COLAS DE PRIORIDAD

UNIVERSIDAD COMPLUTENSE MADRID

ALBERTO VERDEJO

Colas de prioridad

- En las colas “ordinarias” se atiende por riguroso orden de llegada (FIFO).
- También hay colas en las cuales se atiende según la prioridad de los elementos y no según el orden de llegada: son las colas de prioridad.
- Cada elemento tiene una prioridad que determina quién va a ser el primero en ser atendido; hace falta tener un orden total sobre ellos.
- El primero en ser atendido puede ser el elemento con *menor* valor o el elemento con *mayor* valor según se trate de colas de prioridad de mínimos o de máximos, respectivamente.

Ejemplo: cola de prioridad de máximos

- Empezamos con una cola vacía

Cola:

Ejemplo: cola de prioridad de máximos

- Llega el 15

Cola:

Ejemplo: cola de prioridad de máximos

Cola: 15

más prioritario

Ejemplo: cola de prioridad de máximos

- Llega el 7

Cola: 15

más prioritario

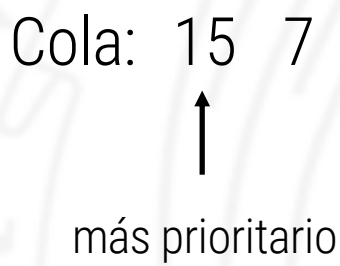
Ejemplo: cola de prioridad de máximos

Cola: 15 7

más prioritario

Ejemplo: cola de prioridad de máximos

- Llega el 10



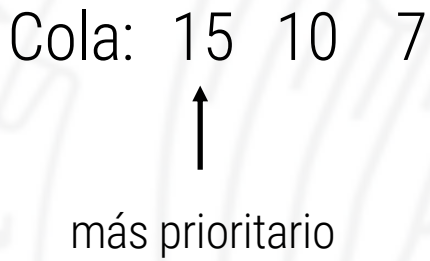
Ejemplo: cola de prioridad de máximos

Cola: 15 10 7

más prioritario

Ejemplo: cola de prioridad de máximos

- Sale el más prioritario



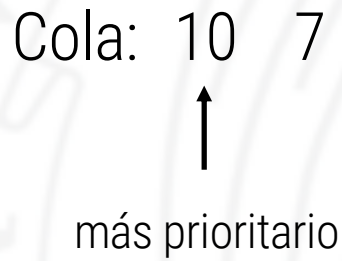
Ejemplo: cola de prioridad de máximos

Cola: 10 7

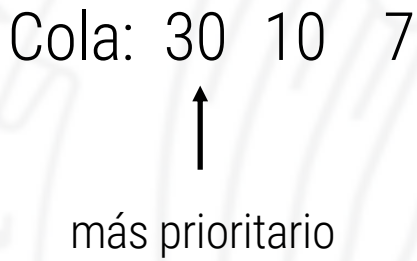
más prioritario

Ejemplo: cola de prioridad de máximos

- Llega el 30

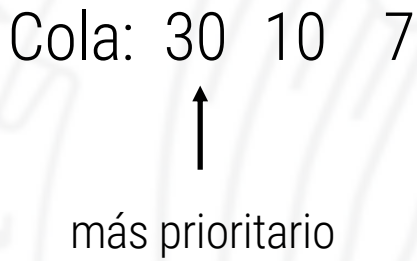


Ejemplo: cola de prioridad de máximos



Ejemplo: cola de prioridad de máximos

- Llega el 23



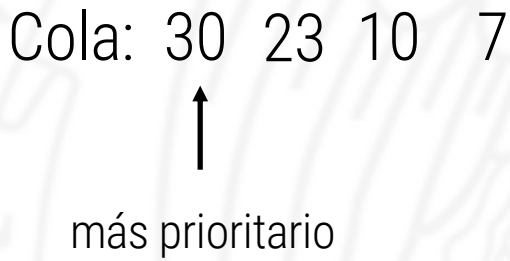
Ejemplo: cola de prioridad de máximos

Cola: 30 23 10 7

más prioritario

Ejemplo: cola de prioridad de máximos

- Sale el más prioritario



Ejemplo: cola de prioridad de máximos

Cola: 23 10 7

más prioritario

TAD de las colas de prioridad

El TAD de las colas de prioridad cuenta con las siguientes operaciones:

- crear una cola de prioridad vacía
- insertar un elemento, `void push(T const& e)`
- consultar el elemento más prioritario, `T const& top() const`
- eliminar el primer elemento, `void pop()`
- determinar si la cola de prioridad es vacía, `bool empty() const`
- consultar el número de elementos de la cola, `int size() const`

