Avance 1 Proyecto

Cristian Garcia, Mariana Garcia

November 3 2025

1 Notas

1. En ocaciones usaremos la abreviación de CompactChainList como CCL para mayor legibilidad.
2. Teniendo en cuenta lo anterior, CCL se representa como: CCL $= \{(\mathrm{v}_0, l_0), (v_1, l_1), \ldots, (v_{n-1}, l_{n-1})\}$ *donde* : $v_i \in Element, \quad l_i \in int$ Cada par (v_i, l_i) representa un elemento y su cantidad en la lista.
3. En este documento, usamos $|CCL|$, esta se refiere a la suma de las longitudes de todos los bloques.

2 Operaciones

1. CompactChainList() : Construye una CompactChainList vacia.
   - Precondición: True
   - Postcondición: CCL $= \langle\rangle$
2. CompactChainList (vector<Element> &v) : Construye una CompactChainList con respecto a un vector.
   - Precondición:v $= \langle\rangle \vee \ v = \langle v_0, v_1, v_2, v_{n-1}\rangle$
   - Postcondición: CCL $= \langle\rangle \vee$ CCL $= \langle(v_0, l_0), (v_1, l_1), (v_2, l_2), \ldots, (v_{n-1}, l_{n-1})\rangle$
3. CompactChainList (CompactChainList &c) : Construye una CompactChainList como copia de los valores de otra. CompactChainList.
   - Precondición: CCL $= \langle\rangle \vee$ CCL $= \langle(v_0, l_0), (v_1, l_1), (v_2, l_2), \ldots, (v_{n-1}, l_{n-1})\rangle$
   - Postcondición: CCL $= \langle\rangle \vee$ CCL $= \langle(v_0, l_0), (v_1, l_1), (v_2, l_2), \ldots, (v_{n-1}, l_{n-1})\rangle$

4. res = searchElement (e) : Retorna la posición de la primera ocurrencia del elemento e.
   - Precondición: CCL = $\langle(v_0, l_0), (v_1, l_1), (v_2, l_2), \ldots, (v_{n-1}, l_{n-1})\rangle \wedge$ e $\in$ Element
   - Postcondición: if (e $\in CCL$) res = p; else res = -1
     donde p es la posición de la primera ocurrencia del elemento e.
5. set (pos, e) : Cambia el elemento ubicado en la posición pos por el nuevo elemento e.
   - Precondición: CCL = $\langle(v_0, l_0), (v_1, l_1), (v_2, l_2), \ldots, (v_{n-1}, l_{n-1})\rangle \wedge 0 \leq pos < |CCL| \wedge e \in$ Element
   - Postcondición: CCL = $\langle(v_0, l_0), \ldots, (v_e, l_e), \ldots, (v_{n-1}, l_{n-1})\rangle$
6. removeFirstOcurrence(e) : Elimina la primera ocurrencia del elemento.
   - Precondición: e $\in Element \wedge e \in CCL \wedge CCL = \langle(v_0, l_0), (v_1, l_1), (v_2, l_2), \ldots, (v_{n-1}, l_{n-1})\rangle$
   - Postcondición:CCL = $\langle(v_0, l_0), \ldots, (v_e, l_{e-1}), \ldots, (v_{n-1}, l_{n-1})\rangle$
7. removeAllOcurrences (e) : Elimina todas las ocurrencias del elemento.
   - Precondición: e $\in Element \wedge e \in CCL \wedge CCL = \langle(v_0, l_0), (v_1, l_1), (v_2, l_2), \ldots, (v_{n-1}, l_{n-1})\rangle$
   - Postcondición: Elimina todas las ocurrencias del elemento e.
8. removeBlockPosition (pos) : Elimina todo el bloque que se encuentre en dicha posición.
   - Precondición: CCL = $\langle(v_0, l_0), (v_1, l_1), (v_2, l_2), \ldots, (v_{n-1}, l_{n-1})\rangle \wedge 0 \leq pos < |CCL|$
   - Postcondición: Busca la posición y elimina el bloque que se encuentra ahí.
9. res = size() : que retorna el tamaño de la CCL
   - Precondición: CCL = $\langle(v_0, l_0), (v_1, l_1), (v_2, l_2), \ldots, (v_{n-1}, l_{n-1})\rangle \vee CCL = \langle\rangle$
   - Postcondición: if (CCL = $\langle\rangle$) res = 0; else res = $|CLL|$
10. insertElement (pos, e) : Inserta el elemento en la posición indicada.

