

Examination Timetabling Problem

ESTADO DE AVANCE

JOSÉ SOUTHERLAND SILVA

CONTENIDO

- 01** RESTRICCIONES DURAS
- 02** RESTRICCIONES BLANDAS
- 03** REPRESENTACIÓN
- 04** LECTURA DE INSTANCIAS
- 05** FUNCIÓN DE EVALUACIÓN
- 06** MOVIMIENTO (HCAM)
- 07** CONCLUSIÓN
- 08** BIBLIOGRAFÍA

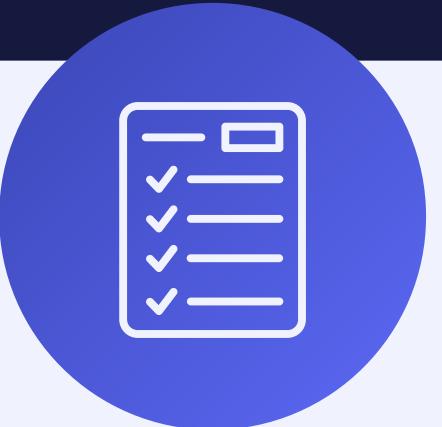
RESTRICCIONES DURAS

Las restricciones duras que se extraen de la Entrega 1 son:



NO MÁS DE UN EXAMEN A LA VEZ

Ningún estudiante debe rendir dos exámenes a la vez.



RENDICIÓN DE EXÁMENES

En cualquier bloque horario, cualquier estudiante está rindiendo a lo más un examen.



BLOQUES HORARIOS

Cada examen se desarrolla en al menos un bloque horario.



ASIGNACIÓN DE SALAS

Cada examen se desarrolla en al menos una sala.



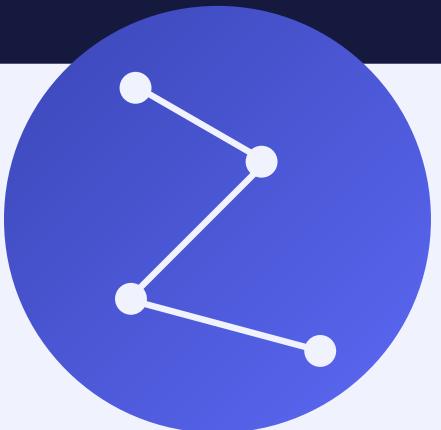
NO MÁS DE UN EXAMEN A LA VEZ

Ningún estudiante debe rendir dos exámenes a la vez.

```
bool hasConflict(const Exam& exam, const vector<int>& timeslot, const map<int, Exam>& exams) {
    for (int examId : timeslot) {
        const vector<string>& students_in_slot = exams.at(examId).students;
        for (const string& student : exam.students) {
            if (find(students_in_slot.begin(), students_in_slot.end(), student) != students_in_slot.end()) {
                return true;
            }
        }
    }
    return false;
}
```

RESTRICCIONES BLANDAS

Las restricciones blandas de la Entrega 1 se reformulan en base a penalizaciones por distancia entre bloques:



PENALIZACIONES POR DISTANCIA ENTRE BLOQUES POR ESTUDIANTE

- +16 no hay bloques que separan a un examen del otro.
- +8 si hay 1 bloque que separa a un examen del otro.
- +4 si hay 2 bloques que separan a un examen del otro.
- +2 si hay 3 bloques que separan a un examen del otro.
- +1 si hay 4 bloques que separan a un examen del otro.
- +0 si hay 5 o más bloques que separan a un examen del otro.



PENALIZACIONES POR DISTANCIA ENTRE BLOQUES POR ESTUDIANTE

```
void calculaPenalizacionPorEstudiante(Student& student, const map<int, int>& examTimeslot) {  
    vector<int> timeslots;  
    for (int exam : student.exams) {  
        timeslots.push_back(examTimeslot.at(exam));  
    }  
    sort(timeslots.begin(), timeslots.end());  
  
    int penalty = 0;  
    for (size_t i = 0; i < timeslots.size() -1; ++i) {  
        int gap = timeslots[i + 1] - timeslots[i];  
        if (gap == 1) penalty += 16;  
        else if (gap == 2) penalty += 8;  
        else if (gap == 3) penalty += 4;  
        else if (gap == 4) penalty += 2;  
        else if (gap == 5) penalty += 1;  
    };  
  
    student.penalty = penalty;  
}
```

REPRESENTACIÓN

Para la representación se crearon los structs Exam y Students.

