Cuaderno de problemas
Fundamentos de algorítmia.

Verificación de algoritmos iterativos

Prof. Isabel Pita

5 de noviembre de 2020

Índice

1. Máximo de un vector y número de veces que aparece.

Número de máximos

Hemos desarrollado un juego online y tenemos en un fichero las puntuaciones de diversos jugadores. Ahora queremos saber cual ha sido la puntuación máxima obtenida y cuantos jugadores la han obtenido. Ha de tenerse en cuenta que el juego permite tener puntuaciones negativas cuando el jugador pierde más puntos de los que obtiene, pero ningún jugador termina con cero puntos.

Entrada

La entrada comienza con un valor entero que indica el número de casos de prueba. Cada caso de prueba consta de una línea con las puntuaciones obtenidas por los jugadores. Cada caso termina con el valor 0.

El número de puntuaciones es mayor que cero y no puede suponerse un límite superior. Las puntuaciones son números enteros diferentes de cero y se sabe que pueden almacenarse en una variable de tipo int.

Salida

Para cada caso de prueba se escribe en una línea el valor máximo y el número de veces que se repite separados por un carácter blanco.

Entrada de ejemplo

```
3
5 7 3 4 7 3 2 6 5 4 7 1 2 7 3 0
6 6 6 0
5 0
```

Salida de ejemplo

```
7 4
6 3
5 1
```

Autor: Isabel Pita

1.1. Especificación del problema

- La precondición del algoritmo (P_{ALG}), debe expresar las propiedades de los datos de entrada.
  - La función tiene un único dato de entrada que es el vector con los valores de las puntuaciones.
  - Para que el problema esté bien definido, el vector debe tener al menos un elemento para que exista un valor máximo. El máximo de un conjunto vacío de valores no está definido.
  - A los valores del vector no se les pide ninguna propiedad. Son números enteros cualesquiera.
- La postcondición del algoritmo (Q_{ALG}), debe expresar las propiedades de los datos de salida.
  - La función tiene dos datos de salida, el valor máximo del vector y el número de veces que aparece.
  - Para expresar el máximo del vector utilizamos la expresión que obtiene el máximo de una serie de valores.
  - Para contar el número de veces que aparece el máximo utilizamos la expresión que nos permite contar los valores que cumplen una propiedad. En este caso la propiedad es que los valores sean iguales que el máximo.

P_{ALG} : $\{v.size() > 0\}$

```
contMax(vector<int> v) dev {int maxi, int cont}
```

Q_{ALG} : $\{(\texttt{maxi} == \text{máx}\ \ k : 0 <= k < v.size() : \texttt{v[k]}) \wedge (\texttt{cont} == \#\ k : 0 \leq k < v.size() : v[k] == \texttt{maxi})\}$

1.2. Implementación.

```
void contMax (std::vector<int> const& v, int& maxi, int& cont) {
  maxi = v[0]; cont = 1; int i = 1;
  while (i < v.size()) {
    if (v[i] > maxi) {
      maxi = v[i];
      cont = 1;
    }
    else if (v[i] == maxi)
      ++cont
    ++i;
  }
}
```

Denominamos:

- B_{BUCLE} a la condición del bucle: i < v.size().
- B_{IF1}, a la primera condición del condicional: v[i] > maxi.
- B_{IF2}, a la segunda condición del condicional: v[i] == maxi.
- A_{BUCLE}, a las instrucciones del cuerpo del bucle.

1.3. Verificación.

- El programa tiene cuatro instrucciones secuenciales. Primero tres asignaciones y después un bucle.
- Nos centramos en la corrección del bucle.
  - Obtenemos la precondición del bucle (P_{BUCLE}) que expresa las propiedades que cumple el estado de ejecución del algoritmo antes de empezar a ejecutar el bucle. En este punto del programa tenemos que el vector de entrada no se ha modificado, por lo que seguirá cumpliendo las propiedades que indica la precondición del algoritmo sobre él. Los parámetros `maxi` y `cont` tienen los valores `v[0]` y 1 respectivamente y la variable local `i` toma el valor 1.

$$P_{BUCLE} : \{v.size() > 0 \wedge \texttt{maxi} == \texttt{v[0]} \wedge \texttt{cont} == 1 \wedge \texttt{i} == 1\}$$

- La postcondición del bucle (Q_{BUCLE}) coincide con la postcondición del algoritmo.

$$Q_{BUCLE} : \{(\texttt{maxi} == \text{máx}\ \ k : 0 <= k < v.size() : \texttt{v[k]}) \wedge (\texttt{cont} == \#\ k : 0 \leq k < v.size() : \texttt{v[k]} == \texttt{maxi})\}$$

- Para demostrar la corrección del bucle necesitamos obtener un predicado invariante que exprese lo que se está calculando en el bucle.
  - En el bucle se calculan dos valores: `maxi` y `cont`.
  - En cada vuelta del bucle, la variable `maxi` contiene el valor máximo de la parte del vector que ya se ha recorrido. Por lo tanto, cuando estamos en la vuelta i-ésima, el valor de `maxi` cumple:
$$\texttt{maxi} == \text{máx}\ \ k : 0 <= k < \texttt{i} : \texttt{v[k]}$$
  - En cada vuelta del bucle, la variable `cont` contiene el número de veces que se ha encontrado el valor de `maxi` en la parte del vector ya recorrida. Por lo tanto en la vuelta i-ésima se cumple:
$$\texttt{cont} == (\#\ k : 0 <= k < \texttt{i} : \texttt{v[k]} == \texttt{maxi})$$
  - Añadimos al invariante una condición sobre la variable de control del bucle que nos hará falta en la parte 3 de la demostración de corrección. Esta propiedad es obvia, ya que la variable de control empieza en 0 y en cada vuelta del bucle se incrementa en una unidad.
$$\texttt{i} <= v.size()$$
  - Observamos que el invariante expresa las propiedades pedidas en la postcondición del bucle pero referidas al intervalo del vector que ya se ha recorrido.
$$Inv : \{(\texttt{maxi} == \text{máx}\ \ k : 0 <= k < \texttt{i} : \texttt{v[k]}) \wedge (\texttt{cont} == \#\ k : 0 <= k < \texttt{i} : \texttt{v[k]} == \texttt{maxi}) \wedge \texttt{i} <= v.size()\}$$

- Para demostrar la terminación del bucle necesitamos definir una función cota. Esta es una función de las variables que intervienen en la condición del bucle que es positiva y decrece en cada vuelta del bucle. Observando el bucle, vemos que la variable `i` crece en cada vuelta, por lo tanto `-i` decrece en cada vuelta. Para que la función sea positiva le sumamos un valor mayor que el máximo que pueda tomar la variable `i`. La función cota elegida es :
$$t(i) = v.size() - \texttt{i} + 1$$

