Algoritmos de ordenamiento sobre secuencias

Algoritmos y Estructuras de Datos I

Ordenamiento de vectores

- proc *ordenar*(inout $s : seq\langle\mathbb{Z}\rangle$){
  Pre $\{s = S_0\}$
  Post $\{mismos(s, S_0) \ \wedge \ ordenado(s)\}$
  }
- pred *mismos*($s, t : seq\langle\mathbb{Z}\rangle$){
  $|s| = |t| \wedge$
  $(\forall i : \mathbb{Z})(0 \leq i < |s|$
  $\to_L \#apariciones(s, s[i]) = \#apariciones(t, s[i]))$
  }
- fun $\#apariciones(s : seq\langle T\rangle, e : T) : \mathbb{Z} =$
  $\sum_{i=0}^{|s|-1}(\text{if } s[i] = e \text{ then } 1 \text{ else } 0 \text{ fi})$
- pred *ordenado*($s : seq\langle\mathbb{Z}\rangle$){
  $(\forall i : \mathbb{Z})(0 \leq i < |s| - 1 \to_L s[i] \leq s[i+1])$
  }

Ordenamiento de vectores

- Modificamos el vector solamente a través de intercambios de elementos.

proc *swap*(inout $s : seq\langle\mathbb{Z}\rangle$, in $i, j : \mathbb{Z}$){
Pre $\{0 \leq i, j < |s| \wedge s = S_0\}$
Post $\{s[i] = S_0[j] \wedge s[j] = S_0[i] \wedge$
$(\forall k : \mathbb{Z})(0 \leq k < |s| \wedge i \neq k \wedge j \neq k \to_L s[k] = S_0[k])\}$
}

- **Propiedad:**

$$\{s = S_0\}$$
$$\text{swap(s,i,j)}$$
$$\{mismos(s, S_0)\}$$

- De esta forma, nos aseguramos que $mismos(s, S_0)$ a lo largo de la ejecución del algoritmo.

Ordenamiento por selección (Selection Sort)

- **Notación:** $s[i, j] = subseq(s, i, j+1)$.
  $s[i, j) = subseq(s, i, j)$.
- **Observación:** *subseq* nunca se indefine (devuelve la secuencia vacía $\langle\rangle$)
- **Idea:** Seleccionar el mínimo elemento e intercambiarlo con la primera posición del vector. Repetir con el segundo, etc.

```
void seleccion(vector<int> &s) {
  for(int i=0; i<s.size(); i++) {
    int pos = // ubicacion del minimo de s entre i y s.size()
    swap(s, i, pos);
  }
}
```

Ordenamiento por selección (Selection Sort)

- Podemos refinar un poco el código:

```
void seleccion(vector<int> &s) {
  for(int i=0; i<s.size()-1; i++) {
    int pos = minimo(s, i, s.size());
    swap(s, i, pos);
  }
}
```

- Surge la necesidad de especificar el problema auxiliar de buscar el mínimo entre i y $s.size()$:

proc $minimo$(inout $s : seq\langle\mathbb{Z}\rangle$, in $d, h : Int$, out $result : \mathbb{Z}$){
  Pre $\{0 \leq d < h \leq |s|\}$
  Post $\{d \leq result < h$
    $\wedge_L\ (\forall i : \mathbb{Z})(d \leq i < h \rightarrow_L s[result] \leq s[i])\}$
}

Buscar el Mínimo Elemento

- proc $minimo$(inout $s : seq\langle\mathbb{Z}\rangle$, in $d, h : \mathbb{Z}$, out $result : \mathbb{Z}$){
  Pre $\{0 \leq d < h \leq |s|\}$
  Post $\{d \leq result < h$
    $\wedge_L(\forall i : \mathbb{Z})(d \leq i < h \rightarrow_L s[result] \leq s[i])\}$
  }
- Necesitamos implementar la especificación de **minimo**:

```
int minimo(vector<int> &s, int d, int h) {
  int result = d;
  for(int i=d+1; i<h; i++) {
    if (s[i]<s[result]) {
      result = i;
    }
  }
  return result;
}
```