Se utiliza la estructura map de C++ para trabajar con ambos en la solución inicial.

```
struct Exam {  
    int id;  
    int student_count;  
    vector<string> students;  
};
```

```
struct Student {  
    string id_student;  
    vector<int> exams;  
    int penalty;  
};
```

LECTURA DE INSTANCIAS

La lectura de instancias se lleva a cabo en dos partes: una para el archivo de exámenes y otra para el archivo de estudiantes.

LECTURA DE INSTANCIAS

Archivo de exámenes

```
void readExamsFile(string filename, map<int, Exam>& exams) {  
    ifstream examsFile(filename, ios::in);  
  
    if (!examsFile.is_open()) {  
        cerr << "Error al abrir archivo " << filename << endl;  
        return;  
    }  
  
    int id_exam, students_count;  
    while (examsFile >> id_exam >> students_count) {  
        exams[id_exam] = {id_exam, students_count};  
    }  
  
    examsFile.close();  
}
```

LECTURA DE INSTANCIAS

Archivo de estudiantes

```
void readStudentsFile(string filename, map<string, Student>& students, map<int, Exam>& exams) {  
    ifstream studentsFile(filename, ios::in);  
  
    if(!studentsFile.is_open()) {  
        cerr << "Error al abrir archivo " << filename << endl;  
        return;  
    }  
  
    string id_stu;  
    int id_exam;  
  
    while (studentsFile >> id_stu >> id_exam) {  
        students[id_stu].id_student = id_stu;  
        students[id_stu].exams.push_back(id_exam);  
        exams[id_exam].students.push_back(id_stu);  
    }  
  
    studentsFile.close();  
}
```

SOLUCIÓN INICIAL

- Algoritmo Greedy que sigue los siguientes pasos:

1. Ordenar exámenes de mayor a menor según la cantidad de estudiantes que deben rendirlos.
2. Tomar como **punto de partida** el examen con mayor cantidad de estudiantes que deben rendirlo y lo asigna en el primer timeslot.
3. Continuar según **función miope**: asignar el siguiente examen con la mayor cantidad de estudiantes que deben rendirlo al siguiente bloque horario que no tenga conflicto entre los estudiantes registrados.

```
tuple<map<int, int>, int, double> initialSolution(map<int, Exam>& exams, map<string, Student>& students) {
    vector<Exam> sortedExams;

    // Ordena exámenes de mayor a menor cantidad de estudiantes
    for (const auto& pair : exams) {
        sortedExams.push_back(pair.second);
    }
    sort(sortedExams.begin(), sortedExams.end(), [](const Exam& a, const Exam& b) {
        return a.student_count > b.student_count;
});

map<int, int> examTimeslot;
vector<vector<int>> timeslots;

for (const Exam& exam: sortedExams) {
    bool assigned = false;
    for (int i = 0; i < timeslots.size(); ++i) {
        if (!hasConflict(exam, timeslots[i], exams)) {
            timeslots[i].push_back(exam.id);
            examTimeslot[exam.id] = i;
            assigned = true;
            break;
        }
    }
    if (!assigned) {
        timeslots.push_back({exam.id});
        examTimeslot[exam.id] = timeslots.size() - 1;
    }
}
double promPenalizacion = calculaPenalizacion(students, examTimeslot);
return {examTimeslot, timeslots.size(), promPenalizacion};
}
```

