**Algorithm**

**for**

**TimeTable Generator**

**CS-08**

**Indian Institute of Information Technology Vadodara**

Team Members

Aman Yadav (201651007)

DakshKumar Gondaliya (201651014)

Kirtika Singhal (201651024)

Mayank Pathela (201651029)

Nikhil Sachan (201651034)

Parmeshwar Kumawat (201651035)

**Revision History**

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Date** | **Reason For Changes** | **Version** |
| Aman Yadav | 15/10/2018 | Initiate | 1 |
| Mayank Pathela | 25/10/2018 | Reviewed | 1 |

# **Algorithm**

ranD (slots, count) {

let i

let viableDays = []

for i in slots

if (slots[i].length >= count)

viableDays.push(i)

if(viableDays.length == 0)

return null

let buff = crypto.randomBytes(2);

let n = parseInt(buff.toString('hex'),16)

let index = n % (viableDays.length);

let day = viableDays[index]

let slot = []

for(i=0;i<count;i++){

buff = crypto.randomBytes(2);

n = parseInt(buff.toString('hex'),16)

let s = n % slots[day].length;

slot.push(slots[day][s]);

slots[day].splice(s,1);

return day, slot

function generate(instaces, givenSlots, teachers, sections) // Function to generate and

return a vaible timetable

// or return a message to retry

secInstances = {}

TT = []

secTT = {}

teacherTT = {}

numDays = 0

for x in givenSlots

if(givenSlots[x] > 0)

numDays++

for i in sections

for j in instances

for k in instaces[j].sections

if(instaces[j].sections[k] == sections[i])

instaces[j]["mapp"] = []

secInstances[sections[i]].push(instaces[j])

regenerateCountSec = 0

regenerateFlagSec =false

regenerateListSec =

notPossibleCount = 0

impossible = false

for i in sections

if(impossible)

return ("Table Not Possible")

break

notPossible = false

currentTT = [][]

regenerateCountSI = 0

regenerateFlagSI = false

regenerateListSI =

for j in secInstances[sections[i]]

availSlots = []

for day in givenSlots

daySlots = []

for slot in givenSlots[day]

if(regenerateFlagSI)

slotFlag = true

for a in regenerateListSI.slot

dumFlag = false

for b in slot

if(slot[b] ==

regenerateListSI.slot[a])

dumFlag = true

break

if(!dumFlag)

slotFlag = false

Break

if( ( (!slotFlag) || (day != regenerateListSI.day)) && (teacherTT[secInstances[sections[i]][j].teacher][day][slot] == 0) && (currentTT[day][slot] == 0))

daySlots.push(slot)

regenerateFlagSI = false

else if(regenerateFlagSec)

slotFlag = true

for a in regenerateListSI.slot

dumFlag = false

for b in slot

if(slot[b] == regenerateListSI.slot[a])

dumFlag = true

break

if(!dumFlag)

slotFlag = false

break

if(( (!slotFlag) || (day != regenerateListSec.day))

&&(teacherTT[secInstances[sections[i]][j].teacher][day][slot] == 0) &&

(currentTT[day][slot] == 0))

regenerateFlagSec = false

elseif((teacherTT[secInstances[sections[i]][j].teacher][day][slot] == 0) && (currentTT[day][slot] == 0))

daySlots.push(slot)

availSlots.push(daySlots)

eachDay = secInstances[sections[i]][j].numLectures / numDays

extraDays = secInstances[sections[i]][j].numLectures % numDays

for i in range(numDays)

if(extraDays > 0)

count = eachDay + 1

extraDays = extraDays - 1

else

count = eachDay

flag = true

radCount = 0

while(flag)

const ret = ranD(availSlots, count)

if((ret != undefined) && (ret != null) && (ret.day != undefined) && (ret.slot!= undefined) &&

(ret.day >= 0) && (ret.day < givenSlots.length) && (ret.slot.length == count))

secInstances[sections[i]][j].mapp.push(ret.day,ret.slot)

for z in ret.slot

currentTT[ret.day][ret.slot[z]] = secInstances[sections[i]][j] teacherTT[secInstances[sections[i]][j].teacher][ret.day][ret.slot[z]] =

secInstances[sections[i]][j]

availSlots[ret.day] = []

flag = false

else

if(radCount < 10)

radCount = radCount + 1

else if(regenerateCountSI<100)

regenerateSI = true

regenerateCountSI = regenerateCountSI + 1

flag= false

regenerateFlagSI = true

regenerateListSI = secInstances[sections[i]][j].mapp[0]

for y in secInstances[sections[i]][j].mapp

for w in secInstances[sections[i]][j].mapp.slot

currentTT[secInstances[sections[i]][j].mapp[y].day][secInsa

nces[sections[i]][j].mapp[y].slot[w]] = 0

teacherTT[secInstances[sections[i]][j].teacher][secInstance

s[sections[i]][j].mapp[y].day][secInstances[sections[i]][j].ma

pp[y].slot[w]] = 0

secInstances[sections[i]][j].mapp = []

j--

else

if(regenerateCountSec < 100)

regenerateCountSI = 0

regenerateSec = true

regenerateCountSec++

regenerateFlagSec = true

flag = false

regenerateListSec = secInstances[sections[i]][0].mapp[0]

for x in secInstances[sections[i]]

for y in secInstances[sections[i]][x].mapp

for w in secInstances[sections[i]][x].mapp.slot

teacherTT[secInstances[sections[i]][x].teach

er][secInstances[sections[i]][x].mapp[y].day][

secInstances[sections[i]][x].mapp[y].slot[w]]

= 0

for x in secInstances[sections[i]]

secInstances[sections[i]][x].mapp = []

i--

else

if(notPossibleCount < 1000)

flag = false

regenerateCountSec = 0

notPossible = true

notPossibleCount++

i= -1

TT = []

teacherTT = {}

secTT = {}

currentTT = []

for u in sections

for v in instances

for w in instaces[v].sections

if(instaces[v].sections[w] ==

sections[u])

instaces[v]["mapp"] = []

secInstances[sections[u]].push(instaces[v])

else

impossible = true

flag = false

if(impossible || notPossible || regenerateFlagSec || regenerateFlagSI)

break

if( impossible || notPossible || regenerateFlagSec)

break

if( (!impossible) && (!regenerateFlagSec) && (!notPossible))

TT.push(currentTT)

secTT[sections[i]] = currentTT

if(notPossible)

notPossible = false

if(impossible)

return("Could not generate in this case, please refresh/restart")

return TT