MINI PROJECT – FUNDAMENTALS OF OPTIMIZATION

SEMESTER EXAM SCHEDULE

Group 18

1. *Tran Thanh Truong – 20214938*
2. *Hoang Dinh Dung – 20214882*
3. *Phan Cong Anh – 20210078*

TABLE OF CONTENTS

1. **INTRODUCTION AND DESCRIBTION**

2. **MODELLING ALGORITHMS IMPLEMENTATION**

   - *MIXED INTEGER PROGRAMMING*
   - *CONSTRAINT PROGRAMMING*
   - *BACKTRACKING*
   - *HEURISTIC 1*
   - *HEURISTIC 2*

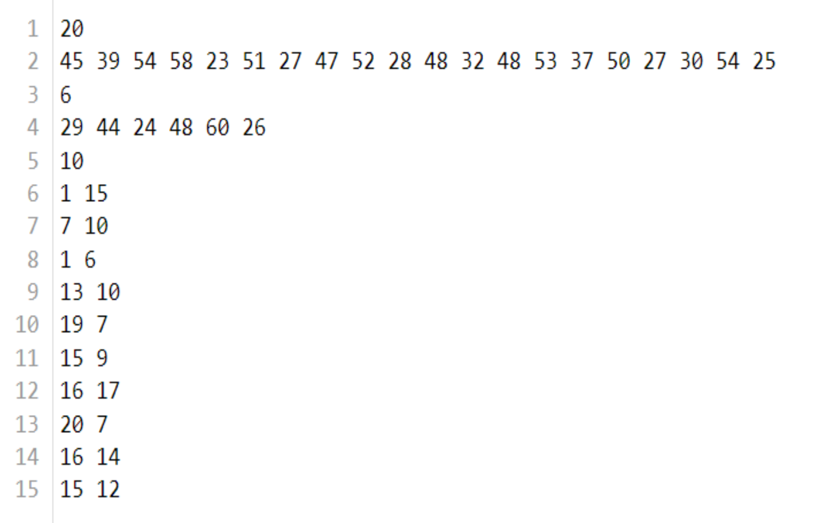
4. **ANALYSIS AND CONCLUSION**

1. INTRODUCTION AND DESCRIBTION

Scheduling semester exam is one of the most popular task at school. To help teachers do this work more conveniently, we give some algorithms that will be explained in the next section to optimize this problem.

PROBLEM DESCRIBTION

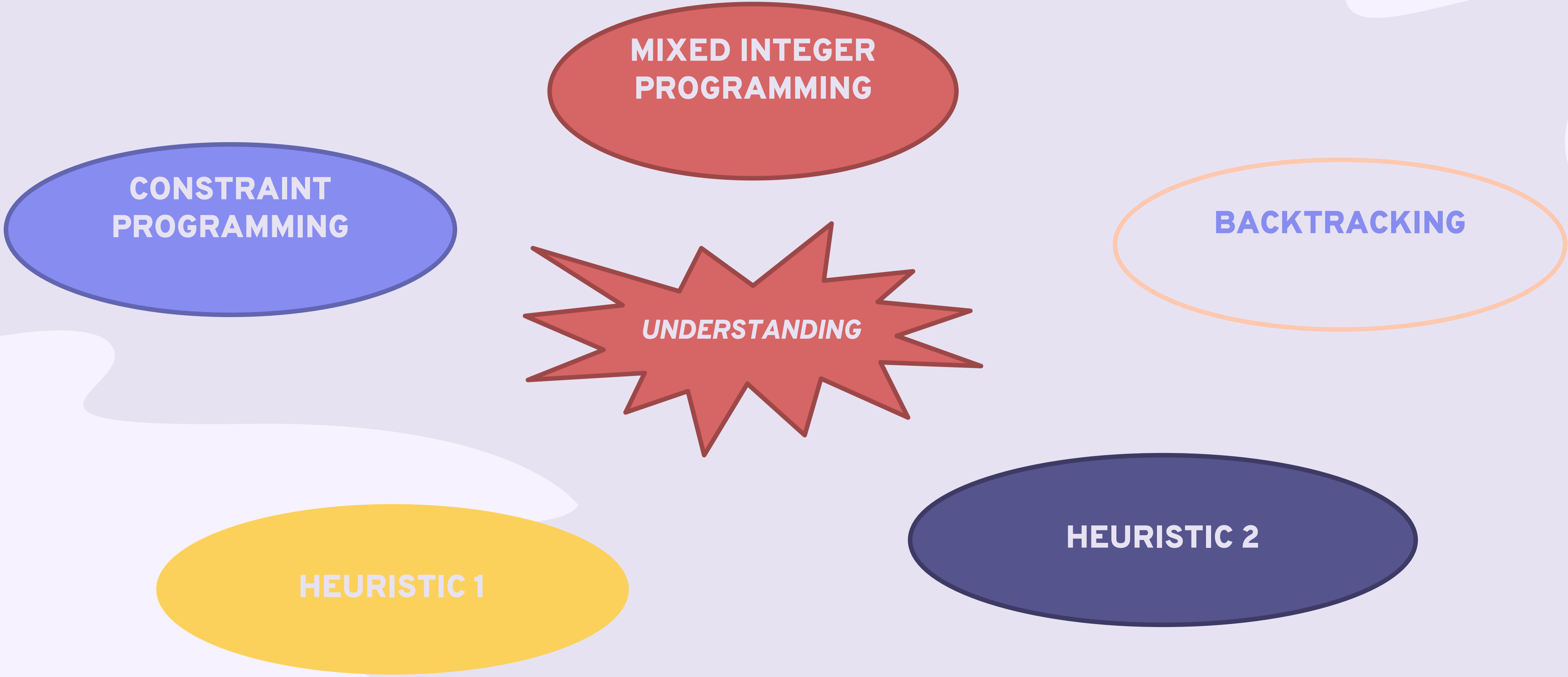
- N: Number of courses to be scheduled
- d(i): Number of students participate in course n(i)
- M: Number of rooms
- c(j): Number of seats of room m(j)
- K: Number of of pairs of courses cannot be grouped together (conflicting courses)
- Pairs of conflicting courses



```
20
45 39 54 58 23 51 27 47 52 28 48 32 48 53 37 50 27 30 54 25
6
29 44 24 48 60 26
10
1 15
7 10
1 6
13 10
19 7
15 9
16 17
20 7
16 14
15 12
```