Recap: Teorema de corrección de un ciclo

- **Teorema.** Sean un predicado I y una función $fv : \mathbb{V} \rightarrow \mathbb{Z}$ (donde $\mathbb{V}$ es el producto cartesiano de los dominios de las variables del programa), y supongamos que $I \Rightarrow \mathrm{def}(B)$. Si

  1. $P_C \Rightarrow I$,
  2. $\{I \wedge B\}\ S\ \{I\}$,
  3. $I \wedge \neg B \Rightarrow Q_C$,
  4. $\{I \wedge B \wedge v_0 = fv\}$ **S** $\{fv < v_0\}$,
  5. $I \wedge fv \leq 0 \Rightarrow \neg B$,

  ... entonces la siguiente tripla de Hoare es válida:

$$\{P_C\}\ \textbf{while B do S endwhile}\ \{Q_C\}$$

Buscar el Mínimo Elemento

- $P_C \equiv 0 \leq d < h \leq |s| \wedge result = d \wedge i = d + 1$
- $Q_C \equiv d \leq result < h$
  $\wedge_L(\forall i : \mathbb{Z})(d \leq i < h \rightarrow_L s[result] \leq s[i])$
- $B \equiv i < h$
- $I \equiv ?$
  - En cada iteración del ciclo, la variable **result** contiene el índice del menor elemento encontrado hasta ahora. Es decir, $s[result]$ el menor elemento de $s[d, i)$.
  - ¡Esto nos proporciona el invariante del ciclo!

  $d \leq result < i \leq h\ \wedge_L\ (\forall j : \mathbb{Z})(d \leq j < i \rightarrow_L s[result] \leq s[j])$
- $fv = h - i$

Buscar el Mínimo Elemento

- $P_C \equiv 0 \leq d < h \leq |s| \wedge result = d \wedge i = d + 1$
- $Q_C \equiv d \leq result < h$
  $\wedge_L (\forall i : \mathbb{Z})(d \leq i < h \rightarrow_L s[result] \leq s[i])$
- $B \equiv i < h$
- $I \equiv d \leq result < i \leq h$
  $\wedge_L (\forall j : \mathbb{Z})(d \leq j < i \rightarrow_L s[result] \leq s[j])$
- $fv = h - i$

```
int minimo(vector<int> &s, int d, int h) {
  int result = s[d];
  for(int i=d+1; i<h; i++) {
    if (s[result] < s[i]) {
      result = i;
    }
  }
  return result;
}
```

Correctitud: Buscar el Mínimo Elemento

- $P_C \equiv 0 \leq d < h \leq |s| \wedge result = d \wedge i = d + 1$
- $Q_C \equiv d \leq result < h$
  $\wedge_L (\forall i : \mathbb{Z})(d \leq i < h \rightarrow_L s[result] \leq s[i])$
- $B \equiv i < h$
- $I \equiv d \leq result < i \leq h$
  $\wedge_L (\forall j : \mathbb{Z})(d \leq j < i \rightarrow_L s[result] \leq s[j])$
- $fv = h - i$

- ¿I es se cumple al principio del ciclo (punto 1.)? ✓
- ¿Se cumple la postcondición del ciclo a la salida del ciclo (punto 3.)? ✓
- ¿Si la función variante alcanza la cota inferior la guarda se deja de cumplir (punto 5.)? ✓

Correctitud: Buscar el Mínimo Elemento

- $I \equiv d \leq result < i \leq h$
  $\wedge_L (\forall j : \mathbb{Z})(d \leq j < i \rightarrow_L s[result] \leq s[j])$
- $fv = h - i$

```
int minimo(vector<int> &s, int d, int h) {
  int result = s[d];
  for(int i=d+1; i<h; i++) {
    if (s[result] < s[i]) {
      result = i;
    }
  }
  return result;
}
```