- Prueba de las 5 condiciones de corrección del bucle:
  1. $P_{BUCLE} \Rightarrow Inv$. El invariante se cumple al comenzar el bucle.
     - $P_{BUCLE} : \{v.size() > 0 \wedge \texttt{maxi} == \texttt{v[0]} \wedge \texttt{cont} == 1 \wedge \texttt{i} == 1\}$
     - La precondición del bucle indica que en el estado antes de comenzar el bucle se cumple `i` == 1 y `maxi` == `v[0]` por lo tanto, la parte del invariante: `maxi` == máx $k : 0 <= k <$ `i` : `v[k]` es cierta porque el rango de k solo admite el valor 0.
     - La precondición del bucle indica que en el estado antes de comenzar el bucle se cumple `i` == 1 y `cont` == 1, por lo tanto, la parte del invariante `cont` == $\#\ k : 0 <= k <$ `i` : `v[k]` == `maxi` es cierto porque el rango de k solo admite el valor 0 y `maxi` == `v[0]`.
     - La precondición del bucle indica que $v.size() > 0$ e `i` == 1, por lo tanto `i` <= $v.size()$.
  2. $\{Inv \wedge B_{BUCLE}\} A_{BUCLE} \{Inv\}$. Las instrucciones del bucle son correctas, porque mantienen el invariante. Tenemos que probar la corrección de las instrucciones del bucle. Planteamos la verificación de las instrucciones
     $Inv \wedge B_{BUCLE} : \{\texttt{maxi} == \text{máx}\ \ k : 0 <= k < \texttt{i} : \texttt{v[k]} \wedge \texttt{cont} == \#\ k : 0 <= k < \texttt{i} : \texttt{v[k]} == \texttt{maxi} \wedge \texttt{i} <= v.size() \wedge \texttt{i} < v.size()\}$

```
if (v[i] > maxi) {
  maxi = v[i];
  cont = 1;
}
else if (v[i] == maxi)
  ++cont
++i;
```

$Inv : \{(\texttt{maxi} = \text{máx}\ k : 0 <= k < \texttt{i} : \texttt{v[k]}) \wedge (\texttt{cont} = \#\ k : 0 <= k < \texttt{i} : \texttt{v[k]} == \texttt{maxi}) \wedge \texttt{i} <= v.size()\}$

- Tenemos dos instrucciones en secuencia. La primera es un condicional y la segunda un incremento.
- Obtenemos el predicado intermedio entre las dos instrucciones aplicando el axioma de asignación al incremento. Para ello sustituimos en el predicado Inv la variable i por i+1 y obtenemos el predicado R_1:
  $Inv \wedge B_{BUCLE} : \{(\texttt{maxi} == \text{máx}\ k : 0 <= k < \texttt{i} : \texttt{v[k]}) \wedge (\texttt{cont} == \#\ k : 0 <= k < \texttt{i} : \texttt{v[k]} == \texttt{maxi}) \wedge \texttt{i} <= v.size() \wedge \texttt{i} < v.size()\}$

```
if (v[i] > maxi) {
  maxi = v[i];
  cont = 1;
}
else if (v[i] == maxi)
  ++cont
```

  $R_1 : \{(\texttt{maxi} == \text{máx}\ k : 0 <= k < \texttt{i} + 1 : \texttt{v[k]}) \wedge (\texttt{cont} == \#\ k : 0 <= k < \texttt{i} + 1 : \texttt{v[k]} == \texttt{maxi}) \wedge \texttt{i} + 1 <= v.size()\}$

```
++i;
```

  $Inv : \{(\texttt{maxi} = \text{máx}\ k : 0 <= k < \texttt{i} : \texttt{v[k]}) \wedge (\texttt{cont} = \#\ k : 0 <= k < \texttt{i} : \texttt{v[k]} == \texttt{maxi}) \wedge \texttt{i} <= v.size()\}$
- Ahora debemos probar la corrección de la instrucción condicional. El condicional tiene tres partes, porque es una instrucción if(..) else if(..) else;. Planteamos las tres partes de la verificación:
  a) Primera parte, se cumple la condición v[i] > maxi.
  $Inv \wedge B_{BUCLE} \wedge B_{IF1} : \{(\texttt{maxi} == \text{máx}\ k : 0 <= k < \texttt{i} : \texttt{v[k]}) \wedge (\texttt{cont} == \#\ k : 0 <= k < \texttt{i} : \texttt{v[k]} == \texttt{maxi}) \wedge \texttt{i} <= v.size() \wedge \texttt{i} < v.size() \wedge \texttt{v[i]} > \texttt{maxi}\}$

```
maxi = v[i];
cont = 1;
```

  $R_1 : \{(\texttt{maxi} == \text{máx}\ k : 0 <= k < \texttt{i} + 1 : \texttt{v[k]}) \wedge (\texttt{cont} == \#\ k : 0 <= k < \texttt{i} + 1 : \texttt{v[k]} == \texttt{maxi}) \wedge \texttt{i} + 1 <= v.size()\}$

  b) Segunda parte, se cumple la segunda condición: v[i] == maxi.
  $Inv \wedge B_{BUCLE} \wedge \neg B_{IF1} \wedge B_{IF2} : \{(\texttt{maxi} == \text{máx}\ k : 0 <= k < \texttt{i} : \texttt{v[k]}) \wedge (\texttt{cont} == \#\ k : 0 <= k < \texttt{i} : \texttt{v[k]} == \texttt{maxi}) \wedge \texttt{i} <= v.size() \wedge \texttt{i} < v.size() \wedge \texttt{v[i]} <= \texttt{maxi} \wedge \texttt{v[i]} == \texttt{maxi}\}$

```
++cont;
```

  $R_1 : \{(\texttt{maxi} == \text{máx}\ k : 0 <= k < \texttt{i} + 1 : \texttt{v[k]}) \wedge (\texttt{cont} == \#\ k : 0 <= k < \texttt{i} + 1 : \texttt{v[k]} == \texttt{maxi}) \wedge \texttt{i} + 1 <= v.size()\}$

  c) Tercera parte. Si no se cumple ninguna de las dos condiciones anteriores, entonces no se ejecuta ninguna instrucción. Se observa que las condiciones anteriores aparecen negadas en la precondición porque si no se ejecuta ninguna acción es porque las condiciones son falsas.
  $Inv \wedge B_{BUCLE} \wedge \neg B_{IF1} \wedge \neg B_{IF2} : \{(\texttt{maxi} == \text{máx}\ k : 0 <= k < \texttt{i} : \texttt{v[k]}) \wedge (\texttt{cont} == \#\ k : 0 <= k < \texttt{i} : \texttt{v[k]} == \texttt{maxi}) \wedge \texttt{i} <= v.size() \wedge \texttt{i} < v.size() \wedge \texttt{v[i]} <= \texttt{maxi} \wedge \texttt{v[i]} \neq \texttt{maxi}\}$

```
;
```

  $R_1 : \{(\texttt{maxi} == \text{máx}\ k : 0 <= k < \texttt{i} + 1 : \texttt{v[k]}) \wedge (\texttt{cont} == \#\ k : 0 <= k < \texttt{i} + 1 : \texttt{v[k]} == \texttt{maxi}) \wedge \texttt{i} + 1 <= v.size()\}$
- Probamos la corrección de la primera parte, es una secuencia de dos instrucciones de asignación:
  $Inv \wedge B_{BUCLE} \wedge B_{IF1} : \{(\texttt{maxi} == \text{máx}\ k : 0 <= k < \texttt{i} : \texttt{v[k]}) \wedge (\texttt{cont} == \#\ k : 0 <= k < \texttt{i} : \texttt{v[k]} == \texttt{maxi}) \wedge \texttt{i} <= v.size() \wedge \texttt{i} < v.size() \wedge \texttt{v[i]} > \texttt{maxi}\}$

```
maxi = v[i];
cont = 1;
```

R_1 : {(maxi == máx k : 0 <= k < i + 1 : v[k]) ∧ (cont == # k : 0 <= k < i + 1 : v[k] == maxi) ∧ i + 1 <= $v.size()$}