We have four periods in a day and the problem is minimizing the number of days scheduled for examination.

2. MODELLING AND ALGORITHMS IMPLEMENTATION



MIXED INTEGER PROGRAMMING

CONSTRAINT PROGRAMMING

BACKTRACKING

UNDERSTANDING

HEURISTIC 1

HEURISTIC 2

• *MIXED INTEGER PROGRAMMING*

1. Define variables

- $X[i][j][k]$: variable that put course n(i) to room m(j) at period k

  $$0 \le i, k \le N, 0 \le j \le M, X[i][j][k] \in \{0; 1\}$$

- y: number of periods

- C: list of conflicting courses

2. Constraints

- **Constraint 1: Pairs of conflicting courses cannot be put in the same period**

$$0 \le X[i1][j][k] + X[i2][j][k] \le 1, \forall (i1, i2) \in C;\ j = 0, \dots, M-1; k = 0, \dots, N-1$$

- **Constraint 2: An course room be putted at most one course in a period**

$$0 \le \sum_{i=0}^{N-1} X[i][j][k] \le 1, \forall j = 0, \dots, M-1; k = 0, \dots, N-1$$

- **Constraint 3: The number of periods**

$$-\infty \le k \times X[i][j][k] - y \le 0, \forall j = 0, \dots, M-1; k = 0, \dots, N-1$$

- **Constraint 4: A course be conducted at most one time in an room**

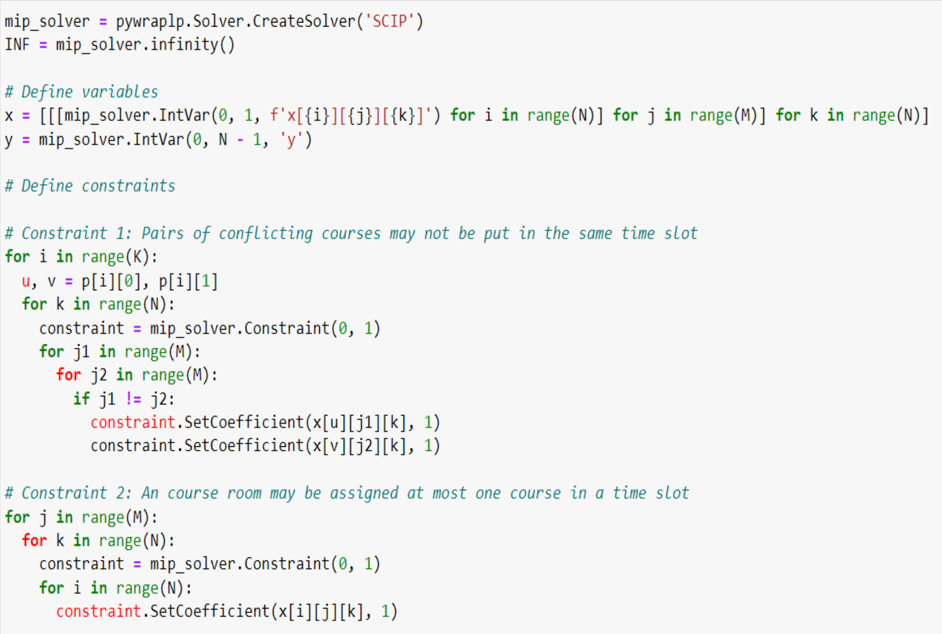
$$0 \le \sum_{j=0}^{M-1} \sum_{k=0}^{N-1} X[i][j][k] \le 1, \forall i = 0, \dots, N-1$$

- **Constraint 5: A course n(i) must be put into a room m(j) with capacity c(j)**

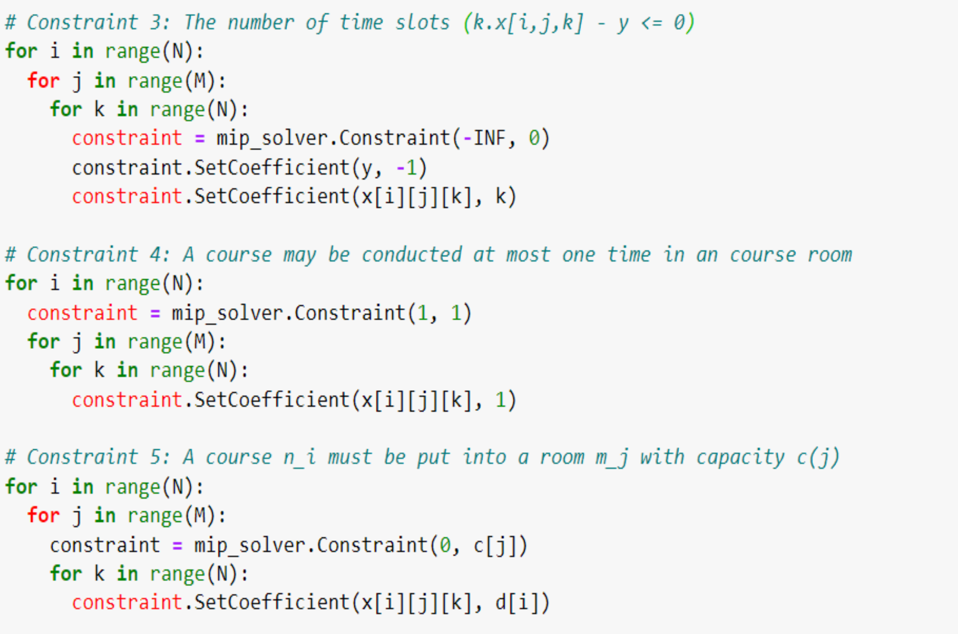
$$0 \le \sum_{k=0}^{N-1} X[i][j][k] \times d[i] \le c[j], \forall i = 0, \dots, N-1; j = 0, \dots, M-1$$

