Algoritmos sobre secuencias ya ordenadas

Algoritmos y Estructuras de Datos I

Apareo (merge) de secuencias ordenadas

- **Problema:** Dadas dos secuencias ordenadas, unir ambas secuencias en un única secuencia ordenada.
- Especificación:
  proc *merge*(in $a, b : seq\langle\mathbb{Z}\rangle$, out $result : seq\langle\mathbb{Z}\rangle$){
  Pre $\{ordenado(a) \ \wedge \ ordenado(b)\}$
  Post $\{ordenado(result) \ \wedge \ mismos(result, a++b)\}$
  }

  pred *mismos*($s, t : seq\langle\mathbb{Z}\rangle$){
  $(\forall x : \mathbb{Z})(\#apariciones(s, x) = \#apariciones(t, x))$
  }
- ¿Cómo lo podemos implementar?
  - Podemos copiar los elementos de a y b a la secuencia c, y después ordenar la secuencia c.
  - Pero selection sort e insertion sort iteran apróximadamente $|c|^2$
  - ¿Se podrá aparear ambas secuencias en una única pasada?

Apareo de secuencias ordenadas

Ejemplo:

- $a = \begin{array}{cccccc} \boxed{1} & \boxed{3} & \boxed{5} & \boxed{7} & \boxed{9} & \boxed{} \\ \uparrow & \uparrow & \uparrow & \uparrow & \uparrow & \uparrow \\ i & i & i & i & i & i \end{array}$
- $b = \begin{array}{ccccc} \boxed{2} & \boxed{4} & \boxed{6} & \boxed{8} & \boxed{} \\ \uparrow & \uparrow & \uparrow & \uparrow & \uparrow \\ j & j & j & j & j \end{array}$
- $c = \begin{array}{cccccccccc} \boxed{?1} & \boxed{?2} & \boxed{?3} & \boxed{?4} & \boxed{?5} & \boxed{?6} & \boxed{?7} & \boxed{?8} & \boxed{?9} & \boxed{} \\ \uparrow & \uparrow & \uparrow & \uparrow & \uparrow & \uparrow & \uparrow & \uparrow & \uparrow & \uparrow \\ k & k & k & k & k & k & k & k & k & k \end{array}$

Apareo de secuencias

- ¿Qué invariante de ciclo tiene esta implementación?

$$
\begin{array}{lll}
I & \equiv & ordenado(a) \wedge ordenado(b) \wedge |c| = |a| + |b| \\
 & \wedge & ((0 \leq i \leq |a| \ \wedge \ 0 \leq j \leq |b| \ \wedge \ k = i + j) \\
 & \wedge_L & (mismos(subseq(a, 0, i) + +subseq(b, 0, j), subseq(c, 0, k)) \\
 & \wedge & ordenado(subseq(c, 0, k)))) \\
 & \wedge & i < |a| \ \rightarrow_L \ (\forall t : \mathbb{Z})(0 \leq t < j \rightarrow_L b[t] \leq a[i]) \\
 & \wedge & j < |b| \ \rightarrow_L \ (\forall t : \mathbb{Z})(0 \leq t < i \rightarrow_L a[t] \leq b[j])
\end{array}
$$

- ¿Qué función variante debería tener esta implementación?

$$fv = |a| + |b| - k$$

Apareo de secuencias

```
vector<int> merge(vector<int> &a, vector<int> &b) {
  vector<int> c(a.size()+b.size());
  int i = 0; // Para recorrer a
  int j = 0; // Para recorrer b
  int k = 0; // Para recorrer c

  while( k < c.size() ) {
    if( /*Si tengo que avanzar i */ ) {
      c[k++] = a[i++];
    } else if(/* Si tengo que avanzar j */) {
      c[k++] = b[j++];
    }
  }
  return c;
}
```

- ¿Cuándo tengo que avanzar i? Cuando j está fuera de rango ó cuando i y j están en rango y $a[i] < b[j]$
- ¿Cuándo tengo que avanzar j? Cuando no tengo que avanzar i

Apareo de secuencias

```
vector<int> merge(vector<int> &a, vector<int> &b) {
  vector<int> c(a.size()+b.size());
  int i = 0; // Para recorrer a
  int j = 0; // Para recorrer b
  int k = 0; // Para recorrer c

  while( k < c.size() ) {
    if( j>=b.size() || ( i<b.size() && a[i] < b[j] )) {
      c[k++] = a[i++];
    } else {
      c[k++] = b[j++];
    }
  }
  return c;
}
```

- Al terminar el ciclo, ¿ya está la secuencia c con los valores finales?

Apareo de secuencias

```
vector<int> merge(vector<int> &a, vector<int> &b) {
  vector<int> c(a.size()+b.size());
  int i = 0; // Para recorrer a
  int j = 0; // Para recorrer b
  int k = 0; // Para recorrer c

  while( k < c.size() ) {
    if( j>=b.size() || (i<b.size() && a[i] < b[j] )) {
      c[k++] = a[i++];
    } else {
      c[k++] = b[j++];
    }
  }
  return c;
}
```

- ¿Cuántes iteraciones realiza este programa en **peor caso** (como máximo)?
  - Realiza a lo sumo $|a| + |b|$ iteraciones

The welfare crook

- **Problema:** Dadas tres secuencias ordenadas, sabemos que hay al menos un elemento en común entre ellos. Encontrar los índices donde está al menos uno de estos elementos repetidos.
- Usamos iv, jv y kv para denotar las posiciones en las que las secuencias coinciden.
- proc $crook$(in $a, b, c : seq\langle\mathbb{Z}\rangle$, out $i, j, k : \mathbb{Z}$){

