

Algorítmica

Relación 3: Algoritmos voraces

Ejercicio 1

```
bool Factible(const int menor, const int longitud, const int actual){
    return ((actual + menor) <= longitud);
}

int Seleccion(vector<int> &programas, int longitud, int actual){
    int menor = -1;
    auto iter = programas.begin();

    if(Factible(*iter, longitud, actual) && !programas.empty()){ //Cuando esté
vacío el vector de programas o no sea factible será que ya habremos terminado
        menor = (*iter);
        //porque estarán ordenados de menor a mayor
        programas.erase(iter);
    }

    return menor;
}

vector<int> cintaVoraz(const int longitud, vector<int> programas){
    vector<int> resultado;
    int actual = 0;
    int programaAniadir;
    bool terminar = false;

    QuickSort(programas, 0, (int)programas.size());

    for(int i = 0; i < programas.size() && !terminar; i++){
        programaAniadir = Seleccion(programas, longitud, actual);

        if(programaAniadir != -1){
            resultado.push_back(programaAniadir);
            actual += programaAniadir;
        }
        else{
            terminar = true;
        }
    }

    return resultado;
}
```

En el peor de los casos, este algoritmo voraz tendrá una eficiencia de $O(n)$.

Ejercicio 3

```

int min(vector< pair<char, int> > usado_TD){
    int min = 100000000; // Valor muy alto
    int posicion = -1;

    for(int i = 0; i < usado_TD.size(); i++){
        if(usado_TD[i].first != 'U' && usado_TD[i].second < min){
            min = usado_TD[i].second;
            posicion = i;
        }
    }

    return posicion;
}

int Seleccion(vector< pair<char, int> > &usado_TD){
    int posicion;

    posicion = min(usado_TD);
    usado_TD[posicion].first = 'U'; // Usado

    return posicion;
}

vector<int> planificacionVoraz(vector< pair<int, int> > duracion_terminacion){
    vector<int> planificacion;
    vector< pair<char, int> > ordenacion;
    pair<char, int> valor_vector;

    valor_vector.first = 'N'; // No usado

    // Calculamos una especie de "heurística" y a partir de ahí haremos la
    planificación de procesos
    for(int i = 0; i < duracion_terminacion.size(); i++){
        valor_vector.second = duracion_terminacion[i].second -
duracion_terminacion[i].first;
        ordenacion.push_back(valor_vector);
    }

    for(int i = 0; i < ordenacion.size(); i++){
        planificacion.push_back(Seleccion(ordenacion));
    }

    return planificacion;
}

```

En el peor de los casos, este algoritmo voraz tendrá una eficiencia de $O(n^2)$.

Ejercicio 4

```
int Seleccion(vector<int> &tamamos){
    int seleccionado = -1;
    auto iter = tamamos.begin();

    seleccionado = (*iter);
    tamamos.erase(iter);

    return seleccionado;
}

int unionVoraz(vector<int> tamanoCintas){
    vector<int> sumasParciales;
    int movimientosTotal = 0;

    QuickSort(tamanoCintas, 0, (int)tamanoCintas.size());

    movimientosTotal += Seleccion(tamanoCintas);

    for(int i = 0; !tamanoCintas.empty(); i++){
        movimientosTotal += Seleccion(tamanoCintas);
        sumasParciales.push_back(movimientosTotal);
    }

    movimientosTotal = 0;

    for(int i = 0; i < sumasParciales.size(); i++){
        movimientosTotal += sumasParciales[i];
    }

    return movimientosTotal;
}
```

En el peor de los casos, este algoritmo voraz tendrá una eficiencia de $O(n)$.

Ejercicio 5

```
int Seleccion(vector<int> &gasolineras, int kmDeposito){
    int lejana = 0;
    bool terminar = false;

    for(auto it = gasolineras.begin(); it != gasolineras.end() && !terminar; it++){
        kmDeposito -= (*it);
        lejana += (*it);

        if(kmDeposito < 0){
            kmDeposito += (*it);
            lejana -= (*it);
            terminar = true;
        }

        if(!gasolineras.empty()){
            it = gasolineras.erase(it);
        }
    }

    return lejana;
}

vector<int> gasolinerasVoraz(vector<int> gasolineras, const int kmDeposito){
    vector<int> planViaje;
    int repostaje = -1;

    for (int i = 0; !gasolineras.empty(); i++) {
        repostaje = Seleccion(gasolineras, kmDeposito);
        planViaje.push_back(repostaje);
    }

    return planViaje;
}
```

En el peor de los casos, este algoritmo voraz tendrá una eficiencia de $O(n^2)$.