in range(m):
                if(self.TYPE[i][j] == UNKNOWN and
len(self.DOM[i][j]) == 2):
                    if(takemin[i][j] and takemax[i][j]):
                        return False
                    elif(takemin[i][j]):
                        self.DOM[i][j] = [min(self.DOM[i][j])]
                    elif(takemax[i][j]):
                        self.DOM[i][j] = [max(self.DOM[i][j])]
                    else:
                        pass
        if(self.remain > cnt or self.remain < 0):
            return False
        return True


    # use UB-LB or can use DOM len
```

```
    def calc_LCV(self, x, y):
        ret = 2
        for dx in range(-2, 3, 1):
            for dy in range(-2, 3, 1):
                nx, ny = x + dx, y + dy
                if((not (dx == 0 and dy == 0)) and inrange(nx,ny)
and self.TYPE[nx][ny] == UNKNOWN):
                    ret = min(ret, len(self.DOM[nx][ny]))
        return ret

    # return - size of domain, because we want heuristic to be
bigger => better
    def calc_MRV(self, x, y):
        return -len(self.DOM[x][y])

    #how many (valuabe) constraint are besides it
    def calc_DEG(self, x, y):
        ret = 0
        for dx in range(-1, 2, 1):
            for dy in range(-1, 2, 1):
                nx, ny = x + dx, y + dy
                if((not (dx == 0 and dy == 0)) and inrange(nx,ny)
and self.TYPE[nx][ny] == CONSTRAINT):
                    ret += 1
        return ret

    def check_AC(self):
        if(self.remain != 0): # bug, mistype == 0
            return False
        for i in range(n):
            for j in range(m):
                if(self.TYPE[i][j] == UNKNOWN):
                    return False
                if(self.TYPE[i][j] == CONSTRAINT):
                    cnt = 0
                    for dx in range(-1, 2, 1):
                        for dy in range(-1, 2, 1):
                            nx, ny = i + dx, j + dy
```

```
                            if((not (dx == 0 and dy == 0)) and
inrange(nx,ny) and self.TYPE[nx][ny] == ASSIGNED):
                                cnt += self.VAL[nx][ny]
                    if cnt != C[i][j]:
                        return False
        return True

    # return a Node with value updated after assign
    def assign(self, xy, val):
        x, y = xy
        ret = deepcopy(self)
        ret.remain -= val
        ret.TYPE[x][y] = ASSIGNED
        ret.VAL[x][y] = val
        return ret


# In[8]:


def solve(id = 0, input_tc=None):
    global n, m, n_mine, C, A
    # input
    if id == -1:
        all_input = list(map(int,input_tc.strip().split()))
    else:
        all_input = list(map(int,test_data[id].strip().split()))

    #apply input to variables
    n, m, n_mine = all_input[:3]
    flat_constraint = all_input[3:]
    assert(len(flat_constraint) == n*m)
    #print(n,m, len(flat_constraint))
    C = np.asarray([ [ flat_constraint[i*n+j] for j in range(m)]
for i in range(n)])
    A = np.asarray([[ -1 if (C[i][j] == -1) else -2 for j in
range(m)] for i in range(n)])

    #empty stack, init statistics
```

```
    node_popped = 0
    node_pushed = 0
    node_cut = 0
    start_t = time()
    time_elasped = 0
    fail_leaf = 0
    solution = None

    stack = list()
    init_node = Node()
    init_node.remain = n_mine
    init_node.TYPE = [[ UNKNOWN if (C[i][j] == -1) else CONSTRAINT
for j in range(m)] for i in range(n)]
    init_node.VAL = [[ None for j in range(m)] for i in range(n)]
    init_node.DOM = [[[MINE, NOMINE] for j in range(m)] for i in
range(n)]

    if(FORWARD and not init_node.forward_domain()):
        return None
    stack.append(init_node)
    node_pushed += 1

    while(len(stack) != 0):
        cur = stack[-1]
        stack.pop(-1)
        node_popped += 1

        # debug msg
        if(node_popped % 100 == 0):
            #print("Node pushed", node_pushed)
            #print("Node popped", node_popped)
            #printA(cur.VAL)
            time_passed = time() - start_t
            if( time_passed > TIMELIMIT):
                print("TLE with time limit {}".format(TIMELIMIT))
                break