3. Objective

$$\Longrightarrow \boldsymbol{Minimize(y)}$$



```
mip_solver = pywraplp.Solver.CreateSolver('SCIP')
INF = mip_solver.infinity()

# Define variables
x = [[[mip_solver.IntVar(0, 1, f'x[{i}][{j}][{k}]') for i in range(N)] for j in range(M)] for k in range(N)]
y = mip_solver.IntVar(0, N - 1, 'y')

# Define constraints

# Constraint 1: Pairs of conflicting courses may not be put in the same time slot
for i in range(K):
  u, v = p[i][0], p[i][1]
  for k in range(N):
    constraint = mip_solver.Constraint(0, 1)
    for j1 in range(M):
      for j2 in range(M):
        if j1 != j2:
          constraint.SetCoefficient(x[u][j1][k], 1)
          constraint.SetCoefficient(x[v][j2][k], 1)

# Constraint 2: An course room may be assigned at most one course in a time slot
for j in range(M):
  for k in range(N):
    constraint = mip_solver.Constraint(0, 1)
    for i in range(N):
      constraint.SetCoefficient(x[i][j][k], 1)
```



```
# Constraint 3: The number of time slots (k.x[i,j,k] - y <= 0)
for i in range(N):
  for j in range(M):
    for k in range(N):
      constraint = mip_solver.Constraint(-INF, 0)
      constraint.SetCoefficient(y, -1)
      constraint.SetCoefficient(x[i][j][k], k)

# Constraint 4: A course may be conducted at most one time in an course room
for i in range(N):
  constraint = mip_solver.Constraint(1, 1)
  for j in range(M):
    for k in range(N):
      constraint.SetCoefficient(x[i][j][k], 1)

# Constraint 5: A course n_i must be put into a room m_j with capacity c(j)
for i in range(N):
  for j in range(M):
    constraint = mip_solver.Constraint(0, c[j])
    for k in range(N):
      constraint.SetCoefficient(x[i][j][k], d[i])
```

• CONSTRAINT PROGAMMING

1. Define variables

- $X[i]$: period of course n(i)

$$i \in \{1, 2, \dots, n\}, X[i] \in \{1, 2, \dots, n\}$$

- $Y[i][j]$: course n(i) be putted in room m(j)

$$i \in \{1, 2, \dots, N\}, j \in \{1, 2, \dots, M\}, Y[i][j] \in \{0; 1\}$$

- m: number of periods that need for scheduling exam
- p: list of pairs conflicting courses

2. Constraint

- **Constraint 1: Pairs of conflicting courses may not be put in the same period**