- Precondición: CCL $= \langle (v_0, l_0), (v_1, l_1), (v_2, l_2), \ldots, (v_{n-1}, l_{n-1}) \rangle \wedge 0 \leq pos < |CCL| \wedge \ e \in Element$
- Postcondición: CCL $= \langle (v_0, l_0), \ldots, (v_e, l_{e+1}), \ldots, (v_{n-1}, l_{n-1}) \rangle$

11. res = getConsecutiveOcurrences (v) : devuelve cuántas veces ocurre la subsecuencia de forma consecutiva.
    - Precondición: CCL $= \langle (v_0, l_0), (v_1, l_1), (v_2, l_2), \ldots, (v_{n-1}, l_{n-1}) \rangle \wedge v = \langle v_0, v_1, v_2, \ldots, v_{n-1} \rangle$
    - Postcondición: $\begin{array}{ll} \text{if (v} \notin CCL) & \text{res} = 0 \\ \text{else} & \text{res} = \text{c} \end{array}$
      donde c es la cantidad de veces que ocurre la subsecuencia de forma consecutiva en CLL.
12. res = getIndexFirstConsecutiveOcurrence (v) : determina el índice en el que empieza la primera ocurrencia del vector en el CCL.
    - Precondición: CCL $= \langle (v_0, l_0), (v_1, l_1), (v_2, l_2), \ldots, (v_{n-1}, l_{n-1}) \rangle \wedge v = \langle v_0, v_1, v_2, \ldots, v_{n-1} \rangle \wedge \ v \in CCL$
    - Postcondición: res = i, donde i es el índice de la primera ocurrencia consecutiva de v en CCL.
13. res = getOcurrences (v) : devuelve cuántas veces ocurre la subsecuencia v, no necesariamente de forma consecutiva.
    - Precondición: CCL $= \langle (v_0, l_0), (v_1, l_1), (v_2, l_2), \ldots, (v_{n-1}, l_{n-1}) \rangle \wedge v = \langle v_0, v_1, v_2, \ldots, v_{n-1} \rangle$
    - Postcondición: $\begin{array}{ll} \text{if (v} \notin CCL) & \text{res} = 0 \\ \text{else} & \text{res} = \text{c} \end{array}$
      donde c es la cantidad de veces que ocurre la subsecuencia v en CLL (no necesariamente consecutiva).
14. res = getIndexFirstOcurrence (v) : Determina el índice en el que empieza la primera ocurrencia de v en CCL (no necesariamente consecutiva).
    - Precondición: CCL $= \langle (v_0, l_0), (v_1, l_1), (v_2, l_2), \ldots, (v_{n-1}, l_{n-1}) \rangle \wedge v = \langle v_0, v_1, v_2, \ldots, v_{n-1} \rangle \wedge \ v \in CCL$
    - Postcondición: res = i, donde donde i es el índice de la primera ocurrencia de v en CCL.
15. res = getLexicographicFusion (c) : Retorna la fusión lexicográfica entre el objeto y la instancia que se recibe

- Precondición: c $\in$ $CompactChainList \wedge CCL = \langle (v_0, l_0), (v_1, l_1), (v_2, l_2), \ldots, (v_{n-1}, l_{n-1}) \rangle \vee$ $CCL = \langle\rangle \vee \ c = \langle (v_0, l_0), (v_1, l_1), (v_2, l_2), \ldots, (v_{n-1}, l_{n-1}) \rangle \vee \ c = \langle\rangle$
- Postcondición:

if (CCL = $\langle\rangle \wedge \ c = \langle\rangle$)	res = $\langle\rangle$
else if (CCL = $\langle (v_0, l_0), \ldots, (v_{n-1}, l_{n-1}) \rangle \wedge \ c = \langle\rangle$)	res = CCL
else if (c = $\langle (w_0, m_0), \ldots, (w_{n-1}, m_{n-1}) \rangle \wedge \ CCL = \langle\rangle$)	res = c
else (c = $\langle (w_0, m_0), \ldots, (w_{n-1}, m_{n-1}) \rangle$ $\wedge \ CCL = \langle (v_0, l_0), \ldots, (v_{n-1}, l_{n-1}) \rangle$)	
res = fusion lexicografica de ambas instancias c y CCL, donde se agrupa de menor a mayor.	