Encontramos los dos predicados intermedios R_{10} y R_{11}, aplicando el axioma de asignación.:

$Inv \wedge B_{BUCLE} \wedge B_{IF1}$: {(maxi == máx k : 0 <= k < i : v[k]) ∧ (cont == # k : 0 <= k < i : v[k] == maxi) ∧ i <= $v.size()$ ∧ i < $v.size()$ ∧ v[i] > maxi}
R_{10} : {(v[i] == máx k : 0 <= k < i + 1 : v[k]) ∧ (1 == # k : 0 <= k < i + 1 : v[k] == v[i]) ∧ i + 1 <= $v.size()$}

```
maxi = v[i];
```

R_{11} : {(maxi == máx k : 0 <= k < i + 1 : v[k]) ∧ (1 == # k : 0 <= k < i + 1 : v[k] == maxi) ∧ i + 1 <= $v.size()$}

```
cont = 1;
```

R_1 : {(maxi == máx k : 0 <= k < i + 1 : v[k]) ∧ (cont == # k : 0 <= k < i + 1 : v[k] == maxi) ∧ i + 1 <= $v.size()$}

Si un estado cumple $Inv \wedge B_{BUCLE} \wedge B_{IF1}$ también cumple R_{10} porque en $Inv \wedge B_{BUCLE} \wedge B_{IF1}$ se cumple que v[i] > maxi y maxi == máx k : 0 <= k < i : v[k]. Por lo tanto v[i] es mayor que todos los elementos del vector entre 0 e i, y por lo tanto se cumple la primera parte de R_{10}. Por otra parte, como v[i] es estrictamente mayor que maxi no hay ningún elemento igual a v[i] entre 0 e i-1, por lo que el número de elementos entre 0 e i iguales a v[i] es sólo el propio v[i].

- Probamos la corrección de la segunda parte, es una instrucción de asignación, aplicamos el axioma de asignación y obtenemos el predicado R_{20}:
$Inv \wedge B_{BUCLE} \wedge \neg B_{IF1} \wedge B_{IF2}$: {(maxi == máx k : 0 <= k < i : v[k]) ∧ (cont == # k : 0 <= k < i : v[k] == maxi) ∧ i <= $v.size()$ ∧ i < $v.size()$ ∧ v[i] <= maxi ∧ v[i] == maxi}
R_{20} : {(maxi == máx k : 0 <= k < i + 1 : v[k]) ∧ (cont + 1 == # k : 0 <= k < i + 1 : v[k] == maxi) ∧ i + 1 <= $v.size()$}

```
++cont;
```

R_1 : {(maxi == máx k : 0 <= k < i + 1 : v[k]) ∧ (cont == # k : 0 <= k < i + 1 : v[k] == maxi) ∧ i + 1 <= $v.size()$}

Si un estado cumple $Inv \wedge B_{BUCLE} \wedge \neg B_{IF1} \wedge B_{IF2}$ también cumple R_{20} porque # k : 0 <= k < i+1 : v[k] == maxi) == (# k : 0 <= k < i : v[k] == maxi)+1 por ser v[i] == maxi. Como cont = (# k : 0 <= k < i : v[k] == maxi) en el estado por que se cumple $Inv \wedge B_{BUCLE} \wedge \neg B_{IF1} \wedge B_{IF2}$ se tiene cont + 1 = #k : 0 <= k < i + 1 : v[k] == maxi.

- Probamos la corrección de la tercera parte. Esta parte es vacía. Si ninguna de las dos condiciones anteriores es cierta entonces se debe cumplir la postcondición, es decir, el predicado R_{30} coincide con R_1:
$Inv \wedge B_{BUCLE} \wedge \neg B_{IF1} \wedge \neg B_{IF2}$: {(maxi == máx k : 0 <= k < i : v[k]) ∧ (cont == # k : 0 <= k < i : v[k] == maxi) ∧ i <= $v.size()$ ∧ i < $v.size()$ ∧ v[i] <= maxi ∧ v[i] ≠ maxi}
R_{30} : {(maxi == máx k : 0 <= k < i + 1 : v[k]) ∧ (cont == # k : 0 <= k < i + 1 : v[k] == maxi) ∧ i + 1 <= $v.size()$}

```
;
```

R_1 : {(maxi == máx k : 0 <= k < i + 1 : v[k]) ∧ (cont == # k : 0 <= k < i + 1 : v[k] == maxi) ∧ i + 1 <= $v.size()$}

Si un estado cumple $Inv \wedge B_{BUCLE} \wedge \neg B_{IF1} \wedge \neg B_{IF2}$ también cumple R_{30}, porque si v[i] < maxi no cambia el valor del máximo ni el contador del número de máximos por lo

que (máx $k : 0 <= k < \texttt{i} : \texttt{v[k]}) == (\text{máx } k : 0 <= k < \texttt{i} + 1 : \texttt{v[k]})$ y $(\# k : 0 <= k < \texttt{i} : \texttt{v[k]} == \texttt{maxi}) == (\# k : 0 <= k < \texttt{i} + 1 : \texttt{v[k]} == \texttt{maxi})$

3. $\{Inv \wedge \neg B_{BUCLE}\} \Rightarrow Q_{BUCLE}$. Cuando el bucle termina se cumple la postcondición.
   $\{Inv \wedge \neg B_{BUCLE}\} : \{(\texttt{maxi} == \text{máx } k : 0 <= k < \texttt{i} : \texttt{v[k]}) \wedge (\texttt{cont} == \# k : 0 <= k < \texttt{i} : \texttt{v[k]} == \texttt{maxi}) \wedge \texttt{i} <= v.size() \wedge \texttt{i} >= v.size()\}$

   La propiedad indica que se tiene que cumplir $\texttt{i} <= v.size()$ e $\texttt{i} >= v.size()$ la única posibilidad es que $\texttt{i} == v.size()$. Como el invariante garantiza que $\{(\texttt{maxi} == \text{máx } k : 0 <= k < \texttt{i} : \texttt{v[k]}) \wedge (\texttt{cont} == \# k : 0 <= k < \texttt{i} : \texttt{v[k]} == \texttt{maxi})$ se deduce $(\{\texttt{maxi} == \text{máx } k : 0 <= k < v.size() : v[\texttt{k}]) \wedge (\texttt{cont} == \#k : 0 <= k < v.size() : \texttt{v[k]} == \texttt{maxi})$

4. $\{Inv \wedge B_{BUCLE}\} \Rightarrow t > 0$. El invariante y la condición el bucle garantizan que la función cota es positiva. La función cota elegida es $t(i) = v.size() - \texttt{i} + 1$, como el invariate garantiza $\texttt{i} <= v.size()$ entonces $t(i) > 0$.