$$X[i] \neq X[j], \forall (i,j) \in p$$

- **Constraint 2: An course room is assigned at most one course in a period**

$$\sum_{j=1}^{M} Y[i][j] = 1, \forall i = 1, \dots, N$$

- **Constraint 3: Courses with same period cannot use the same room**

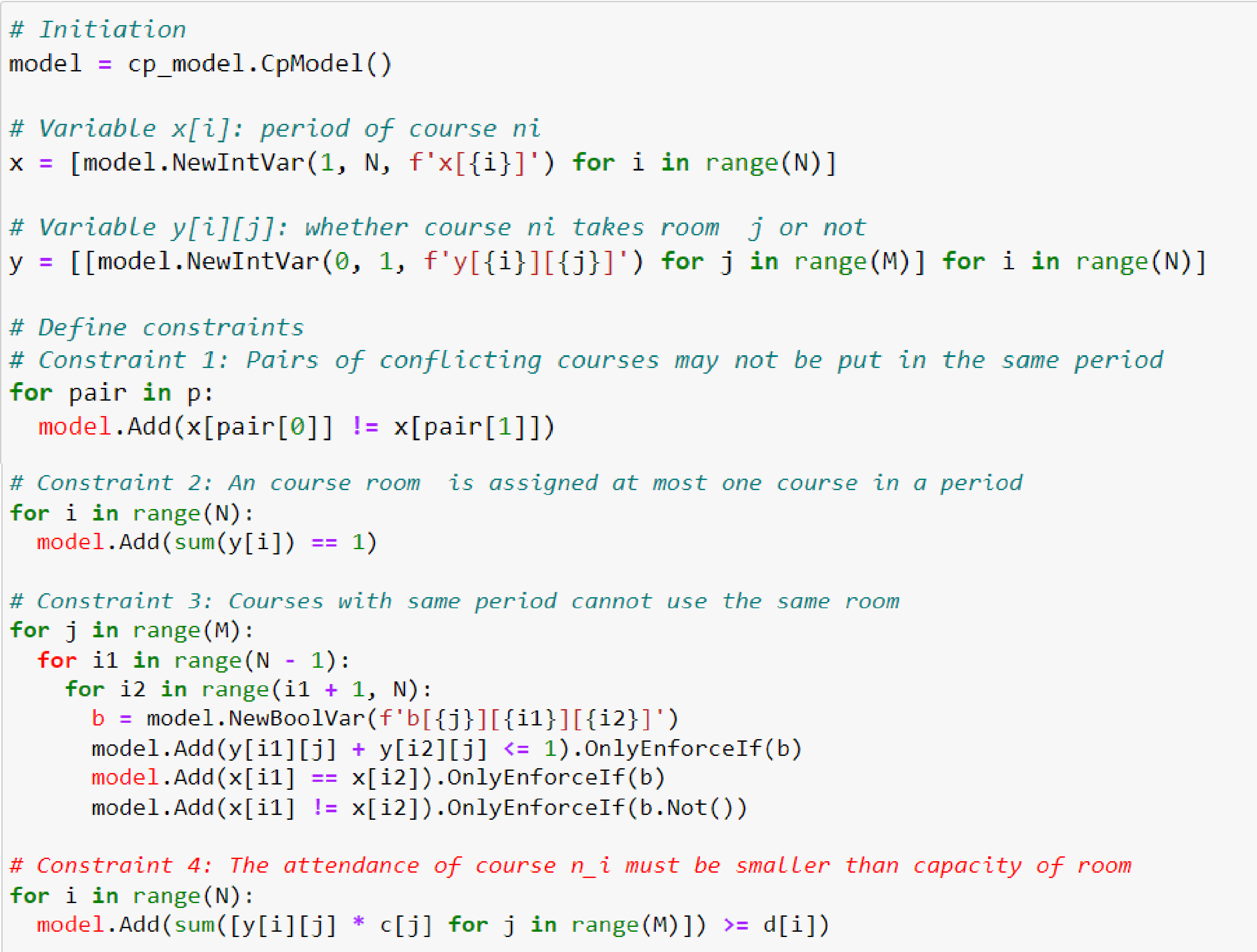
$$X[i] = X[j] \Rightarrow Y[i][k] + Y[j][k] \leq 1, \forall i, j \in \{1, \dots, N\}; k \in \{1, \dots, M\}$$

- **Constraint 4: The attendance of course n(i) must be smaller than capacity of room**

$$d[i] \leq \sum_{j=1}^{M} Y[i][j] \times c[j], \forall i = 1, \dots, N$$

3. Objective

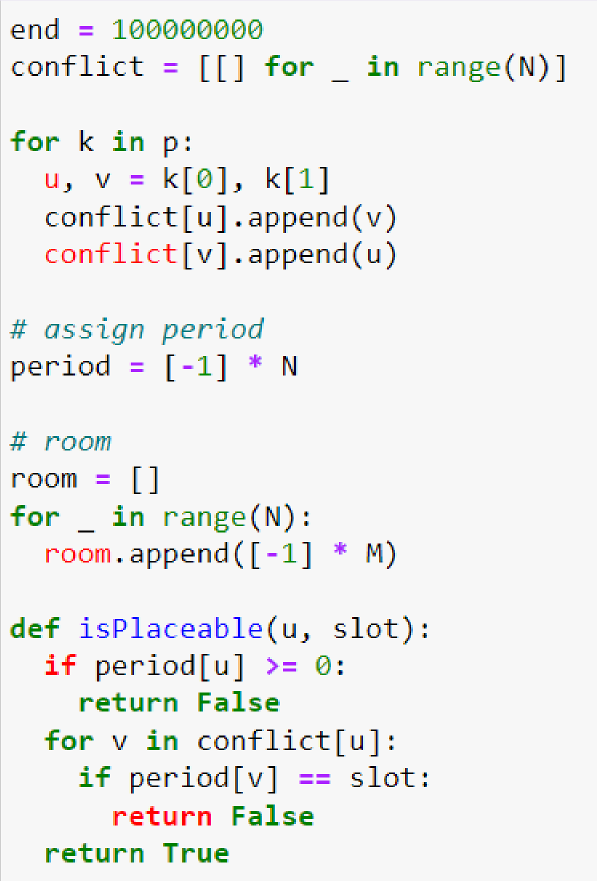
$\Longrightarrow Minimize(m)$ **with** $m = max(X)$