        ## add variable todo
        todo = []
```

```
        for i in range(n):
            for j in range(m):
                if(cur.TYPE[i][j] == UNKNOWN):
                    mrv = cur.calc_MRV(i, j) if MRV else 0
                    deg = cur.calc_DEG(i, j) if DEG else 0
                    todo.append((mrv, deg, (i,j)))

        # done assignment
        if(len(todo) == 0):
            if(not cur.check_AC()):
                print(FAIL_MSG)
                fail_leaf += 1
                continue
            else:
                solution = cur
                break

        # Start Assignments by heuristic, first MVC, then DEG
        #todo.sort(reverse=True)
        todo.sort(reverse=False)
        #False with MRV DEG is quicker =>
        #desireto too see negative MRV nigger first and degree
bigger first
        topush = []
        FAILA = False
        for mcv_h, deg_h, (nx, ny) in todo:
            # LCV heuristic
            # the one with greater dimension of freedom explore
first
            nxt = []
            for val in cur.DOM[nx][ny]: # 0, 1
                nxti = cur.assign((nx, ny), val)
                flag = nxti.forward_domain()
                if(FORWARD and not flag):
                    node_cut += 1*pow(2, len(todo)-1)
                    continue
                if(LCV):
                     h = nxti.calc_LCV(nx, ny) # constant of
25cells
```

```
                elif(GLOBALH):
                    h = int((val == MINE and len(todo)/2 <
cur.remain) or (val == NOMINE and len(todo)/2 >= cur.remain) )
                else :
                    h = 0
                node_pushed += 1
                nxt.append((nxti, h))

            if(CUTA):
                if(len(nxt) == 0):
                    FAILA = True
                    #printA(cur.DOM)
                    #print(nx, ny)
                    break
            #nxt.sort(key=lambda x :x[1], reverse=True) # try
smaller LSV, earlier pop
            nxt.sort(key=lambda x :x[1], reverse=False) # bigger
LCV, later insertion, earlier exlporation
            for nxtnode, h_ in nxt:
                topush.append(nxtnode)

        if(CUTA and FAILA):
            node_pushed -= len(topush)
            node_cut += len(topush)*pow(2,len(todo)-1)
            continue
        else:
            stack.extend(topush)

    #end of CSP solver
    end_t = time()
    time_elasped = end_t - start_t

    #print(solution.VAR)
    #if(time_elasped >= TIMELIMIT):
    #    return None, 1000, 1000, 0, TIMELIMIT
    return solution, node_pushed, node_popped, node_cut,
time_elasped
```

```
# In[9]:


def print_statistics(ANS, node_pushed, node_popped, node_cut,
time_elasped):
    global n, m,A, C
    print("Node pushed", node_pushed)
    print("Node popped", node_popped)
    print("time elasped", time_elasped)
    print("Node cut", node_cut)
    printA(C)
    print()

    if(time_elasped < TIMELIMIT):
        # combine rule and result
        final_board = [
            [ str(C[i][j]) if A[i][j]==-2 else ".x"[ANS.VAL[i][j]]
for j in range(m)] for i in range(n)
        ]

        # print with color
        for i in range(n):
            for j in range(m):
                print("%s"%(fg(1)) if A[i][j] != -2 else "",
final_board[i][j], "%s"%(attr(0))if A[i][j] != -2 else "", end=' ')
            print()


# In[10]:


solution, node_pushed, node_popped, node_cut, time_elasped  =
solve(3)
print_statistics(solution, node_pushed, node_popped, node_cut,
time_elasped)
```

```
# In[11]:


tests = [
    [6, 6, 10, 100],
    [6, 6, 20, 100],
    [6, 6, 30, 100],
    [10, 10, 40, 100],
    [10, 10, 50, 100],
]

def get_file_name(wrapper):
    n, m, v, tcn = wrapper
    return "testcase_{}_{}_{}_{}.txt".format(n,m,v,tcn), 
"stat_{}_{}_{}_{}.txt".format(n,m,v,tcn)


# In[12]:


#get_file_name(tests[0])


# In[13]:


def exp(wrapper):
    n, m, v, tcn = wrapper
    inf, outf = get_file_name(wrapper)
    attr = ["popped", "pushed", "cut", "time_elasped"]
    stat = {}
    for s in attr:
        stat[s] = []
    with open(inf, "r") as fo:
        tcs = fo.readlines()
        #print(tcs[0])
        for i in range(tcn):
            solution, node_pushed, node_popped, node_cut, 
time_elasped  = solve(-1, tcs[i])
```