- res = expand () : Retorna una lista de elementos con la secuencia expandida.
  - Precondición: CCL = $\langle\rangle \vee CCL = \langle (v_0, l_0), \ldots, (v_{n-1}, l_{n-1}) \rangle$
  - Postcondición:

if (CCL = $\langle\rangle$)	res = $\langle\rangle$
else	res = $\langle s_0, s_1, s_2, s_{n-1} \rangle$

    donde cada bloque (v_i, l_i) genera l_i elementos (v_i) consecutivos en res.
- res = operator < (oth) : Realiza la comparación lexicográfica de las secuencias representadas por dos instancias.
  - Precondición: oth $\in$ $CompactChainList \wedge CCL = \langle\rangle \vee CCL =$ $\langle (v_0, l_0), \ldots, (v_{n-1}, l_{n-1}) \rangle \vee \ oth = \langle\rangle$
  - Postcondición:
    res = true, cuando al momento de comparar la primera CCL es menor lexicograficamente que la segunda.
    res = false, cuando al momento de comparar la primera CCL es mayor o igual lexicograficamente que la segunda.
- res = operator ==(const CompactChainList &oth) const
  - Precondición: CCL = $\langle (v_0, l_0), (v_1, l_1), (v_2, l_2), \ldots, (v_{n-1}, l_{n-1}) \rangle \vee$ $CCL = \langle\rangle \vee oth = \langle (w_0, w_0), (w_1, w_1), (w_2, w_2), \ldots, (w_{n-1}, w_{n-1}) \rangle \vee$ $oth = \langle\rangle$
  - Postcondición:
    res = true, si el valor, tamaño y números de bloques son los mismos para ambas instancias.
    res = false, al existir alguna diferencia entre ambas instancias.
- res = operator ==(const Element &oth) const
  - Precondición: CCL = $\langle (v_0, l_0), (v_1, l_1), (v_2, l_2), \ldots, (v_{n-1}, l_{n-1}) \rangle \vee$ $CCL = \langle\rangle \wedge \ oth \in Element$
  - Postcondición:
    res = true, si el valor, tamaño y números de bloques son los

mismos para ambas instancias.
res = false, al existir alguna diferencia entre ambas instancias.

- res = operator ==(const int &oth) const

- Precondición: CCL = $\langle(v_0, l_0), (v_1, l_1), (v_2, l_2), \ldots, (w_{n-1}, w_{n-1})\rangle \vee$ $CCL = \langle\rangle \wedge$ $oth \in$N

- Postcondición:
res = true, si el valor, tamaño y números de bloques son los mismos para ambas instancias.
res = false, al existir alguna diferencia entre ambas instancias.

- res = operator +(oth): Realiza la fusión lexicográfica de dos instancias.

  - Precondición: oth $\in CompactChainList \wedge CCL = \langle(v_0, l_0), \ldots, (v_{n-1}, l_{n-1})\rangle \vee$ $CCL = \langle\rangle \vee$ $oth = \langle(w_0, w_0), \ldots, (w_{n-1}, w_{n-1})\rangle \vee$ $oth = \langle\rangle$

  - Postcondición:

if (CCL = $\langle\rangle \wedge$ $oth = \langle\rangle$)	res = $\langle\rangle$
else if (CCL = $\langle(v_0, l_0), \ldots, (v_{n-1}, l_{n-1})\rangle \wedge$ $oth = \langle\rangle$)	res = CCL
else if (oth = $\langle(v_0, l_0), \ldots, (v_{n-1}, l_{n-1})\rangle \wedge$ $CCL = \langle\rangle$)	res = oth
else (oth = $\langle(v_0, l_0), \ldots, (v_{n-1}, l_{n-1})\rangle$ $\wedge\ CCL = \langle(v_0, l_0), \ldots, (v_{n-1}, l_{n-1})\rangle$)	
res = fusion lexicografica de ambas instancias oth y CCL, donde se compara bloque por bloque para sacar el menor entre ambos.	