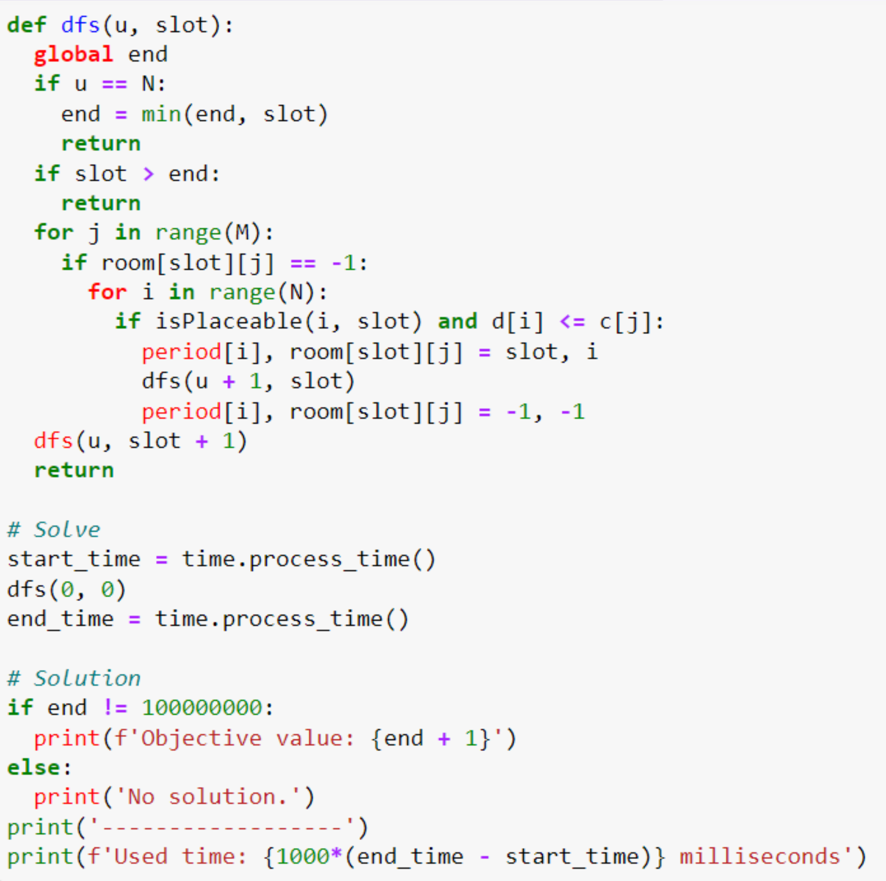
```python
# Initiation
model = cp_model.CpModel()

# Variable x[i]: period of course ni
x = [model.NewIntVar(1, N, f'x[{i}]') for i in range(N)]

# Variable y[i][j]: whether course ni takes room  j or not
y = [[model.NewIntVar(0, 1, f'y[{i}][{j}]') for j in range(M)] for i in range(N)]

# Define constraints
# Constraint 1: Pairs of conflicting courses may not be put in the same period
for pair in p:
  model.Add(x[pair[0]] != x[pair[1]])

# Constraint 2: An course room  is assigned at most one course in a period
for i in range(N):
  model.Add(sum(y[i]) == 1)

# Constraint 3: Courses with same period cannot use the same room
for j in range(M):
  for i1 in range(N - 1):
    for i2 in range(i1 + 1, N):
      b = model.NewBoolVar(f'b[{j}][{i1}][{i2}]')
      model.Add(y[i1][j] + y[i2][j] <= 1).OnlyEnforceIf(b)
      model.Add(x[i1] == x[i2]).OnlyEnforceIf(b)
      model.Add(x[i1] != x[i2]).OnlyEnforceIf(b.Not())

# Constraint 4: The attendance of course n_i must be smaller than capacity of room
for i in range(N):
  model.Add(sum([y[i][j] * c[j] for j in range(M)]) >= d[i])
```

- ***BACKTRACKING***

```
end := a very very large number
define function: dfs(u,slot):
  if u == N:
    end = min(slot, end)
    return
  if slot > end:
    return
  for each room:
    if room is free:
      for each course:
        if course can be putted and attendance <= capacity:
          put course to that room
          put course to that slot
          dfs(u+1,slot)
          free that room
          free that slot
  dfs(u,slot+1)
  return
```



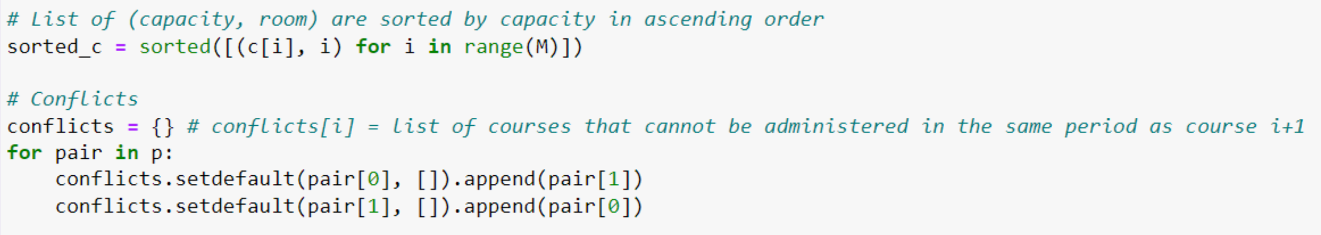
```
end = 100000000
conflict = [[] for _ in range(N)]

for k in p:
  u, v = k[0], k[1]
  conflict[u].append(v)
  conflict[v].append(u)

# assign period
period = [-1] * N

# room
room = []
for _ in range(N):
  room.append([-1] * M)

def isPlaceable(u, slot):
  if period[u] >= 0:
    return False
  for v in conflict[u]:
    if period[v] == slot:
      return False
  return True
```