Colas de prioridad en la STL de C++

- La librería `queue` de la STL contiene la clase `priority_queue` que implementa colas de prioridad de máximos.
- Dado un orden, como $<$, la operación `top()` devuelve el elemento mayor, el que se encuentra más a la derecha en el orden

$$a_1 < a_2 < \cdots < a_n$$

Colas de prioridad en la STL de C++

- Podemos utilizar esas colas como colas de mínimos si cambiamos el objeto comparador, utilizando el operador $>$:

$$b_1 > b_2 > \cdots > b_n$$

- El comparador es el tercer argumento de la plantilla:

```
priority_queue<int, vector<int>, greater<int>> cola_min;
```

Unidad Curiosa de Monitorización

La Unidad Curiosa de Monitorización (UCM) se encarga de leer los datos proporcionados por una serie de sensores y enviar con cierta periodicidad los datos obtenidos y procesados a los usuarios que se han registrado previamente.

La UCM admite que los usuarios se registren proporcionando un Identificador único y un Periodo, el intervalo de tiempo que transcurrirá entre dos envíos consecutivos de información a ese usuario.

Acaban de registrarse varios usuarios. ¿Podrías decir a quiénes irán dirigidos los *K* primeros envíos de información? Si dos o más usuarios tienen que recibir la información al mismo tiempo, los envíos se realizan en orden creciente de sus identificadores de usuario.

Unidad Curiosa de Monitorización

```
struct registro {
    int momento;  // cuándo le toca
    int id;       // identificador (se utiliza en caso de empate)
    int periodo;  // tiempo entre consultas
};


bool operator<(registro const& a, registro const& b) {
    return b.momento < a.momento ||
           (a.momento == b.momento && b.id < a.id);
}
```

Unidad Curiosa de Monitorización

```
bool resuelveCaso() {
    int N; // número de usuarios registrados
    cin >> N;

    if (N == 0) return false; // no hay más casos

    priority_queue<registro> cola;

    // leemos los registros
    for (int i = 0; i < N; ++i) {
        int id_usu, periodo;
        cin >> id_usu >> periodo;
        cola.push({periodo, id_usu, periodo});
    }
```

Unidad Curiosa de Monitorización

```
    int envios; // número de envíos a mostrar
    cin >> envios;

    while (envios--) {
       auto e = cola.top(); cola.pop();
       cout << e.id << '\n';
       e.momento += e.periodo;
       cola.push(e);
    }
    cout << "---\n";
    return true;
}
```

Unidad Curiosa de Monitorización

¿Y si el $<$ ordena *de menor a mayor*?

```
struct registro {
    int momento;  // cuándo le toca
    int id;       // identificador (se utiliza en caso de empate)
    int periodo;  // tiempo entre consultas
};

bool operator<(registro const& a, registro const& b) {
    return a.momento < b.momento ||
           (a.momento == b.momento && a.id < b.id);
}

bool operator>(registro const& a, registro const& b) {
    return b < a;
}
```

Unidad Curiosa de Monitorización

```
bool resuelveCaso() {
    int N; // número de usuarios registrados
    cin >> N;

    if (N == 0) return false; // no hay más casos

    priority_queue<registro, vector<registro>, greater<registro>> cola;

    // leemos los registros
    for (int i = 0; i < N; ++i) {
        int id_usu, periodo;
        cin >> id_usu >> periodo;
        cola.push({periodo, id_usu, periodo});
    }
```

Unidad Curiosa de Monitorización

¿Y si el $<$ ordena *de una forma diferente*?

```
struct registro {
    int momento;  // cuándo le toca
    int id;       // identificador (se utiliza en caso de empate)
    int periodo;  // tiempo entre consultas
};

struct comp_registro {
    bool operator()(registro const& a, registro const& b) {
        return b.momento < a.momento ||
               (a.momento == b.momento && b.id < a.id);
    }
};
```

Unidad Curiosa de Monitorización

```
bool resuelveCaso() {
    int N; // número de usuarios registrados
    cin >> N;

    if (N == 0) return false; // no hay más casos

    priority_queue<registro, vector<registro>, comp_registro> cola;

    // leemos los registros
    for (int i = 0; i < N; ++i) {
        int id_usu, periodo;
        cin >> id_usu >> periodo;
        cola.push({periodo, id_usu, periodo});
    }
```