5. $\{Inv \wedge B_{BUCLE} \wedge t == T\} A_{BUCLE} \{t < T\}$. La función cota disminuye en cada vuelta del bucle.
   Planteamos la verificación:
   $\{Inv \wedge B_{BUCLE} \wedge t == T\} : \{(\texttt{maxi} == \text{máx } k : 0 <= k < \texttt{i} : \texttt{v[k]}) \wedge (\texttt{cont} == \# k : 0 <= k < \texttt{i} : \texttt{v[k]} == \texttt{maxi}) \wedge \texttt{i} <= v.size() \wedge \texttt{i} < v.size() \wedge v.size() - \texttt{i} + 1 == T\}$

```
if (v[i] > maxi) {
  maxi = v[i];
  cont = 1;
}
else if (v[i] == maxi)
  ++cont
++i;
```

   $\{v.size() - \texttt{i} + 1 < T\}$

   Hacemos el mismo estudio que en el paso 2 para la propiedad que queremos probar ahora. Aplicamos el axioma de asignación.
   $\{Inv \wedge B_{BUCLE} \wedge t == T\} : \{(\texttt{maxi} == \text{máx } k : 0 <= k < \texttt{i} : \texttt{v[k]}) \wedge (\texttt{cont} == \# k : 0 <= k < \texttt{i} : \texttt{v[k]} == \texttt{maxi}) \wedge \texttt{i} <= v.size() \wedge \texttt{i} < v.size() \wedge v.size() - \texttt{i} + 1 == T\}$
   $R_{50} : \{v.size() - (\texttt{i} + 1) + 1 < T\}$

```
if (v[i] > maxi) {
  maxi = v[i];
  cont = 1;
}
else if (v[i] == maxi)
  ++cont
```

   $R_{51} : \{v.size() - (\texttt{i} + 1) + 1 < T\}$

```
++i;
```

   $\{v.size() - \texttt{i} + 1 < T\}$
   La instrucción condicional no modifica ninguna variable libre del predicado R_{51} por lo que el predicado R_{50} coincide con R_{51}.
   Para probar que un estado que cumple $\{Inv \wedge B_{BUCLE} \wedge t == T\}$: cumple también $\{v.size() - (\texttt{i} + 1) + 1 < T\}$ basta con fijarnos en que partimos de que se cumple $v.size() - \texttt{i} + 1 == T$.

2. Segmento de longitud máxima. Todos con la selección.

Todos con la selección

Si la selección nacional gana el próximo partido en Málaga, habrá ganado 6 partidos seguidos. Hacia bastante tiempo que no tenía una racha ganadora seguida tan larga. Nuestro periodista encargado de seguir el partido del próximo sábado quiere saber cual ha sido la racha ganadora más larga de la selección en todo la historia, para poder contarlo durante el partido.

Para ello recopila los datos y le pide a un amigo informático que le ayude a analizarlos con un programa. Deben obtener el máximo número de partidos seguidos que ha conseguido ganar la selección, si ha ocurrido varias veces que se ganasen este número de partidos, y hace cuantos partidos que finalizó la última racha.

Requisitos de implementación.

Implementar una función que reciba en un vector los datos, y devuelva la información pedida en el problema.

Entrada

La entrada consta de una serie de casos de prueba. Cada caso de prueba tiene dos líneas. En la primera se indica el número de partidos jugados por la selección. En la segunda se indica la diferencia de goles entre los dos equipos. Un valor positivo indica que la selección ganó el partido, un valor cero indica que empató y un valor negativo que perdió.

Salida

Para cada caso de prueba se escribe en una línea el número máximo de partidos seguidos ganados, el número de veces que se ha ganado este número de partidos seguidos y el número de partidos jugados desde que finalizó la última racha ganadora.

Entrada de ejemplo

```
10
2 0 -3 1 1 0 2 1 -1 2
9
-1 3 1 2 0 1 -2 4 3
10
1 1 3 0 1 -1 4 3 1 2
3
-1 0 -1
```

Salida de ejemplo

```
2 2 2
3 1 5
4 1 0
0 0 3
```

Autor: Isabel Pita.

2.1. Especificación del problema

- La precondición del algoritmo (P_{ALG}), debe expresar las propiedades de los datos de entrada.
  - La función tiene un único dato de entrada que es el vector con los resultados de los partidos.
  - El vector puede tener cualquier número de elementos, o ser vacío.
  - A los valores del vector no se les pide ninguna propiedad. Son números enteros cualesquiera.
- La postcondición del algoritmo (Q_{ALG}), debe expresar las propiedades de los datos de salida.
  - La función tiene tres datos de salida, el máximo número de partidos seguidos ganados (`np`), el número de veces que se han ganado este número máximo de partidos (`nv`) y el número de partidos desde que se ganó por última vez este número máximo de partidos (`fg`).
  - Para expresar el máximo número de partidos seguidos ganados necesitamos poder referirnos a *los partidos seguidos ganados* para ello definimos un predicado que sea cierto si todos los valores del vector entre dos posiciones dadas son positivos. Definimos el predicado *secPositiva* que dado un vector y dos índices nos indica si las posiciones entre los dos índices son todas positivas.

    $$secPositiva(v, p, q) \equiv \forall k : p <= k < q : v[k] > 0$$

    Ahora definimos una expresión utilizando la expresión máx que calcule el máximo de todos los segmentos cuyas componentes sean positivas. Los segmentos los expresamos por medio de su principio y su final. La longitud del segmento se obtiene restando a la posición final la posición inicial. En la expresión que nos da el máximo de los valores, utilizamos dos variables ligadas p y q. p indica el principio del intervalo y q indica el final del intervalo: $[p \dots q)$. Se les exige que $p <= q$ ya que el intervalo puede ser vacío. La longitud de un intervalo está dada por la diferencia $q - p$

    $$\texttt{np} == \text{máx}\ \ p, q : 0 <= p <= q <= v.size() \wedge secPositiva(v, p, q) : q - p$$

  - Para contar el número de segmentos máximos, es decir de longitud np, que hay utilizamos la expresión que nos permite contar los valores que cumplen una propiedad. En este caso la propiedad es que el segmento que empieza en una posición y tiene longitud máxima esté formado todo él por valores positivos. La longitud máxima está dada por el valor de salida `np`.

    $$\texttt{nv} = \#p : 0 <= p < v.size() - \texttt{np} + 1 : secPositiva(v, p, p + \texttt{np})$$

  - El número de partidos desde que se gano por última vez el número máximo, es decir desde que se ganaron por última vez `np` partidos seguidos, lo expresamos indicando que desde la última vez que se ganó ese número de partidos no se han vuelto a ganar tantos seguidos. Damos la vuelta a este argumento y lo que expresamos es que si hay una racha con `np` partidos ganados, esa racha o es la última o ocurrió antes de la última. Denotamos por `fg` es el número de partidos desde que terminó la última racha, que es el valor que devuelve la función. Observamos que $v.size() - \texttt{fg}$ es el elemento siguiente en el vector al final de la última racha y que la última racha comienza en $v.size() - \texttt{fg} - \texttt{np}$. El siguiente predicado expresa que si hay una racha de longitud `np` su comienzo debe ser igual o anterior al comienzo del último.

    $$\forall j : 0 \leq j < v.size() - \texttt{np} \wedge secPositiva(v, j, j + \texttt{np}) : j \leq v.size() - \texttt{fg} - \texttt{np}$$

    Con este predicado indicamos que si hay una racha con `np` partidos ganados esta empieza en $v.size() - \texttt{fg} - \texttt{np}$ o antes, pero no se garantiza que empiece exactamente en ese momento. para garantizar que la última racha empieza en $v.size() - \texttt{fg} - \texttt{np}$ necesitamos otro predicado:

    $$secPositiva(v, v.size() - \texttt{fg} - \texttt{np}, v.size() - \texttt{fg})$$

Con esto la especificación queda:

P_{ALG} : $\{v.size() \geq 0\}$

```
segmentoMaximo(vector<int> v) dev {int np, int nv, int fg}
```

