Relación 3 - Voraces

Rubén Calvo Villazán

1) Almacenar n programas en una cinta de longitud L.

1 - Candidatos a seleccionar: Programas
2 - Candidatos seleccionados: Programas
3 - Función solución: Nº de programas que caben en el disco
4 - Función factible: Cabe en el disco
5 - Función selección: El programa que ocupe menor espacio
6 - Función objetivo: Almacenar nº de programas posible

```
int Almacenar(vector<int> programas, int espacio){
    int guardados = 0; int ocupado = 0;
    priority_queue<int> pq;
    for(int i=0; i<programas.size(); i++)
        pq.push(programas[i]);
    while(ocupado <= espacio){
        int p = pq.top();
        if(ocupado + p <= espacio){
            ocupado += p;
            guardados++;
        }
        pq.pop();
    }
    return guardados;
}
```

5) Repostar en las menores gasolineras posibles

1- Candidatos a seleccionar: gasolineras
2- Candidatos seleccionados: gasolineras
3- Función solución: Conjunto de gasolineras menor
4- Función factible: Relación gasolina, recorrido hasta gasolinera
5 - Función selección: Siguiente gasolinera
6- Función objetivo: Repostar el menor número de veces

Parando en la gasolinera que más lejos esté en relación $\frac{x}{dist}$

```
vector<int> Gasolineras(vector<int> D, int x){
    priority_queue<int, greater> dist;
    vector<int> res;
    for(int i=0; i<D.size(); i++)
        dist.push(D[i]);
    while(dist.size()){
        int x = Seleccionar(dist, x);
        res.push_back(x);
    }
  return res;
}
int Seleccionar(priority_queue<int> dist, int x){
        int g= dist.top();
      while( x/g <1){
            dist.pop();
            g= dist.top();
      }
   return g;
}
```

4) Mezclar cartas menor movimientos posibles

1- Candidatos a seleccionar: cartas
2- Candidatos seleccionados: cartas
3- Función solución: Menos movimientos posibles
4- Función factible: Siempre es factible (siguiente carta)
5- Función selección: Siguiente carta con menos registros
6- Función objetivo: Menor movimientos posibles

```
int MovimientosC(vector<int> cartas){
    priority_queue<int> pq;
    int movimientos = 0;
    for(int i=0; i< cartas.size(); i++)
        pq.push(cartas[i]);

    while (pq.size()){
        movimientos += pq.top();
        pq.pop();
    }

    return movimientos;
}
```

2) Tiempo mínimo de lectura de programas de la cinta

1 - Candidatos a seleccionar: Programas
2 - Candidatos seleccionados: Programas
3 - Función solución: Programas con mínimo tiempo de lectura
4 - Función factible: Cabe en el disco
5 - Función selección: El programa que ocupe menor espacio (Menor tiempo de lectura)
6 - Función objetivo: Almacenar máx. de programas con menor tiempo de lectura

```
int Almacenar (vector<int> programas, int espacio){
    int guardados = 0, ocupado = 0;
    priority_queue<int> pq;
    for(int i=0; i< programas.size(); i++)
        pq.push(programas[i]);
    while(ocupado <= espacio){
        int p = pq.top();
        if(ocupado + p <= espacio){
            ocupado += p;
            guardados++;
        }
        pq.pop();
    }
    return guardados;
}
```

3) Procesar n tareas

1- Candidatos a seleccionar: tareas
2- Candidatos seleccionados: Tareas
3- Función solución: Todas las tareas procesadas
4- Función factible: Termina antes de su instante exigido
5- Función selección: Siguiente tarea con mayor relación ti - pi + 1
6- Función objetivo: Procesar n tareas

```
struct tarea{ bool operator< (const tarea &t1, const tarea &t2){
                return ((t1.ti - t1.pi+1) < (t2.ti - t2.pi+1));
  int ti;     }
  int pi;
};

int Procesar_Tareas (const vector<tarea> &T, int n){
    int procesados = 0;
    priority_queue <tarea> pq;
    for(int i=0; i< t.size(); i++)
        pq.push(T[i]);
    while (pq.size() && procesados <= n){
        procesados++;
        pq.pop();
    }
  return procesados;
}
```