

ESTRUCTURAS DE DATOS

APLICACIONES DE TIPOS ABSTRACTOS DE DATOS

Gestión de una academia (3)


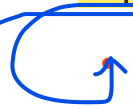

Manuel Montenegro Montes
Departamento de Sistemas Informáticos y Computación
Facultad de Informática – Universidad Complutense de Madrid

Lista de espera



Requisitos

De modo que si el curso no tiene plazas espera en una cola de espera a que alguien se de de baja de un curso para poder matricularse.

- Cada curso tiene una lista de espera.
- Si un estudiante se matricula en un curso y no hay plazas disponibles, se le pone en lista de espera.  supongo que será una cola.
- Cuando un estudiante se da de baja en un curso, se matricula automáticamente al primero de la lista de espera (si existe).
- Operaciones añadidas o modificadas:
 -  Dar de baja a un estudiante.
 -  La operación de matrícula de un curso devuelve un booleano indicando si el estudiante ha sido matriculado o está en lista de espera.

Lista de espera en cursos

```
class Academia {  
public:  
    ...  
  
private:  
    struct InfoCurso {  
        std::string nombre;  
        int numero_plazas;  
        std::unordered_set<Estudiante> estudiantes;  
        std::queue<Estudiante> lista_espera;  
        InfoCurso(const std::string &nombre,  
                   int numero_plazas);  
    };  
    std::unordered_map<Curso, InfoCurso> cursos;  
}
```

El orden en el que salen los elementos debe ser FIFO

Matrícula en un curso (cambios)

```
class Academia {
public:
    bool matricular_en_curso(const Estudiante &est, const Curso &curso) {
        InfoCurso &info_curso = buscar_curso(curso);
        InfoEstudiante &info_est = buscar_estudiante(est);
        if (info_curso.estudiantes.contains(est)) {
            throw std::domain_error("estudiante ya matriculado"); Si está ya matriculado no lo podemos volver a matricular.
        }
        Si no esta matriculado.
        if (info_curso.estudiantes.size() < info_curso.numero_plazas) { Si hay plazas disponibles.
            info_curso.estudiantes.insert(est);
            info_est.cursos.insert(curso); Lo añadido normal al curso
            return true; si se ha matriculado añadido el curso al estudiante también
        } else { Si no hay plazas...
            info_curso.lista_espera.push(est); Meto en la cola de espera al estudiante.
            return false; NO se ha podido matricular.
        }
    }
...
private:
    ...
    std::unordered_map<Curso, InfoCurso> cursos;
}
```

Darse de baja en un curso

```
class Academia {  
public:  
    void dar_de_baja_en_curso(const Estudiante &id_est, const Curso &nombre_curso) {  
        InfoCurso &curso = buscar_curso(nombre_curso);  
  
        auto it_estudiante = curso.estudiantes.find(id_est);  
        if (it_estudiante != curso.estudiantes.end()) {  
            curso.estudiantes.erase(it_estudiante);  
            it_estudiante->cursos.erase(curso.nombre);  
  
            while (!curso.lista_espera.empty() && curso.estudiantes.size() < curso.numero_plazas) {  
                const Estudiante &nif_primerio = curso.lista_espera.front();  
                curso.lista_espera.pop();  
                if (!curso.estudiantes.contains(nif_primerio)) {  
                    curso.estudiantes.insert(nif_primerio);  
                    estudiantes.at(nif_primerio).cursos.insert(curso.nombre);  
                }  
            }  
        }  
    }  
private:  
    std::unordered_map<Curso, InfoCurso> cursos;  
}
```

Estudiante que quiere darse de baja y el nombre del curso del que quiere darse de baja.

busco el curso

busco el estudiante en el curso

Si fuera igual a end implicaría que no hemos encontrado al estudiante. Podríamos añadir un throw para que se mostrará que no existe.

Hace correr la lista de espera.