Q_{ALG} : $\{(np == \text{máx}\ p,q : 0 <= p <= q <= v.size() \wedge secPositiva(v,p,q) : q-p) \wedge$
$(nv = \#p : 0 <= p < v.size() - \texttt{np} + 1 : secPositiva(v,p,p+\texttt{np})) \wedge$
$(\forall j : 0 \leq j < v.size() - \texttt{np} \wedge secPositiva(v,j,j+\texttt{np}) : j \leq v.size() - \texttt{fg} - \texttt{np}) \wedge$
$(secPositiva(v, v.size() - \texttt{fg} - \texttt{np}, v.size() - \texttt{fg}))\}$

2.2. Implementación.

```
struct tSol {
    int ganados;
    int veces;
    int ultimosPerdidos;
};

tSol resolver(std::vector<int> const& v){
    int longMax = 0; int ultMax = -1; int numVeces = 0; int longAct = 0;
    int i = 0;
    while (i < v.size()) {
        if (v[i] > 0) { // El elemento continua la racha
            ++longAct;
            if (longMax < longAct) { // Mejora la racha anterior
                longMax = longAct;
                ultMax = i;
                numVeces = 1;
            }
            else if (longMax ==longAct) { // Iguala la racha anterior
                ++numVeces;
                ultMax = i;
            }
        }
        else longAct = 0; // Se rompe la racha
        ++i;
    }
    return {longMax, numVeces, (int)v.size() - ultMax - 1};
}
```

Denominamos:

- B_{BUCLE} a la condición del bucle: i < v.size().
- B_{IF11}, a la condición del primer condicional: v[i] > 0.
- B_{IF12}, a la negación de la condición del primer condicional: v[i] $\leq$ 0.
- B_{IF21}, a la primera condición del condicional que está dentro del primer condicional: longMax < longAct.
- B_{IF22}, a la segunda condición del condicional que está dentro del primer condicional: longMax == longAct.
- B_{IF23}, a la negación de las dos condiciones del condicional que está dentro del primer condicional.
- A_{BUCLE}, a las instrucciones del cuerpo del bucle.

2.3. Verificación.

- El programa tiene siete instrucciones secuenciales. Primero cinco asignaciones, después un bucle y despues una instrucción `return`. Esta instrucción hay que tenerla en cuenta porque en ella se hace la asignación de los valores de salida de la función a las variables que hemos utilizado en la especificación para denotar dicha la salida.

- Nos centramos en la corrección del bucle.

  - Obtenemos la precondición del bucle (P_{BUCLE}) que expresa las propiedades que cumple el estado de ejecución del algoritmo antes de empezar a ejecutar el bucle. En este punto del programa tenemos que el vector de entrada no se ha modificado, por lo que seguirá cumpliendo las propiedades que indica la precondición del algoritmo sobre él. Las variables locales toman los valores que se indican en P_{BUCLE}:

    $$P_{BUCLE} : \{v.size() > 0 \land \texttt{longMax} == 0 \land \texttt{ultMax} == -1 \land \texttt{numVeces} == 0 \land \texttt{longAct} == 0 \land i == 0\}$$

  - La postcondición del bucle (Q_{BUCLE}) debemos obtenerla aplicando el axioma de asignación a la postcondición del algoritmo. La asignación que realizamos en la instrucción de retorno es. `np = longMax; nv = numVeces; fg = v.size() - ultMax - 1;`.

    $$\begin{aligned} Q_{BUCLE} : \{&(longMax == \text{máx}\ \ p,q : 0 <= p <= q <= v.size() \land secPositiva(v,p,q) : q-p) \land \\ &(numVeces = \#p : 0 <= p < v.size() - \texttt{longMax} + 1 : secPositiva(v,p,p+\texttt{longMax})) \land \\ &(\forall j : 0 \leq j < v.size() - \texttt{longMax} \land secPositiva(v,j,j+\texttt{longMax}) : \\ &\qquad j \leq v.size() - (v.size() - \texttt{ultMax} - 1) - \texttt{longMax}) \land \\ &(secPositiva(v, v.size() - (v.size() - \texttt{ultMax} - 1) - \texttt{longMax}, v.size() - (v.size() - \texttt{ultMax} - 1)))\} \end{aligned}$$

    ```
    np = longMax;
    nv = numVeces;
    fg = v.size() - ultMax - 1;
    ```

    $$\begin{aligned} Q_{ALG} : \{&(\texttt{np} == \text{máx}\ \ p,q : 0 <= p <= q <= v.size() \land secPositiva(v,p,q) : q-p) \land \\ &(\texttt{nv} = \#p : 0 <= p < v.size() - \texttt{np} + 1 : secPositiva(v,p,p+\texttt{np})) \land \\ &(\forall j : 0 \leq j < v.size() - \texttt{np} \land secPositiva(v,j,j+\texttt{np}) : j \leq v.size() - \texttt{fg} - \texttt{np}) \land \\ &(secPositiva(v, v.size() - \texttt{fg} - \texttt{np}, v.size() - \texttt{fg}))\} \end{aligned}$$

    La postcondición del bucle se puede simplificar un poco:

    $$\begin{aligned} Q_{BUCLE} : \{&(\texttt{longMax} == \text{máx}\ \ p,q : 0 <= p <= q <= v.size() \land secPositiva(v,p,q) : q-p) \land \\ &(\texttt{numVeces} = \#p : 0 <= p < v.size() - \texttt{longMax} + 1 : secPositiva(v,p,p+\texttt{longMax})) \land \\ &(\forall j : 0 \leq j < v.size() - \texttt{longMax} \land secPositiva(v,j,j+\texttt{longMax}) : j \leq \texttt{ultMax} + 1 - \texttt{longMax}) \land \\ &(secPositiva(v, \texttt{ultMax} + 1 - \texttt{longMax}, \texttt{ultMax} + 1))\} \end{aligned}$$

- Para demostrar la corrección del bucle necesitamos obtener un predicado invariante que exprese lo que se está calculando en el bucle.

  - En el bucle se calculan los valores de `longAct`, `longMax`, `ultMax`, y `numVeces`. Además se va modificando la variable de control `i`.
  - En cada vuelta del bucle, la variable `longMax` contiene la longitud máxima encontrada en la parte del vector que ya se ha recorrido. Por lo tanto, cuando estamos en la vuelta i-ésima, el valor de `longMax` cumple:

    $$\texttt{longMax} == \text{máx}\ \ p,q : 0 <= p <= q <= i \land secPositiva(v,p,q) : q-p$$

  - En cada vuelta del bucle, la variable `numVeces` contiene el número de veces que se ha encontrado una secuencia de `longMax` valores positivos en la parte del vector ya recorrida. Por lo tanto en la vuelta i-ésima se cumple:

    $$\texttt{numVeces} = \#p : 0 <= p < i - \texttt{longMax} + 1 : secPositiva(v,p,p+\texttt{longMax}$$

  - El valor `ultMax` debe tener la última posición del último segmento encontrado con longitud `longMax` en la parte ya recorrida del vector. Por lo tanto debe cumplir:

    $$\begin{aligned} &(\forall j : 0 \leq j < i - \texttt{longMax} \land secPositiva(v,j,j+\texttt{longMax}) : j \leq \texttt{ultMax} + 1 - \texttt{longMax}) \land \\ &secPositiva(v, \texttt{ultMax} + 1 - \texttt{longMax}, \texttt{ultMax} + 1)) \end{aligned}$$

- Debemos también indicar el valor que lleva la variable `longAct`. En cada vuelta del bucle esta variable lleva la longitud de la máxima secuencia de valores positivos que termina en la posición `i` indicada por la variable de control del bucle. Utilizamos la expresión que nos da el máximo de una serie de valores. Observad que en esta ocasión la variable ligada se refiere a la longitud del segmento, no a la posición en el vector

$$\texttt{longAct} == \text{máx}\, k : 0 \leq k < v.size() : secPositiva(v, i-k, i) : k.$$

- Añadimos al invariante una condición sobre la variable de control del bucle que nos hará falta en la parte 3 de la demostración de corrección. Esta propiedad es obvia, ya que la variable de control empieza en 0 y en cada vuelta del bucle se incrementa en una unidad.

$$\texttt{i} <= v.size()$$

- El invariante se obtiene como la conjunción de todas laspropiedades anteriores. Observamos que expresa las propiedades pedidas en la postcondición del bucle pero referidas al intervalo del vector que ya se ha recorrido. Además debe incluirse el valor de la variable `longAct` que no aparece en la postcondición pero que es va calculando en el bucle. El valor de esta variable es necesario en la prueba de la corrección.