```
            stat["pushed"].append(node_pushed)
            stat["popped"].append(node_popped)
            stat["cut"].append(node_cut)
            stat["time_elasped"].append(time_elasped)
            #print_statistics(solution, node_pushed, node_popped,
time_elasped)
    for s in attr:
        stat[s].sort
    with open(outf, "w") as fo:
        for s in attr:
            fo.write("Max_{}: {}\n".format(s, sum(stat[s])/tcn))
            fo.write("MID_{}: {}\n".format(s, stat[s][tcn//2]))
            fo.write("AVG_{}: {}\n".format(s, stat[s][-1]))
            print("AVG_{}: {}".format(s, sum(stat[s])/tcn))
            print("MID_{}: {}".format(s, stat[s][tcn//2]))
            print("MAX_{}: {}".format(s, stat[s][-1]))


# In[14]:


#exp(tests[0])


# In[15]:


#for test in tests:
#    exp(test)


# In[20]:


#here stores hyper params
TIMELIMIT = 3 #second

FORWARD = True
# in which order should we explore variables
```

```
# MRV, domain smaller
# DEG, vertex that affect more points
MRV = 1
DEG = 0
# limitation added if assign this value to the variable
LCV = 0
GLOBALH = 0
# If a variable can not be assign, do not need to try sidlings
CUTA = 0
```

```
# In[21]:
```

```
exp([6,6,18,100])
```

```
# In[26]:
```

```
solve(5)
```

```
# In[27]:
```

```
solve(-1, "15 15 19 0 0 1 -1 1 0 0 0 0 0 0 0 1 1 1 0 0 1 1 1 0 0 0
0 0 0 0 1 -1 1 1 1 0 0 0 0 0 0 0 0 0 0 1 1 1 -1 -1 0 0 0 1 2 2 1 0
0 0 0 0 0 0 1 1 -1 0 0 1 -1 -1 1 0 0 0 -1 1 1 0 0 0 1 1 2 2 2 1 0 1
1 2 -1 1 1 1 0 1 -1 1 0 0 0 0 1 -1 2 1 1 -1 -1 -1 1 1 1 1 -1 1 0 1
1 1 0 0 3 3 1 0 1 1 2 -1 2 1 1 0 -1 0 0 -1 -1 1 0 1 -1 2 1 2 -1 1
0 0 0 0 2 -1 1 0 1 1 2 1 2 1 1 0 0 0 0 -1 0 0 0 0 0 1 -1 1 0 1 1 1
1 1 0 0 0 0 0 1 2 2 1 0 1 -1 2 -1 -1 0 0 0 0 0 1 -1 1 0 0 1 1 2 -1
2 0 -1 0 0 0 1 1 1 0 0 0 0 1 1 1")
```

```
# In[28]:
```

```
solve(-1, "15 15 26 0 -1 0 0 0 0 0 0 0 1 1 1 0 -1 -1 0 0 -1 1 1 0
-1 0 0 1 -1 1 0 0 0 1 1 1 -1 1 0 0 -1 0 1 1 1 0 0 0 -1 -1 1 2 2 1
1 1 1 0 0 0 0 0 0 1 1 0 1 -1 -1 2 -1 -1 0 0 0 0 0 0 0 -1 0 1 1 1 2
-1 -1 1 0 0 0 0 0 0 0 0 0 0 -1 1 3 -1 4 2 1 0 0 0 0 0 1 1 1 0 -1 2
-1 -1 -1 1 0 1 -1 0 1 2 -1 1 -1 0 1 2 3 2 2 2 3 -1 0 1 -1 2 1 -1 0
0 0 0 -1 2 -1 -1 2 1 2 2 1 0 -1 1 1 0 0 1 -1 3 -1 1 1 -1 -1 0 1 2
-1 -1 0 0 1 1 2 1 1 1 -1 1 0 1 -1 -1 2 1 0 0 -1 1 -1 1 0 1 2 2 2 1
2 -1 1 1 2 2 2 2 2 0 1 -1 -1 1 0 1 1 1 1 -1 -1 1 1 -1")
```

```
# In[30]:
```

```
solve(-1, "6 6 19 -1 -1 -1 -1 -1 -1 -1 4 -1 -1 -1 2 -1 6 -1 6 -1 -
1 -1 -1 -1 -1 -1 -1 -1 -1 -1 4 -1 -1 2 -1 -1 -1 -1 -1")
```