FUNCIÓN DE EVALUACIÓN

La función de evaluación está compuesta por el cálculo del promedio de las penalizaciones de cada estudiante:

```
double calculaPenalizacion(map<string, Student>& students, const map<int, int>& examTimeslot) {
    double total = 0;
    for (auto& [student_id, student] : students) {
        calculaPenalizacionPorEstudiante(student, examTimeslot);
        total += student.penalty;
    }

    return (total / students.size());
}
```

```
void calculaPenalizacionPorEstudiante(Student& student, const map<int, int>& examTimeslot) {
    vector<int> timeslots;
    for (int exam : student.exams) {
        timeslots.push_back(examTimeslot.at(exam));
    }
    sort(timeslots.begin(), timeslots.end());

    int penalty = 0;
    for (size_t i = 0; i < timeslots.size() -1; ++i) {
        int gap = timeslots[i + 1] - timeslots[i];
        if (gap == 1) penalty += 16;
        else if (gap == 2) penalty += 8;
        else if (gap == 3) penalty += 4;
        else if (gap == 4) penalty += 2;
        else if (gap == 5) penalty += 1;
    };

    student.penalty = penalty;
}
```

MOVIMIENTO HILL CLIMBING ALGUNA MEJORA

El movimiento a realizar con Hill Climbing Alguna Mejora será mediante swap, con solución inicial Greedy.

El swap puede ser útil dado que explora las permutaciones de los exámenes en diferentes timeslots sin hacer cambios drásticos en la misma, lo que puede llevar a encontrar mejores soluciones locales y eventualmente mejorar la solución global.

Sala 1	Sala 2	Sala 3
Examen 1	Examen 2	Examen 8
Examen 3	Examen 10	Examen 7
Examen 5	Examen 4	Examen 9
Examen 6	Examen 11	Examen 12

MOVIMIENTO HILL CLIMBING ALGUNA MEJORA

El movimiento a realizar con Hill Climbing Alguna Mejora será mediante swap, con solución inicial Greedy.

El swap puede ser útil dado que explora las permutaciones de los exámenes en diferentes timeslots sin hacer cambios drásticos en la misma, lo que puede llevar a encontrar mejores soluciones locales y eventualmente mejorar la solución global.



MOVIMIENTO HILL CLIMBING ALGUNA MEJORA

El movimiento a realizar con Hill Climbing Alguna Mejora será mediante swap, con solución inicial Greedy.

El swap puede ser útil dado que explora las permutaciones de los exámenes en diferentes timeslots sin hacer cambios drásticos en la misma, lo que puede llevar a encontrar mejores soluciones locales y eventualmente mejorar la solución global.



MOVIMIENTO HILL CLIMBING ALGUNA MEJORA

El movimiento a realizar con Hill Climbing Alguna Mejora será mediante swap, con solución inicial Greedy.

El swap puede ser útil dado que explora las permutaciones de los exámenes en diferentes timeslots sin hacer cambios drásticos en la misma, lo que puede llevar a encontrar mejores soluciones locales y eventualmente mejorar la solución global.



MOVIMIENTO HILL CLIMBING ALGUNA MEJORA

El movimiento a realizar con Hill Climbing Alguna Mejora será mediante swap, con solución inicial Greedy.

El swap puede ser útil dado que explora las permutaciones de los exámenes en diferentes timeslots sin hacer cambios drásticos en la misma, lo que puede llevar a encontrar mejores soluciones locales y eventualmente mejorar la solución global.



MOVIMIENTO HILL CLIMBING ALGUNA MEJORA

El movimiento a realizar con Hill Climbing Alguna Mejora será mediante swap, con solución inicial Greedy.

El swap puede ser útil dado que explora las permutaciones de los exámenes en diferentes timeslots sin hacer cambios drásticos en la misma, lo que puede llevar a encontrar mejores soluciones locales y eventualmente mejorar la solución global.

Sala 1	Sala 2	Sala 3
Examen 1	Examen 10	Examen 8
Examen 3	Examen 2	Examen 7
Examen 5	Examen 4	Examen 9
Examen 6	Examen 11	Examen 12

CONCLUSIÓN

Se logró generar una solución inicial utilizando Greedy, que termina siendo evaluada en base al promedio de penalizaciones de cada estudiante.

- Se debe verificar que el retorno de la solución inicial sea el correcto, en base a lo que se tratará posteriormente en Hill Climbing.
- Se debe aumentar el límite de datos en los maps.



BIBLIOGRAFÍA

- Kusumawardani, D., Muklason, A., & Supoyo, V. A. (2019, July). **Examination timetabling automation and optimization using greedy-simulated annealing hyper-heuristics algorithm.** In 2019 12th International Conference on Information & Communication Technology and System (ICTS) (pp. 1-6). IEEE.