- Para demostrar la terminación del bucle necesitamos definir una función cota. Esta es una función de las variables que intervienen en la condición del bucle que es positiva y decrece en cada vuelta del bucle. Observando el bucle, vemos que la variable `i` crece en cada vuelta, por lo tanto `-i` decrece en cada vuelta. Para que la función sea positiva le sumamos un valor mayor que el máximo que pueda tomar la variable `i`. La función cota elegida es :

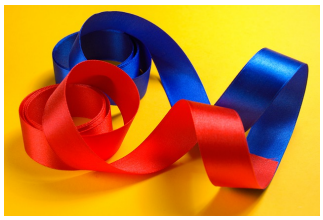
$$t(i) = v.size() - \texttt{i} + 1$$

- De las 5 condiciones de corrección del bucle,

  - La primera se basa en que los predicados del invariante son de rango vacío con los valores de la precondición del bucle.
  - La segunda condición es muy larga de probar y se sale del ámbito del curso.
  - La tercera condición se basa en que $\{Inv \wedge \neg B_{BUCLE}\} \Rightarrow \{\texttt{i} == v.size()\}$ igual que ocurría en el primer ejemplo y por lo tanto los predicados de la postcondición se obtienen a partir de los predicados del invariante.
  - las dos últimas se prueban de manera similar a la prueba realizada en el primer ejemplo. .

3. Algoritmo de partición. Lazos de Colores

Cintas de colores



Tengo una caja llena de cintas de colores rojas, azules y verdes. Las he estado midiendo y ahora quiero ordenarlas por colores, para poder saber facilmente las longitudes que tengo de cada color.

Dada una lista de todas las cintas con sus longitudes, sin ningún orden, debes ordenarlas de forma que al principio de la lista aparezcan las cintas azules, a continuación las verdes y por último las rojas. Dentro de cada grupo no es necesario que las cintas aparezcan en un orden determinado.

Requisitos de implementación.

Se debe implementar una función que reciba un vector con las cintas identificadas por su color y su longitud y modifique el vector para dejar al principio las cintas azules, en el medio las cintas verdes y por último las cintas rojas. La función devolverá dos índices indicando donde empiezan y terminan las cintas verdes. El coste de la función debe ser lineal en el número de cintas.

Debe emplearse el algoritmo de partición con dos índices.

Entrada

La entrada consta de una serie de casos de prueba. Cada caso de prueba consta de dos líneas. En la primera se indica el número n de cintas. En la siguiente se muestra el color de cada cinta seguido de su longitud.

El número de cintas es mayor o igual que 0 y menor que 300.000. Los colores de las cintas se identifican por su primer carácter: `a` para las cintas azules, `v` para las cintas verdes y `r` para las cintas rojas. Las longitudes son números enteros positivos.

Salida

Para cada caso de prueba se escriben tres líneas. La primera comienza con `Azules:` seguido de las longitudes de las cintas azules. La segunda comienza con `Verdes:` seguido de las longitudes de las cintas verdes. La tercera comienza con `Rojas:` seguido de las longitudes de las cintas rojas.

Para poder comparar tu salida con la del juez las longitudes se mostrarán en orden de menor a mayor, aunque esto no debe alterar el algoritmo empleado para resolver el problema, sino solo la salida de los datos.

Entrada de ejemplo

```
6
v 5 r 1 a 3 a 7 v 3 r 6
3
r 4 r 7 r 1
5
a 6 a 2 v 1 v 3 a 5
4
r 5 r 7 a 9 a 3
```

3.1. Especificación del problema

- La precondición del algoritmo (P_{ALG}), debe expresar las propiedades de los datos de entrada.
  - La función tiene un único dato de entrada que es el vector con las características de las cintas.
  - El vector puede tener cualquier número de elementos, o ser vacío.
  - Los valores del vector pueden ser de tres colores solamente. Por lo tanto indicaremos que la primera componente del elemento debe ser `'a'`, `'v'` o `'r'`. Las longitudes de la cintas pueden ser números enteros cualesquiera.
- La postcondición del algoritmo (Q_{ALG}), debe expresar las propiedades de los datos de salida.
  - La función tiene dos datos de salida: `p`, y `q`, que se corresponden con posiciones del vector. La posición `p` indica que todo lo que queda a su izquierda debe ser de color azul y la posición `q` indica que todo lo que queda a su derecha debe ser de color rojo. Lo que hay ente ambas posiciones debe ser de color verde.

    Podemos expresar esta propiedad mediante tres predicados:

    $$\forall j : 0 <= j < \texttt{p} : v[j].first == \texttt{'a'}$$
    $$\forall j : \texttt{p} <= j <= \texttt{q} + 1 : v[j].first == \texttt{'v'}$$
    $$\forall j : \texttt{q} < j <= v.size() : v[j].first == \texttt{'r'}$$

    Para que el problema esté bien especificado deberíamos añadir a la postcondición, que el vector obtenido es una permutación del vector de entrada. Con esto garantizamos que los valores del vector cambian de posición dentro del vector, pero ni se eliminan ni se crean nuevos valores.
- Nos centramos en las tres primeras propiedades y probamos que nuestro programa las cumple.

  $P_{ALG} : \{v.size() \geq 0 \wedge \forall k : 0 \leq k < v.size() : v[k].first \ \ in \ \{\texttt{'a'}, \texttt{'v'}, \texttt{'r'}\}\}$

  ```
  particion(vector<int> v) dev {int p, int q}
  ```

  $Q_{ALG} : \{(\forall j : 0 <= j < \texttt{p} : v[j].first == \texttt{'a'}) \wedge (\forall j : \texttt{p} <= j <= \texttt{q} + 1 : v[j].first == \texttt{'v'})$
  $\wedge (\forall j : \texttt{q} < j <= v.size() : v[j].first == \texttt{'r'})\}$

3.2. Implementación.

```
using psi = std::pair<char, int>;

void particion (std::vector<psi> & v, int &p, int &q) {
    p = 0; q = (int)v.size()-1; int k = 0;
    while (k ≤ q ) {
        if (v[k].first == 'v') ++k;
        else if ( v[k].first == 'a')  {
            std::swap(v[p],v[k]);
            ++p; ++k;
        }
        else {
            std::swap(v[q],v[k]);
            --q;
        }
    }
}
```

Denominamos:

- B_{BUCLE} a la condición del bucle: k <= q.
- B_{IF1}, a la condición del primer condicional: v[k].first == 'v'.
- B_{IF2}, a la negación de la condición del primer condicional: v[k].first == 'a'.
- A_{BUCLE}, a las instrucciones del cuerpo del bucle.