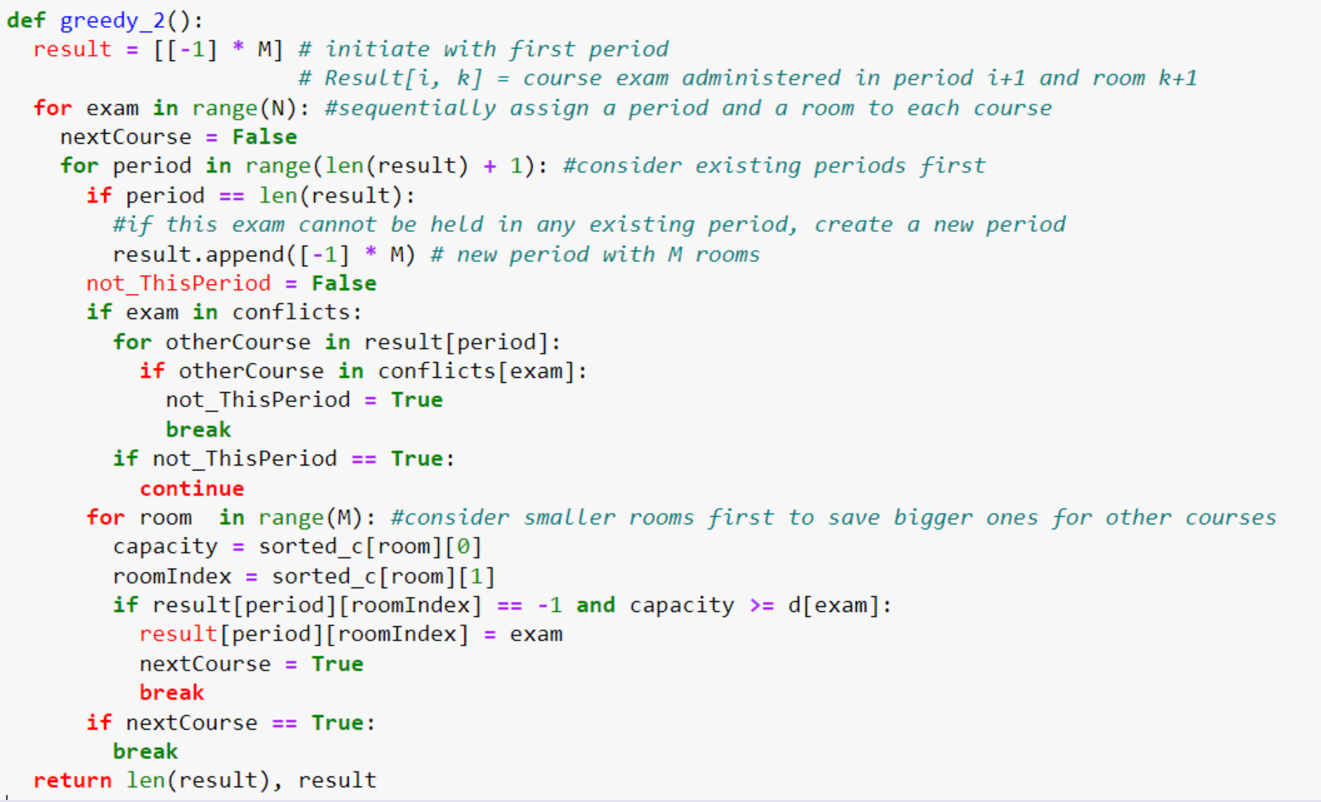
```
def dfs(u, slot):
  global end
  if u == N:
    end = min(end, slot)
    return
  if slot > end:
    return
  for j in range(M):
    if room[slot][j] == -1:
      for i in range(N):
        if isPlaceable(i, slot) and d[i] <= c[j]:
          period[i], room[slot][j] = slot, i
          dfs(u + 1, slot)
          period[i], room[slot][j] = -1, -1
  dfs(u, slot + 1)
  return

# Solve
start_time = time.process_time()
dfs(0, 0)
end_time = time.process_time()

# Solution
if end != 100000000:
  print(f'Objective value: {end + 1}')
else:
  print('No solution.')
print('------------------')
print(f'Used time: {1000*(end_time - start_time)} milliseconds')
```

• HEURISTIC 1

```
sort list of (capacity,room) in ascending order of capacity
for each course:
  for each period:
    if course test cannot be putted in any period:
      add new period
    if two course conflict in same period:
      consider the next period
    else:
      for each sorted capacity:
        if attendance <= capacity and at that room and period have no test:
          put course to that room and period
          consider the next course
```