- ¿I se preserva en cada iteración (punto 2.)? ✓
- ¿La función variante es estrictamente decreciente (punto 4.)?✓

Ordenamiento por selección (Selection Sort)

- Volvamos ahora al programa de ordenamiento por selección:

```
void seleccion(vector<int> &s) {
  for(int i=0; i<s.size(); i++) {
    int pos = minimo(s, i, s.size());
    swap(s, i, pos);
  }
}
```

- $P_C \equiv i = 0 \wedge s = S_0$
- $Q_C \equiv mismos(s, S_0) \wedge \ ordenado(s)$
- $B \equiv i < |s|$
- $I \equiv ?$
  - ¡Luego de la i-ésima iteración, $s[0, i)$ contiene los i primeros elementos ordenados! ¿Tenemos entonces el invariante del ciclo?
  - $I \equiv mismos(s, S_0) \wedge ordenado(s[0, i)) \wedge (0 \leq i \leq |s|)$
- $fv = |s| - i$

Ordenamiento por selección (Selection Sort)

- $I \equiv 0 \leq i \leq |s| \ \wedge \ mismos(s, S_0) \ \wedge \ ordenado(s[0, i))$
- $fv = |s| - i$

```
void seleccion(vector<int> &s) {
  for(int i=0; i<s.size(); i++) {
    int pos = minimo(s, i, s.size());
    swap(s, i, pos);
  }
}
```

- ¿I se preserva en cada iteración (punto 2.)? ✗
- Contraejemplo:
  - Si arrancamos la iteración con $i = 1$ y $s = \langle 100, 2, 1 \rangle$
  - Terminamos con $i = 2$ y $s = \langle 100, 1, 2 \rangle$ que no satisface I

Debemos reforzar el invariante para probar la corrección:

$$I \equiv \ mismos(s, S_0) \ \wedge \ ordenado(s[0, i)) \wedge$$
$$(0 \leq i \leq |s| \wedge_L (\forall j, k : \mathbb{Z})((0 \leq j < i \wedge i \leq k < |s|) \rightarrow_L s[j] \leq s[k]))$$

Correctitud: Ordenamiento por selección (Selection Sort)

$$\begin{aligned} I \ \equiv \ & mismos(s, S_0) \ \wedge \ ordenado(s[0, i)) \\ \wedge \ & (0 \leq i \leq |s| \wedge_L (\forall j, k : \mathbb{Z})((0 \leq j < i \wedge i \leq k < |s|) \rightarrow_L s[j] \leq s[k])) \end{aligned}$$

Gráficamente:

$x \in s[0, i)$	$y \in s[i+1, \|s\|)$
$\leq y$	$\geq x$
ordenado	?

Correctitud: Ordenamiento por selección (Selection Sort)

- $P_C \equiv i = 0 \wedge s = S_0$
- $Q_C \equiv mismos(s, S_0) \wedge \ ordenado(s)$
- $B \equiv i < |s|$
- $I \equiv \ mismos(s, S_0) \ \wedge \ ordenado(s[0, i)) \wedge$ $(0 \leq i \leq |s| \wedge_L (\forall j, k : \mathbb{Z})((0 \leq j < i \wedge i \leq k < |s|) \rightarrow_L s[j] \leq s[k]))$
- $fv = |s| - i$

- ¿I es se cumple al principio del ciclo (punto 1.)? ✓
- ¿Se cumple la postcondición del ciclo a la salida del ciclo (punto 3.)?✓
- ¿Si la función variante alcanza la cota inferior la guarda se deja de cumplir (punto 5.)?✓

Correctitud: Ordenamiento por selección (Selection Sort)