3.3. Verificación.

- El programa tiene cuatro instrucciones secuenciales. Primero tres asignaciones, y después un bucle.
- Nos centramos en la corrección del bucle.
  - Obtenemos la precondición del bucle (P_{BUCLE}) que expresa las propiedades que cumple el estado de ejecución del algoritmo antes de empezar a ejecutar el bucle. En este punto del programa tenemos que el vector de entrada no se ha modificado, por lo que seguirá cumpliendo las propiedades que indica la precondición del algoritmo sobre él. Los parámetros de salida y las variables locales toman los valores que se indican en P_{BUCLE}:

$$P_{BUCLE} : \{v.size() \geq 0 \wedge \forall k : 0 \leq k < v.size() : v[k].first \ \ in \ \{'\mathtt{a}','\mathtt{v}','\mathtt{r}'\} \\ \wedge \mathtt{p} == 0 \wedge \mathtt{q} == v.size() - 1 \wedge \mathtt{k} == 0\}$$

  - La postcondición del bucle (Q_{BUCLE}) coincide con la postcondición del algoritmo, ya que no hay ninguna instrucción después del bucle.

$$Q_{ALG} : \{(\forall j : 0 <= j < \mathtt{p} : v[j].first == '\mathtt{a}') \wedge (\forall j : \mathtt{p} <= j <= \mathtt{q}+1 : v[j].first == '\mathtt{v}') \\ \wedge (\forall j : \mathtt{q} < j <= v.size() : v[j].first == '\mathtt{r}')\}$$

- Para demostrar la corrección del bucle necesitamos obtener un predicado invariante que exprese lo que se está calculando en el bucle.
  - En el bucle se modifican los valores del vector intercambiando los valores de algunas componentes. También se modifican los valores de las variables `p`, `k` y `q`.
  - En cada vuelta del bucle, se cumple la propiedad que indica que los valores a la izquierda de `p` son de color azul, que los valores entre `p` y `k` son de color verde y que los valores a la derecha de `q` son de color rojo.
  - Por lo tanto proponemos como invariante

$$Inv : \{(\forall j : 0 <= j < \mathtt{p} : v[j].first == '\mathtt{a}') \wedge (\forall j : \mathtt{p} <= j < \mathtt{k} : v[j].first == '\mathtt{v}') \\ \wedge (\forall j : \mathtt{q} < j <= v.size() : v[j].first == '\mathtt{r}') \wedge 0 \leq \mathtt{p} \leq \mathtt{k} \leq \mathtt{q}+1 < v.size()\}$$

    Añadimos los límites de las variables para que pueda realizarse la prueba formal de la corrección.

- Para demostrar la terminación del bucle necesitamos definir una función cota. Esta es una función de las variables que intervienen en la condición del bucle que es positiva y decrece en cada vuelta del bucle. Observando el bucle, vemos que la variable `k` crece solo en las dos primeras partes de la instrucción condicional, mientras que la variable `q` decrece en la parte en que no crece `k`. Por lo tanto proponemos una función, que dependa de `k` y `q`. que sea positiva y decrezca. La función cota elegida es :

$$t(k, q) = q - k + 1$$

- De las 5 condiciones de corrección del bucle,
  - $P_{BUCLE} \Rightarrow Inv$. El invariante se cumple al comenzar el bucle.
    - $P_{BUCLE} : \{v.size() \geq 0 \wedge \forall k : 0 \leq k < v.size() : v[k].first \ \ in \ \{'\mathtt{a}','\mathtt{v}','\mathtt{r}'\}$ $\wedge \mathtt{p} == 0 \wedge \mathtt{q} == v.size() - 1 \wedge \mathtt{k} == 0\}$
    - La primera parte del invariante se cumple porque en un estado en que `p` $== 0$ el predicado $\forall j : 0 <= j < p : v[j].first == '\mathtt{a}'$ se cumple porque el rango del cuantificador universal es vacío y por lo tanto el predicado es cierto.
    - La segunda parte del invariante se cumple porque en un estado que cumple `p` $== 0$ y `k` $== 0$ el predicado $\forall j : p <= j < k : v[j].first == '\mathtt{v}'$ se cumple porque el rango del cuantificador universal es vacío.
    - La tercera parte del invariante se cumple porque en un estado que cumple `q` $== v.size()-1$ el predicado $\forall j : q < j <= v.size() : v[j].first == '\mathtt{r}'$ es cierto porque el rango del cuantificador universal es vacío.

- La última parte del invariante $0 \leq p \leq k \leq q+1 < v.size()$ se cumple con los valores asignados a los parámetros y variables locales.

- $\{Inv \wedge B_{BUCLE}\} A_{BUCLE} \{Inv\}$. Las instrucciones del bucle son correctas, porque mantienen el invariante. Tenemos que probar la corrección de las instrucciones del bucle. Planteamos la verificación de las instrucciones.

  $Inv \wedge B_{BUCLE} : \{(\forall j : 0 <= j < \mathtt{p} : v[j].first == \mathtt{'a'}) \wedge (\forall j : \mathtt{p} <= j < \mathtt{k} : v[j].first == \mathtt{'v'})$
  $\wedge(\forall j : \mathtt{q} < j <= v.size() : v[j].first == \mathtt{'r'}) \wedge 0 \leq \mathtt{p} \leq \mathtt{k} \leq \mathtt{q}+1 < v.size() \wedge \mathtt{k} <= \mathtt{q}\}$

  ```
  if (v[k].first == 'v') ++k;
  else if ( v[k].first == 'a')  {
       std::swap(v[p],v[k]);
       ++p; ++k;
  }
  else {
       std::swap(v[q],v[k]);
       --q;
  }
  ```

  $Inv : \{(\forall j : 0 <= j < \mathtt{p} : v[j].first == \mathtt{'a'}) \wedge (\forall j : \mathtt{p} <= j < \mathtt{k} : v[j].first == \mathtt{'v'})$
  $\wedge(\forall j : \mathtt{q} < j <= v.size() : v[j].first == \mathtt{'r'}) \wedge 0 \leq \mathtt{p} \leq \mathtt{k} \leq \mathtt{q}+1 < v.size()\}$

  La instrucción del bucle es un condicional con tres partes. Planteamos la verificación de cada una de las partes:

  1. Primera condición del bucle cierta.

     $Inv \wedge B_{BUCLE} \wedge B_{IF1} : \{(\forall j : 0 <= j < \mathtt{p} : v[j].first == \mathtt{'a'}) \wedge (\forall j : \mathtt{p} <= j < \mathtt{k} : v[j].first == \mathtt{'v'})$
     $\wedge(\forall j : \mathtt{q} < j <= v.size() : v[j].first == \mathtt{'r'}) \wedge 0 \leq \mathtt{p} \leq \mathtt{k} \leq \mathtt{q}+1 < v.size() \wedge \mathtt{k} <= \mathtt{q} \wedge$
     $v[k].first == \mathtt{'v'}\}$

     ```
     ++k;
     ```

     $Inv : \{(\forall j : 0 <= j < \mathtt{p} : v[j].first == \mathtt{'a'}) \wedge (\forall j : \mathtt{p} <= j < \mathtt{k} : v[j].first == \mathtt{'v'})$
     $\wedge(\forall j : \mathtt{q} < j <= v.size() : v[j].first == \mathtt{'r'}) \wedge 0 \leq \mathtt{p} \leq \mathtt{k} \leq \mathtt{q}+1 < v.size()\}$

     Aplicamos el axioma de asignación para la instrucción `k = k + 1` y obtenemos el predicado R_{10}. Cada aparición de `k` en el predicado Inv la sustituimos por `k + 1`.