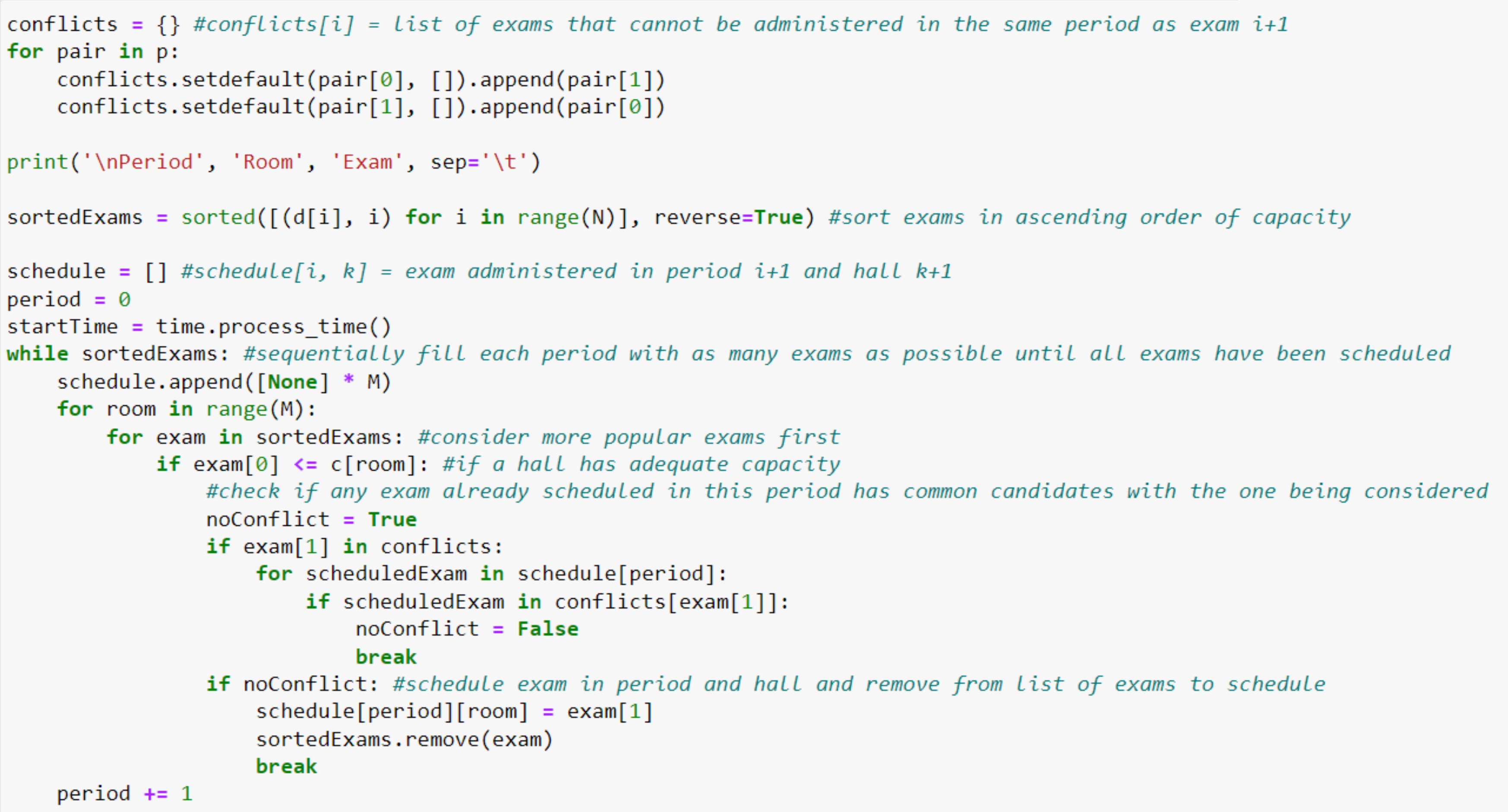
```python
# List of (capacity, room) are sorted by capacity in ascending order
sorted_c = sorted([(c[i], i) for i in range(M)])

# Conflicts
conflicts = {} # conflicts[i] = list of courses that cannot be administered in the same period as course i+1
for pair in p:
    conflicts.setdefault(pair[0], []).append(pair[1])
    conflicts.setdefault(pair[1], []).append(pair[0])
```



```python
def greedy_2():
  result = [[-1] * M] # initiate with first period
                      # Result[i, k] = course exam administered in period i+1 and room k+1
  for exam in range(N): #sequentially assign a period and a room to each course
    nextCourse = False
    for period in range(len(result) + 1): #consider existing periods first
      if period == len(result):
        #if this exam cannot be held in any existing period, create a new period
        result.append([-1] * M) # new period with M rooms
      not_ThisPeriod = False
      if exam in conflicts:
        for otherCourse in result[period]:
          if otherCourse in conflicts[exam]:
            not_ThisPeriod = True
            break
        if not_ThisPeriod == True:
          continue
      for room  in range(M): #consider smaller rooms first to save bigger ones for other courses
        capacity = sorted_c[room][0]
        roomIndex = sorted_c[room][1]
        if result[period][roomIndex] == -1 and capacity >= d[exam]:
          result[period][roomIndex] = exam
          nextCourse = True
          break
      if nextCourse == True:
        break
  return len(result), result
```

• *HEURISTIC 2*

```
list_of_exam = sorted([(attendant, i)])
while list_of_exam not empty:
    allocate a new period for remaining exams
    for each room:
        for exam in list_of_exam:
            if attendant <= capacity:
                if have no exam scheduled conflicts in this period:
                    put exam in this room and this period
                    consider the next room
```



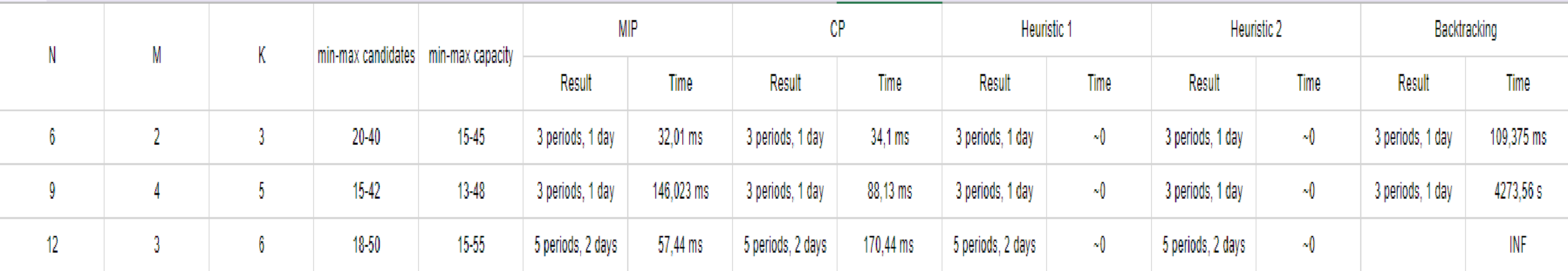
```python
conflicts = {} #conflicts[i] = list of exams that cannot be administered in the same period as exam i+1
for pair in p:
    conflicts.setdefault(pair[0], []).append(pair[1])
    conflicts.setdefault(pair[1], []).append(pair[0])

print('\nPeriod', 'Room', 'Exam', sep='\t')

sortedExams = sorted([(d[i], i) for i in range(N)], reverse=True) #sort exams in ascending order of capacity

schedule = [] #schedule[i, k] = exam administered in period i+1 and hall k+1
period = 0
startTime = time.process_time()
while sortedExams: #sequentially fill each period with as many exams as possible until all exams have been scheduled
    schedule.append([None] * M)
    for room in range(M):
        for exam in sortedExams: #consider more popular exams first
            if exam[0] <= c[room]: #if a hall has adequate capacity
                #check if any exam already scheduled in this period has common candidates with the one being considered
                noConflict = True
                if exam[1] in conflicts:
                    for scheduledExam in schedule[period]:
                        if scheduledExam in conflicts[exam[1]]:
                            noConflict = False
                            break
                if noConflict: #schedule exam in period and hall and remove from list of exams to schedule
                    schedule[period][room] = exam[1]
                    sortedExams.remove(exam)
                    break
    period += 1
```