- $I \equiv \ mismos(s, S_0) \ \wedge \ ordenado(s[0, i)) \wedge$ $(0 \leq i \leq |s| \wedge_L (\forall j, k : \mathbb{Z})((0 \leq j < i \wedge i \leq k < |s|) \rightarrow_L s[j] \leq s[k]))$
- $fv = |s| - i$

```
void seleccion(vector<int> &s) {
  for(int i=0; i<s.size(); i++) {
    int pos = minimo(s, i, s.size());
    swap(s, i, pos);
  }
}
```

- ¿I se preserva en cada iteración (punto 2.)? ✓
- ¿La función variante es estrictamente decreciente (punto 4.)?✓

Ordenamiento por selección (Selection Sort)

```
int minimo(vector<int> &s, int d, int h) {
  int result = s[d];
  for(int i=d+1; i<h; i++) {
    if (s[result] < s[i]) {
      result = i;
    }
  }
  return result;
}
void seleccion(vector<int> &s) {
  for(int i=0; i<s.size(); i++) {
    int pos = minimo(s,i,s.size());
    swap(s, i, pos);
  }
}
```

- ¿Cómo se comporta este algoritmo?
- Veámoslo en `https://visualgo.net/es/sorting`.

Ordenamiento por selección (Selection Sort)

- ¿Cuántas iteraciones ejecuta este programa en peor caso?
  - Para ello contamos la cantidad de veces que se ejecuta el **if** de `minimo`

$$\text{ejecuciones}_{if} = \sum_{i=0}^{|s|-1} |s|-i = |s|\times|s|-\frac{(|s|-1)\times|s|}{2} \leq (|s|)^2.$$

  - Decimos que el algoritmo de ordenamiento por selección es un algoritmo cuadrático (¡más información en **algo2**!).
- ¿Puede ejecutarse una cantidad menor de veces?
  - Siempre se ejecuta la misma cantidad de veces. El peor caso es igual al mejor caso.

Ordenamiento por selección (Selection Sort)

- Variantes del algoritmo básico:
  1. Cocktail sort: consiste en buscar en cada iteración el máximo y el mínimo del vector por ordenar, intercambiando el mínimo con i y el máximo con $|s| - i - 1$.
  2. Bingo sort: consiste en ubicar todas las apariciones del valor mínimo en el vector por ordenar, y mover todos los valores mínimos al mismo tiempo (efectivo si hay muchos valores repetidos).
- Ambas variantes también son algoritmos cuadráticos (iteran una cantidad cuadrática de veces).

Intervalo

Break!

Ordenamiento por inserción (Insertion Sort)

- Veamos un segundo algoritmo, en el que usaremos como invariante únicamente que:

$$I \equiv 0 \leq i \leq |s| \ \wedge_L \ mismos(s, S_0) \ \wedge \ ordenado(s[0, i))$$

- Esto implica que (a diferencia del algoritmo de selection sort) en cada iteración los primeros i elementos están ordenados, sin ser necesariamente los i elementos más pequeños del vector.

Ordenamiento por inserción (Insertion Sort)

$$I \equiv 0 \leq i \leq |s| \ \wedge_L \ mismos(s, S_0) \wedge \ ordenado(s[0, i))$$

```
void insercion(vector<int> &s) {
  for(int i=0; i<s.size(); i++) {
    // Tenemos que preservar el invariante ...
  }
}
```

- ¿I es se cumple al principio del ciclo (punto 1.)? ✓
- ¿Se cumple la postcondición del ciclo a la salida del ciclo (punto 3.)? ✓
- ¿I se preserva en cada iteración (punto 2.)?
  - Sabiendo que los primeros i elementos están ordenados, tenemos que hacer que los primeros $i+1$ elementos pasen a estar ordenados!
  - ¿Cómo lo podemos hacer?
    - ¡Insertando s[i] en la posición temporaria que le corresponda!