  Pre $\{ordenado(a) \ \wedge \ ordenado(b) \ \wedge \ ordenado(c)$
  $\wedge(\exists iv, jv, kv : \mathbb{Z})$
  $((0 \leq iv < |a| \wedge \ 0 \leq jv < |b| \wedge \ 0 \leq kv < |c|)$
  $\wedge_L \ a[iv] = b[jv] = c[kv])\}$

  Post $\{a[i] = b[j] = c[k] \wedge$
  $(0 \leq i < |a| \wedge \ 0 \leq j < |b| \wedge \ 0 \leq k < |c|)\}$

  }

The welfare crook

- $a = \begin{array}{|c|c|c|c|c|c|c|c|}\hline x\text{-}10 & ? & ? & x & ? & ? & ? & \dots \\ \hline \multicolumn{1}{c}{\uparrow} & \multicolumn{1}{c}{} & \multicolumn{1}{c}{} & \multicolumn{1}{c}{\uparrow} \\ \multicolumn{1}{c}{i} & \multicolumn{1}{c}{} & \multicolumn{1}{c}{} & \multicolumn{1}{c}{iv} \end{array}$
- $b = \begin{array}{|c|c|c|c|c|c|c|c|}\hline x\text{-}20 & ? & ? & ? & x & ? & ? & \dots \\ \hline \multicolumn{1}{c}{\uparrow} & \multicolumn{1}{c}{} & \multicolumn{1}{c}{} & \multicolumn{1}{c}{} & \multicolumn{1}{c}{\uparrow} \\ \multicolumn{1}{c}{j} & \multicolumn{1}{c}{} & \multicolumn{1}{c}{} & \multicolumn{1}{c}{} & \multicolumn{1}{c}{jv} \end{array}$
- $c = \begin{array}{|c|c|c|c|c|c|c|c|}\hline x\text{-}30 & ? & ? & ? & ? & ? & x & \dots \\ \hline \multicolumn{1}{c}{\uparrow} & \multicolumn{1}{c}{} & \multicolumn{1}{c}{} & \multicolumn{1}{c}{} & \multicolumn{1}{c}{} & \multicolumn{1}{c}{} & \multicolumn{1}{c}{\uparrow} \\ \multicolumn{1}{c}{k} & \multicolumn{1}{c}{} & \multicolumn{1}{c}{} & \multicolumn{1}{c}{} & \multicolumn{1}{c}{} & \multicolumn{1}{c}{} & \multicolumn{1}{c}{kv} \end{array}$
- ¿Cuál es el invariante de esta implementación?

$$I \equiv 0 \leq i \leq iv \;\wedge\; 0 \leq j \leq jv \;\wedge\; 0 \leq k \leq kv$$

- ¿Cuál es una función variante para esta implementación?

$$fv = (iv - i) + (jv - j) + (kv - k)$$

The welfare crook

- Comenzamos con $i = j = k = 0$, y vamos subiendo el valor de estas variables.

```
void welfareCrook(vector<int> &a, vector<int> &b, vector<int> &c,
    int &i, int &j, int &k) {
  i = 0, j = 0, k = 0;
  while( a[i] != b[j] || b[j] != c[k] ) {
    // Incrementar i, j o k!
  }
  // i=iv, j=jv, k=kv
}
```

The welfare crook

- ¿A cuál de los índices podemos incrementar?
- Alcanza con avanzar cualquier índice que no contenga al máximo entre $a[i]$, $b[j]$ y $c[k]$
- En ese caso, el elemento que no es el máximo no es el elemento buscado

```
i = 0, j = 0, k = 0;
while( a[i] != b[j] || b[j] != c[k] ) {
  if( a[i] < b[j] ) {
    i++;
  } else if( b[j] < c[k] ) {
    j++;
  } else {
    k++;
  }
}
```

The welfare crook

```
i = 0, j = 0, k = 0;
while( a[i] != b[j] || b[j] != c[k] ) {
  if( a[i] < b[j] ) {
    i++;
  } else if( b[j] < c[k] ) {
    j++;
  } else {
    k++;
  }
}
```

- ¿Por qué se preserva el invariante?
  1. $I \wedge B \wedge a[i] < b[j]$ implica $i < iv$, entonces es seguro avanzar i.
  2. $I \wedge B \wedge b[j] < c[k]$ implica $j < jv$, entonces es seguro avanzar j.
  3. $I \wedge B \wedge a[i] \geq b[j] \wedge b[j] \geq c[k]$ implica $k < kv$, por lo tanto es seguro avanzar k.

The welfare crook

```
void welfareCrook(vector<int> &a, vector<int> &b, vector<int> &c,
      int &i, int &j, int &k) {
  i = 0, j = 0, k = 0;
  while( a[i] != b[j] || b[j] != c[k] ) {
    if( a[i] < b[j] ) {
      i++;
    } else if( b[j] < c[k] ) {
      j++;
    } else {
      k++;
    }
  }
}
```

- ¿Cuántas iteraciones realiza este programa en **peor caso** (i.e. como máximo)?
  - Como máximo tiene que realizar $|a| + |b| + |c|$ iteraciones

Bibliografía

- Vickers et al. - Reasoned Programming
  - 6.6 - Sorted Merge (apareo)
- David Gries - The Science of Programming
  - Chapter 16 - Developing Invariants (Welfare Crook)