- res = operator[](const int pos) const: obtener el elemento que está en cierta posición de la secuencia.

  - Precondición: CCL = $\langle(v_0, l_0), \ldots, (v_{n-1}, l_{n-1})\rangle \vee$ $CCL = \langle\rangle \wedge$ $pos \in$ N $\wedge\ 0 \leq pos < tamaño$

  - Postcondición:
res = element, cuando existe elemento en el numero dado en pos.

- res = operator[](const int pos): obtener el elemento que está en cierta posición de la secuencia.

  - Precondición: CCL = $\langle(v_0, l_0), \ldots, (v_{n-1}, l_{n-1})\rangle \vee$ $CCL = \langle\rangle \wedge$ $pos \in$ N $\wedge\ 0 \leq pos < tamaño$

  - Postcondición:
res = referencia al elemento que dice la posición del numero pos en la secuencia.
CCL' = CCL, modificación en la referencia que nos da res.

- modifyAllOcurrences (one, two) : Cambia todas las ocurrencias del primer elemento (one) por el segundo elemento (two).

- Precondición: CCL = $\langle (v_0, l_0), (v_one, l_one), \ldots, (v_{n-1}, l_{n-1}) \rangle \wedge$ $one \in Element \wedge\ two \in Element$
- Postcondición: CCL = $\langle (v_0, l_0), (v_one, l_one), \ldots, (v_{n-1}, l_{n-1}) \rangle$

- push_front (e, num) : Agrega el elemento e, la cantidad de veces num al inicio de CCL. En caso de que v_0 sea igual a e, sumamos las cantidades.
  - Precondición: CCL = $\langle (v_0, l_0), (v_1, l_1), (v_2, l_2), \ldots, (v_{n-1}, l_{n-1}) \rangle \vee$ $CCL = \langle\rangle \wedge\ e \in Element \wedge\ num \in$ N
  - Postcondición:

if (CCL = $\langle\rangle$)	CCL = $\langle (e, num) \rangle$
else if ($v_0 = e$)	CCL = $\langle (v_0, l_0 + num), \ldots, (v_{n-1}, l_{n-1}) \rangle$
else	CCL = $\langle (e, num), (v_0, l_0), \ldots, (v_{n-1}, l_{n-1}) \rangle$

- push_back (e, num) : Agrega el elemento e, la cantidad de veces num al final de CCL. En caso de quev_{n-1} sea igual al elemento que queremos añadir, sumamos las cantidades.
  - Precondición: CCL = $\langle (v_0, l_0), (v_1, l_1), (v_2, l_2), \ldots, (v_{n-1}, l_{n-1}) \rangle \vee$ $CCL = \langle\rangle \wedge\ e \in Element \wedge\ num \in$ N
  - Postcondición:

if (CCL = $\langle\rangle$)	CCL = $\langle (e, num) \rangle$
else if ($v_{n-1} = e$)	CCL = $\langle (v_0, l_0), \ldots, (v_{n-1}, l_{n-1} + num) \rangle$
else	CCL = $\langle (v_0, l_0), \ldots, (v_{n-1}, l_{n-1}), (e, num) \rangle$

- sortVectorCCL (v) : Ordena lexicográficamente un vector de instancias de CCL.
  - Precondición: v = $\langle CCL_0, CCL_1, \ldots, CCL_{n-1} \rangle$, donde cada $CCL_i \in$ $CompactChainList$
  - Postcondición: v' = $\langle CCL'_0,\ CCL'_1, \ldots, CCL_{n-1}{}' \rangle$

- combineEquals() : Combina los Element que sean iguales y los ordena de menor a mayor.
  - Precondición: CCL = $(v_0, L_0), \ldots, (v_{n-1}, l_{n-1})$
  - Postcondición: CCL' = $(v'_0, L'_0), \ldots, (v_{n-1}{}', l_{n-1}{}')$,donde se sman todas las cantidades l_i de los v_i iguales y se ordenan de menor a mayor