A PICTURE IS WORTH A THOUSAND WORDS

3. ANALYSIS AND CONCLUSION

OUR RESULTS

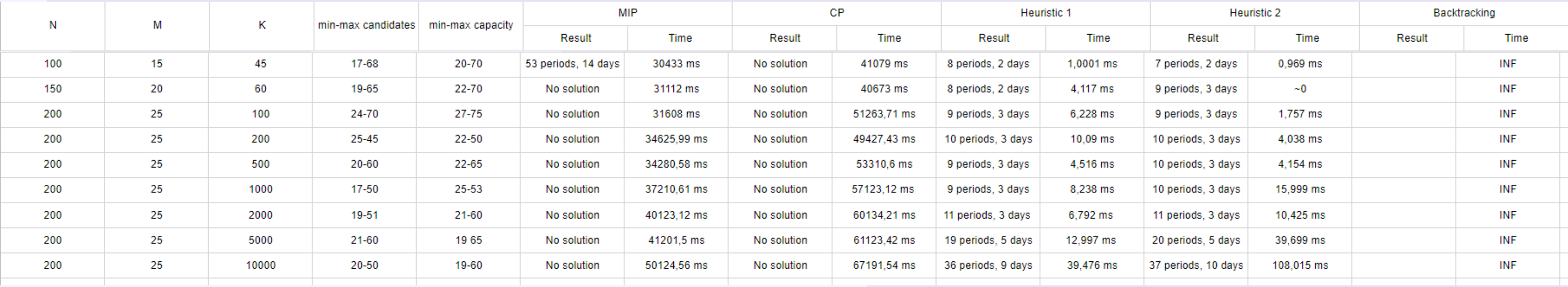
- With some small datasets, MIP, CP and Backtracking work quite well. Heuristic 1 and Heuristic 2 give the same results but they run very fast.



N	M	K	min-max candidates	min-max capacity	MIP		CP		Heuristic 1		Heuristic 2		Backtracking	
					Result	Time	Result	Time	Result	Time	Result	Time	Result	Time
6	2	3	20-40	15-45	3 periods, 1 day	32,01 ms	3 periods, 1 day	34,1 ms	3 periods, 1 day	~0	3 periods, 1 day	~0	3 periods, 1 day	109,375 ms
9	4	5	15-42	13-48	3 periods, 1 day	146,023 ms	3 periods, 1 day	88,13 ms	3 periods, 1 day	~0	3 periods, 1 day	~0	3 periods, 1 day	4273,56 s
12	3	6	18-50	15-55	5 periods, 2 days	57,44 ms	5 periods, 2 days	170,44 ms	5 periods, 2 days	~0	5 periods, 2 days	~0		INF

OUR RESULTS

- With large datasets, Backtracking algorithm takes infinity time, MIP and CP work ineffectively, while Heuristic 1 and Heuristic 2 solve the problem very well.



N	M	K	min-max candidates	min-max capacity	MIP		CP		Heuristic 1		Heuristic 2		Backtracking	
					Result	Time	Result	Time	Result	Time	Result	Time	Result	Time
100	15	45	17-68	20-70	53 periods, 14 days	30433 ms	No solution	41079 ms	8 periods, 2 days	1,0001 ms	7 periods, 2 days	0,969 ms		INF
150	20	60	19-65	22-70	No solution	31112 ms	No solution	40673 ms	8 periods, 2 days	4,117 ms	9 periods, 3 days	~0		INF
200	25	100	24-70	27-75	No solution	31608 ms	No solution	51263,71 ms	9 periods, 3 days	6,228 ms	9 periods, 3 days	1,757 ms		INF
200	25	200	25-45	22-50	No solution	34625,99 ms	No solution	49427,43 ms	10 periods, 3 days	10,09 ms	10 periods, 3 days	4,038 ms		INF
200	25	500	20-60	22-65	No solution	34280,58 ms	No solution	53310,6 ms	9 periods, 3 days	4,516 ms	10 periods, 3 days	4,154 ms		INF
200	25	1000	17-50	25-53	No solution	37210,61 ms	No solution	57123,12 ms	9 periods, 3 days	8,238 ms	10 periods, 3 days	15,999 ms		INF
200	25	2000	19-51	21-60	No solution	40123,12 ms	No solution	60134,21 ms	11 periods, 3 days	6,792 ms	11 periods, 3 days	10,425 ms		INF
200	25	5000	21-60	19 65	No solution	41201,5 ms	No solution	61123,42 ms	19 periods, 5 days	12,997 ms	20 periods, 5 days	39,699 ms		INF
200	25	10000	20-50	19-60	No solution	50124,56 ms	No solution	67191,54 ms	36 periods, 9 days	39,476 ms	37 periods, 10 days	108,015 ms		INF

AWESOME WORDS