     $Inv \wedge B_{BUCLE} \wedge B_{IF1} : \{(\forall j : 0 <= j < \mathtt{p} : v[j].first == \mathtt{'a'}) \wedge (\forall j : \mathtt{p} <= j < \mathtt{k} : v[j].first == \mathtt{'v'})$
     $\wedge(\forall j : \mathtt{q} < j <= v.size() : v[j].first == \mathtt{'r'}) \wedge 0 \leq \mathtt{p} \leq \mathtt{k} \leq \mathtt{q}+1 < v.size() \wedge \mathtt{k} <= \mathtt{q} \wedge$
     $v[k].first == \mathtt{'v'}\}$

     $R_{10} : \{(\forall j : 0 <= j < \mathtt{p} : v[j].first == \mathtt{'a'}) \wedge (\forall j : \mathtt{p} <= j < \mathtt{k}+1 : v[j].first == \mathtt{'v'})$
     $\wedge(\forall j : \mathtt{q} < j <= v.size() : v[j].first == \mathtt{'r'}) \wedge 0 \leq \mathtt{p} \leq \mathtt{k}+1 \leq \mathtt{q}+1 < v.size()\}$

     ```
     ++k;
     ```

     $Inv : \{(\forall j : 0 <= j < \mathtt{p} : v[j].first == \mathtt{'a'}) \wedge (\forall j : \mathtt{p} <= j < \mathtt{k} : v[j].first == \mathtt{'v'})$
     $\wedge(\forall j : \mathtt{q} < j <= v.size() : v[j].first == \mathtt{'r'}) \wedge 0 \leq \mathtt{p} \leq \mathtt{k} \leq \mathtt{q}+1 < v.size()\}$

     Vemos que si un estado de ejecución cumple $Inv \wedge B_{BUCLE} \wedge B_{IF1}$ entonces también cumple R_{10}.

     - La parte del predicado correspondiente a los valores azules y rojos se cumple trivialmente, ya que se pide lo mismo en el predicado $Inv \wedge B_{BUCLE} \wedge B_{IF1}$ que en el predicado R_{10}.
     - Respecto a la parte del predicado correspondiente a los valores verdes, vemos que en $Inv \wedge B_{BUCLE} \wedge B_{IF1}$ tenemos que $(\forall j : \mathtt{p} <= j < \mathtt{k} : v[j].first == \mathtt{'v'})$ y además se cumple $v[k].first == \mathtt{'v'}$ por ser la condición del condicional cierta. De estos dos predicados se deduce la condición de R_{10} respecto a los lazos verdes: $\forall j : \mathtt{p} <= j < \mathtt{k}+1 : v[j].first == \mathtt{'v'}$
     - Quedan por probar las últimas desigualdades de R_{10}. En el predicado $Inv \wedge B_{BUCLE} \wedge B_{IF1}$ tenemos que $0 \leq \mathtt{p} \leq \mathtt{k} \leq \mathtt{q}+1 < v.size() \wedge \mathtt{k} <= \mathtt{q}$, de donde se deduce $0 \leq \mathtt{p} \leq \mathtt{k}+1 \leq \mathtt{q}+1 < v.size()$.

  2. Segunda condición del bucle cierta.

     $Inv \wedge B_{BUCLE} \wedge \neg B_{IF1} \wedge B_{IF2} : \{(\forall j : 0 <= j < \mathtt{p} : v[j].first == \mathtt{'a'}) \wedge (\forall j : \mathtt{p} <= j < \mathtt{k} : v[j].first == \mathtt{'v'})$
     $\wedge(\forall j : \mathtt{q} < j <= v.size() : v[j].first == \mathtt{'r'}) \wedge 0 \leq \mathtt{p} \leq \mathtt{k} \leq \mathtt{q}+1 < v.size() \wedge \mathtt{k} <= \mathtt{q} \wedge$
     $v[k].first \neq \mathtt{'v'} \wedge v[k].first == \mathtt{'a'}\}$

```
++k;
```

$Inv : \{(\forall j : 0 <= j < \texttt{p} : v[j].first == \texttt{'a'}) \wedge (\forall j : \texttt{p} <= j < \texttt{k} : v[j].first == \texttt{'v'})$
$\wedge(\forall j : \texttt{q} < j <= v.size() : v[j].first == \texttt{'r'}) \wedge 0 \leq \texttt{p} \leq \texttt{k} \leq \texttt{q} + 1 < v.size()\}$

La prueba es semejante a la anterior

3. Tercera condición del bucle cierta.
$Inv \wedge B_{BUCLE} \wedge \neg B_{IF1} \wedge \neg B_{IF2} : \{(\forall j : 0 <= j < \texttt{p} : v[j].first == \texttt{'a'}) \wedge (\forall j : \texttt{p} <= j < \texttt{k} : v[j].first == \texttt{'v'})$
$\wedge(\forall j : \texttt{q} < j <= v.size() : v[j].first == \texttt{'r'}) \wedge 0 \leq \texttt{p} \leq \texttt{k} \leq \texttt{q} + 1 < v.size() \wedge \texttt{k} <= \texttt{q} \wedge$
$v[k].first \neq \texttt{'v'} \wedge v[k].first \neq \texttt{'a'}\}$

```
++k;
```

$Inv : \{(\forall j : 0 <= j < \texttt{p} : v[j].first == \texttt{'a'}) \wedge (\forall j : \texttt{p} <= j < \texttt{k} : v[j].first == \texttt{'v'})$
$\wedge(\forall j : \texttt{q} < j <= v.size() : v[j].first == \texttt{'r'}) \wedge 0 \leq \texttt{p} \leq \texttt{k} \leq \texttt{q} + 1 < v.size()\}$

La prueba es semejante a la anterior

- $\{Inv \wedge \neg B_{BUCLE}\} \Rightarrow Q_{BUCLE}$. Cuando el bucle termina se cumple la postcondición.
$\{Inv \wedge \neg B_{BUCLE}\} : \{(\forall j : 0 <= j < \texttt{p} : v[j].first == \texttt{'a'}) \wedge (\forall j : \texttt{p} <= j < \texttt{k} : v[j].first == \texttt{'v'})$
$\wedge(\forall j : \texttt{q} < j <= v.size() : v[j].first == \texttt{'r'}) \wedge 0 \leq \texttt{p} \leq \texttt{k} \leq \texttt{q} + 1 < v.size() \wedge \texttt{k} > \texttt{q}\}$
De las dos últimas condiciones del bucle $0 \leq \texttt{p} \leq \texttt{k} \leq \texttt{q} + 1 < v.size() \wedge \texttt{k} > \texttt{q}$ deducimos que $\texttt{k} + 1 == \texttt{q}$ Si sustituimos este valor de k en el segundo predicado del invariante obtenemos $\forall j : \texttt{p} <= j < \texttt{q} + 1 : v[j].first == \texttt{'v'}$ que es el segundo predicado de la postcondición del bucle. Los otros predicados se obtienen directamente del invariante.
- Las dos últimas se prueban de manera similar a la prueba realizada en el primer ejemplo. .