Ordenamiento por inserción (Insertion Sort)

```
void insercion(vector<int> &s) {
  for(int i=0; i<s.size(); i++) {
    for(int j=i; j>0 && s[j] < s[j-1] ; j--) {
      swap(s, j, j-1);
    }
  }
}
```

$$I_{ext} \equiv mismos(s, S_0) \wedge ordenado(s[0, i)) \wedge (0 \leq i \leq |s|)$$

- ¿Cuál es el invariante del `for` interno?

$$\begin{aligned} I_{int} \ \equiv \ & 0 \leq j \leq i \\ \wedge \ & mismos(s[0, i+1), S_0[0, i+1)) \\ \wedge \ & s[i+1, |s|) = S_0[i+1, |s|) \\ \wedge \ & ordenado(s[0, j)) \ \wedge \ ordenado(s[j, i+1)) \\ \wedge \ & (\forall k : \mathbb{Z})(j < k \leq i \rightarrow_L s[j] < s[k]) \end{aligned}$$

Ordenamiento por inserción (Insertion Sort)

Gráficamente:

- Invariante del ciclo exterior

ordenada	
	?
$s[0, i)$	$s[i, \|s\|)$

- Invariante del ciclo interior

ordenada		ordenada	
	x	$> x$	?
$s[0, j)$	$s[j]$	$s[j+1, i+1)$	$s[i+1, \|s\|)$

Ordenamiento por inserción (Insertion Sort)

- ```
  void insercion(vector<int> &s) {
    for(int i=0; i<s.size(); i++) {
      for( int j=i; j>0 && s[j] < s[j-1] ; j--) {
        swap(s, j, j-1);
      }
    }
  }
  ```
- ¿Cómo son las funciones variantes de cada ciclo?

$$\begin{aligned} fv_{ext} &= |s| - i \\ fv_{int} &= j \end{aligned}$$

Ordenamiento por inserción (Insertion Sort)

```
void insercion(vector<int> &s) {
  for(int i=0; i<s.size(); i++) {
    for( int j = i; j>0 && s[j] < s[j-1] ; j--) {
      swap(s, j, j-1);
    }
  }
}
```

- ¿Cómo se comporta este algoritmo de ordenamiento?
- Veámoslo en `https://visualgo.net/es/sorting`.

Ordenamiento por inserción (Insertion Sort)

- ¿Cuántas veces se ejecuta el **swap** del ciclo interior?
  - ¡Depende de los datos!
- Analizamos el peor caso (es decir, que el `while` interior realice $i+1$ iteraciones).

$$\text{ejecuciones}_{if} = \sum_{i=0}^{|s|} i + 1 = \frac{|s| \times (|s|+1)}{2} + |s| \leq |s|^2$$

- El algoritmo de ordenamiento por inserción también es un algoritmo cuadrático (itera una cantidad cuadrática de veces)
- **Observación:** Selection sort e insertion sort se pueden generalizar a secuencias de tipo T con un predicado de orden $\leq$ (no solamente funcionan con secuencias de $\mathbb{Z}$)

Eficiencia de los Algoritmos de ordenamiento

- Tanto selection sort como insertion sort son algoritmos **cuadráticos** (iteran una cantidad cuadrática de veces)
- ¿Hay algoritmos con comportamiento más eficiente?
  - Quicksort y BubbleSort: Peor caso cuadrático $(|s|)^2$
  - Mergesort y Heapsort: Peor caso: $|s| \times log_2(|s|)$
  - Counting sort (para secuencias de enteros). Peor caso: $|s|$
  - Radix sort (para secuencias de enteros). Peor caso: 2^{32}
- Bubble sort está en la práctica 8. El resto los van a ver en **algo2**.
```

Bibliografía

- Vickers et al. - Reasoned Programming
  - 6.5 - Insertion Sort
- NIST- Dictionary of Algorithms and Data Structures
  - Selection Sort - https://xlinux.nist.gov/dads/HTML/selectionSort.html
  - Bingo Sort - https://xlinux.nist.gov/dads/HTML/bingosort.html
  - Cocktail Sort - https://xlinux.nist.gov/dads/HTML/bidirectionalBubbleSort.html