JUSTIFICACION

JUSTIFICACION DE LA CONDICION 2:

```
//privada
void Cjt_servidor::condicio2(Arbre<int> &a,list<int> &l,Pelicula &pel, int &ancho, int &alt, int &prof) {
  //Pre: a = A, l = L, pel = Pelicula, ancho contine la suma de los anchos de banda de los nodos superiores,
alt indica en que altura estamos encada momento,
  // prof se utiliza para indicar al nodo padre la profundidad maxima del servidor que contiene la pelicula
  if (not a.es_buit()){
    alt++;
    int valor = a.arrel();
    int n = pel.getId();
    if(servidors[valor-1].contPelicula(n) and servidors[valor-1].getTiempo() == 0) {//el servido visitado
contiene la pelicula
      ancho += servidors[valor-1].getAncho();
      l.insert(l.begin(),valor);
      prof = alt;
    }
    Arbre<int> a1,a2;
    a.fills(a1,a2);
    list<int> ret1,ret2;
    int ancho1,ancho2;
    ancho1 = ancho2 = ancho;
    int alt1,alt2,prof1,prof2;
    alt1 = alt2 =  alt;
    prof1 = prof2 = prof;
    condicio2(a1,ret1,pel,ancho1,alt1,prof1);//ancho1  = ancho + (ancho de todo lo demas)
    condicio2(a2,ret2,pel,ancho2,alt2,prof2);//ancho2  = ancho + (ancho de todo lo demas)
    //HI: ancho1 indica el ancho de banda total de a1,ancho2 indica el ancho de banda total de a2,
    // prof1 indica la profundidad maxima del servidor que contiene la pelicula de a1,
    // prof2 indica la profundidad maxima del servidor que contiene la pelicula de a2,
    a.plantar(valor,a1,a2);
    if(ancho1 >= ancho2 ) {
      if(ancho1 == ancho2) {
            if(prof1 <= prof2) {
              ancho = ancho1;
              l.splice(l.end(),ret1);
              alt = alt1;
              prof = prof1;
            }
            else {
              ancho = ancho2;
              l.splice(l.end(),ret2);
              alt = alt2;
              prof = prof2;
            }
      }
      else{
            ancho = ancho1;
            l.splice(l.end(),ret1);
            alt = alt1;
            prof = prof1;
      }
    }
    else {
      ancho = ancho2;
      l.splice(l.end(),ret2);
      alt = alt2;
      prof = prof2;
    }
  }
}
```

·***Caso recursivo:*** Si hemos entrado en el caso recursivo significa que no esta vacío entonces tenemos que incrementar la altura ya que contamos con un nodo mas.
El caso recursivo consiste en compara el resultado de las llamadas recursivas de los hijos del árbol, Primero tenemos que hacer las llamadas recursivas diciéndole la altura que estamos y el ancho de banda sumando por el conjunto ya visitado de servidores, para obtener el ancho de banda máximo y la profundidad de cada hijo. Una vez obtenida la información comparamos los ancho de banda, puede pasar que:

- ·ancho1 sea mayor o igual que ancho2 : una vez aquí tenemos que distinguir 2 casos:
  - ·ancho1 sea igual que ancho2: si se da el caso miramos la profundidad de cada árbol para elegir el que tenga menor profundidad pero puede pasar que:
    - ·prof1 se menor o igual que prof2 : en este caso elegimos el árbol de la izquierda.
    - ·prof1 es mayor que prof2: elegimos el árbol de la derecha.
  - ·ancho1 es mayor que ancho2: elegimos el árbol de la izquierda.
- ·Ancho1 sea menor que ancho2: elegimos el árbol de la derecha.