```
# In[31]:
```

```
solve(-1, "10 10 8 1 2 -1 1 0 0 0 0 -1 0 1 -1 2 -1 0 0 0 0 -1 0 1
1 1 0 0 -1 0 0 0 0 -1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 -1 1 1 1 0 0 0 0 0 0 1 3 -1 3 -1 -1 1 -1 0 -1 1 -1
-1 -1 1 1 -1 2 1 0 -1 2 3 2 -1 1 2 -1 1")
```

```
# In[32]:
```

```
solve(-1, "10 10 18 1 -1 1 0 1 1 1 0 0 -1 2 2 1 0 2 -1 2 0 1 1 -1
-1 1 1 3 -1 -1 2 3 -1 1 1 1 -1 2 -1 -1 -1 3 -1 -1 0 1 1 -1 2 3 -1
3 2 0 0 0 0 1 -1 1 -1 1 -1 0 0 0 -1 1 2 2 2 3 3 1 -1 0 0 -1 1 -1 2
-1 -1 -1 2 0 0 0 -1 2 4 4 3 -1 2 0 0 0 0 1 -1 -1 1")
```

```
# In[33]:
```

```
solve(-1, "15 15 12 1 1 0 0 0 0 0 0 0 0 0 0 0 0 -1 1 0 0 0 -1 0
0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 -1
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 1 1 1 0 0 0 0 0 1 -1 1
0 0 1 1 2 -1 1 0 0 0 0 1 -1 2 1 0 0 1 -1 2 1 1 0 0 0 0 1 -1 1 0 0
0 1 1 1 0 0 0 0 1 1 2 1 1 0 0 0 0 0 0 1 1 1 0 1 -1 1 0 1 1 1 0 0 0
0 1 -1 1 0 1 1 1 0 2 -1 2 0 0 0 0 1 -1 1 0 0 0 0 0 2 -1 2 0 0 0 1
1 1 0 0 0 0 0 0 1 1 1 0 0 0 1 -1 3 2 1 0 0 0 0 0 0 0 -1 0 0 1 2 -1
-1 -1 0 0 0 0 -1 0 -1 0")


# In[ ]:
```

2. testcase_generator.py

```
# author = yt lin
# usage = python3 testcase_generator.py < 6 6 10 100

import os
import numpy as np
import random

## hyper params

n = 6
m = 6
nv = 10
n_mine = 0

## use input
n, m, nv, N = list(map(int,input().strip().split()))

## init all positions
all_coordinate = []
for i in range(n):
    for j in range(m):
        all_coordinate.append((i,j))
```

```
def inrange(i, j):
    return i >= 0 and i < n and j >= 0 and j < m

def gen(n, m, nv):
    n_mine = 0
    ## pick variable positions
    POSV = random.sample(all_coordinate, nv)
    MARK = [[ 0 for j in range(m)] for i in range(n)]
    for pos in POSV:
        MARK[pos[0]][pos[1]] = 1
    # variables
    A = [[ random.choice([0,1]) if MARK[i][j] else -1 for j in
range(m)] for i in range(n)]
    for i in range(n):
        for j in range(m):
            n_mine += int(A[i][j] == 1)
    # constraints init
    B = [[ 0 for j in range(m)] for i in range(n)]
    # calc constraints
    for i in range(n):
        for j in range(m):
            if A[i][j] == -1:
                for di in range(-1, 2, 1):
                    for dj in range(-1, 2, 1):
                        if(not (di == dj and di == 0) and
inrange(i+di, j+dj)):
                                B[i][j] += int(A[i+di][j+dj] == 1)
    TC = [[ B[i][j] if A[i][j] == -1 else -1  for j in range(m)]
for i in range(n)]
    SAMPLE_ANS = [[ B[i][j] if A[i][j] == -1 else "ox"[A[i][j]]
for j in range(m)] for i in range(n)]
    return n, m, n_mine, TC, SAMPLE_ANS


def create_tc_file(n, m, nvar, ntc):
    with open("testcase_" + str(n) + "_" +  str(m) + "_" +
str(nvar) + "_" + str(ntc) + ".txt", "w") as fo:
        for i in range(ntc):
            n, m, n_mine, TC, SAMPLE_ANS = gen(m, m, nvar)
```

```
            TC_str = str(n) + " " +  str(m) + " " + str(n_mine)
            for i in range(n):
                for j in range(m):
                    TC_str += " " + str(TC[i][j])
            fo.write(TC_str)
            fo.write("\n")

create_tc_file(n, m, nv, N)
```