·***Finalización:*** El número de elemento de a cumple los requisitos de una función de cota:

(a) es un numero natural, (b) decrece en cada llamada recursiva

JUSTIFICACION DE LA OPCION 5

```
void Cjt_peticion::pelicula_solicitada(int t1, int t2, int n) {
  //Pre: t1 <= t2, n es el numero de peliculas en el Conjunto de peliculas
  vector<int> pel(n,0);//contiene el numero de peticiones
  pair<int,int> max(n,0);//first=id, second=numero de peticions
  list<Peticion>::iterator it = listpet.begin();

  //Inv: pel= vector con la misma mida que el conjunto de peliculas donde cada posicion del
vector(idPelicula-1) indica el numero de
  //        peticiones a esa pelicula,
  //        max = estructura para tener guardado el id y el numero de peticiones maximo de la lista de
peticiones.
  //        it = iterador que recorre la lista desde inicio a fin.
  while (it != listpet.end()) {
    if((*it).getT_i() >= t1 and (*it).getT_i() <= t2) {
      ++pel[(*it).getIdPelicula()-1];
      if(max.second <= pel[(*it).getIdPelicula()-1]) {
            if (max.second == pel[(*it).getIdPelicula()-1]) {
              if (max.first > (*it).getIdPelicula()) {
                max.first = (*it).getIdPelicula();
              }
            }
            else {
              max.first = (*it).getIdPelicula();
              max.second = pel[(*it).getIdPelicula()-1];
            }
      }
    }
    ++it;
  }
  if (max.second == 0) max.first = 0;
  cout << "Pelicula mas solicitada" << endl;
  cout << max.first << " " << max.second << endl;
  //Post: Mostramos por pantalla la pelicula mas solicitada con t1 <= ti <= t2
}
```

Justificación del bucle:

·***inicializaciones:*** De entrada consideramos que no hemos tratado ningún elemento de la lista, Por lo cual como no hemos visitado ninguno consideramos en el vector 'pel' todas las posiciones a cero ya que no hemos consultado ninguna petición, la estructura 'max' consideramos como la película mas solicitada la que tiene como Id mas grande, para poder cumplir siempre la condición interna del bucle, 'it' apunta al principio de la lista y si la lista esta vacía apunta también al final.

·***Condición de salida:*** cuando la condición del bucle es falsa tenemos, en la estructura 'max' el numero máximo de peticiones.

·***Cuerpo del bucle:*** La condición del bucle y el invariante garantiza que hay una petición en la lista y que la podemos tratar.
Para poder incrementar en 'vec' el numero de peticiones tenemos que comprobar que el tiempo inicial de la petición esta en intervalo [t1 .. t2].
·Puede pasar que no este en intervalo: no haría nada seguiría cumpliendo el invariante para la siguiente iteracion.
·Si esta en el intervalo: tendremos que incrementar (+1) la posición del vector pel[idPelicula – 1] i compararíamos con el máximo de peticiones guardado de la estructura max con el numero de peticiones de la película, puede pasar:

- ·Que el numero de peticiones de la película se inferior al máximo guardado, entonces en la estructura max no tenemos que modificarla.
- ·Que el numero de peticiones de la película se superior o igual, si tenemos esta condición tenemos que tratarlas por separado:
  - ·Si el numero de peticiones de la película es igual al máximo guardado en max, tenemos que escoger el identificador mas pequeño entre el idPelicula de la petición y el guardado en la estructura max una vez elegido guardarlo en la estructura max.
  - ·Si el numero de peticiones de la pelicula es superior que el guardado en max, tenemos que guardar en la estructura max el identificador y el numero de peticiones.

Luego de mirar las condiciones incrementamos el iterador para apuntar al siguiente elemento de la lista, una vez llegado a este punto cumple el invariante.

·***Finalización:*** los valores de max cumple las condiciones, el iterador lo incrementamos.

JUSTIFICACION DEL MARCAJE:

·***Inicialización:*** Si ponemos a it apuntando al inicio de la lista no se a marcado ningún servidor como ocupado.

·***condición de salida:*** De la negación de la condición del bucle y de Inv tenemos que it=l.end() y, por tanto, podemos garantizar que hemos recorrido toda la lista.

·**Cuerpo del bucle:** Si entramos en el bucle eso quiere decir que it contiene un identificador de servidor a modificar

·***Finalización:*** it incrementa una posición en cada iteración.