INFO0947: Construction de Programme

Groupe 34: Timothy Smeers, Soline Lèbre

29 mars 2021

Table des matières

1 Sous-problèmes

SP_1 : Balayer le tableau
- Input : Un tableau d'entier de taille N
- Output : \
- Objets utilisé(s) :
  - ⋆ unsigned int i ;
  - ⋆ unsigned int j ;
  - ⋆ int $T[N]$;

SP_2 : Trouver le plus grand nombre présent dans le tableau et incrémenté une variable de 1 pour chaque occurence trouvée
- Input : Un tableau d'entier de taille N
- Output : Le maximum et le nombre d'occurence(s)
- Objets utilisé(s) :
  - ⋆ int max ;
  - ⋆ unsigned int occurence ;
  - ⋆ unsigned int $T[N]$;

Lien entre les SP_s : $SP_1 \subset SP_2$

2 Notations

$\#i \bullet (i \in [0, N-1]) \mid (max_i\, T[N])$

3 Précondition & postcondition

Préconditon : $T[N] \neq NULL \ \&\& \ N > 0 \rightarrow assert(T\,!= NULL \ \&\& \ N > 0)$;

Postcondition :
- $T = T_0 \ \&\& \ N = N_0 \ \&\& \ max\,! = 0 \ \&\& \ occurence > occurence_0$
- max contient la valeur du plus grand nombre présent dans $T[N]$
- $occurence$ contient le nombre d'occurence de max présent dans $T[N]$

4 Invariant graphique

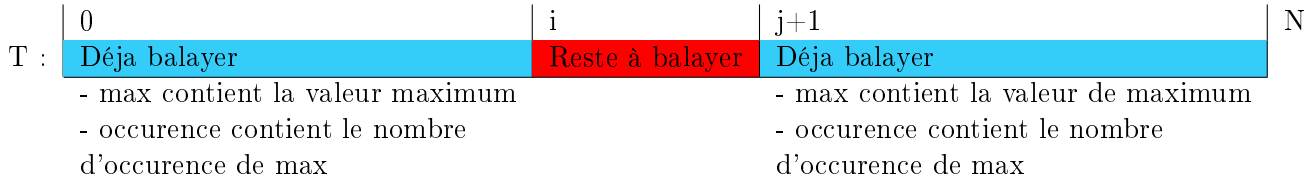

FIGURE 1 – Invariant graphique

1. $G.B = i <= j$
2. $C.A = i > j$
3. $F_t = j - i$

4.1 Règle de complexité

R_1 : Instruction de base
$T(.) = 1$

R_2 : Séquence d'instruction
$T(.) = T_1(.) + T_2(.) + \ldots + T_k(.)$

R_3 : if
$T(.) = max(T_1(.), T_2(.))$

R_4 : Switch
$T(.) = max(T1(.), T2(.), \ldots, T_k(.), T_{k+1}(.))$

R_5 : Boucle
$T(.) = T_1(.) + T_2(.) + \ldots + T_k(.)$
$= \sum_{i=1}^{k} \underbrace{T_i(.)}_{\text{gardien de boucle}}$

R_6 : Programme

4.2 Calcul de la complexité du code

```
int multiplicite ( int *T , const int N , int *max ) {
assert(T != NULL && N > 0) ;

unsigned int i = 0 ;
unsigned int j = N-1 ;
unsigned int occurence = 0 ;                                   } T(A)
*max = T[0] ;
  while (i <= j) {                                                                                    }
                                                                                                      }
      if (T[i] > *max || T[j] > *max){                         }                                      }
         if( T[i] == T[j]){                                    }
             *max = T[i] ;                                     } T(B''_a1)                            }
           occurence = 2 ;                                     }                                      }
                                                               }                                      }
         }else if(T[i] > T[j]){                                }
             *max = T[i] ;                                     } T(B''_a2)                            }
           occurence = 1 ;                                     }                                      }
                                                               }                                      }
             }else{                                            }                                      }
           *max = T[j] ;                                       } T(B''_a3)                            }
         occurence = 1 ;                                       }                                      }
       }                                                       }                                      }
                                                                                                      }
      }else if(*max == T[i] || *max == T[j]){                  }                                      }
            if(i == j)                                         }                                      }
          occurence++ ;                                        } T(B''_b1)                            }
        else if(T[i] == T[j])                                  }                                      }
          occurence +=2 ;                                      } T(B''_b2)                            }
              else                                             }                                      }
          occurence++ ;                                        } T(B''_b3)                            }
                                                               }                                      }
                                                                                                      }
                                  }                                                                   }
                                                                                                      }
     i++ ;                                                     }                                      }
      j- ;                                                     } T(B'_c)                              }
  }                                                                                                   }
return(occurence) ;
}
```

- $T(A) = 1$ Par application de la règle 1
- $T(B''_{a1}) = 2$ Par application de la règle 1, 2, 3
- $T(B''_{a2}) = 2$ Par application de la règle 1, 2, 3
- $T(B''_{a3}) = 2$ Par application de la règle 1, 2, 3
- $T(B'_a) = max(T(B''_{a1}), T(B''_{a2}), T(B''_{a3}))$ Par application de la règle 2 et 3
  $T(B'_a) = 2$
- $T(B''_{b1}) = 1$ Par application de la règle 1 et 2 et 3
- $T(B''_{b2}) = 1$ Par application de la règle 1 et 2 et 3
- $T(B''_{b3}) = 1$ Par application de la règle 1 et 2 et 3

- $T(B'_b) = max(T(B''_{b1}), T(B''_{b2}), T(B''_{b3}))$ Par application de la règle 2 et 3
  $T(B'_b) = 1$
- $T(B'_c) = 2$ Par application de la règle 1
- $T(B') = max(T(B'_a), T(B'_b))$ Par application de la règle 2 et 3
  $T(B') = 2$
- $T(B) = \sum_{i=1}^{k} T_i(.)$ Par application de la règle 5
  $T(B) = n/2$
- $T(n) = T(A) + T(B)$ Par application de la règle 2 et 1 $T(n) = (n/2) + 2$

5 Invariant formel

$$0 \leq i \leq N-1 \wedge max = max_i\, T[N] \wedge occurence = \#i \bullet (i \mid max)$$

6 Code

6.1 main.c

```c
#include <stdio.h>
#include "multiplicite.h"

int main(){

   int T [9] = {13 , 16 , 19 , 17 ,19 , 12 , 2, 4, 10};
   int max ;

   int occurrences = multiplicite(T, 9, &max);
   printf("%d - %d\n", occurrences, max);
}
```

Codingstyle 1 – main.c

6.2 multiplicite.h

```
#ifndef MULTIPLICITE_H_
#define MULTIPLICITE_H_

/**
 * @fn int multiplicite(int*, const int, int*)
 * @brief Obtain, for T, the greatest value of T as well as the number
 * of occurrences of this value in T.
 *
 * @pre T != NULL && N > 0
 * @post T = T0 && N = N0 && max != 0 && occurence > occurence0
 *         max contains the value of the largest number present in T [N]
 *         occurrence contains the number of occurrences of max present in T [N]
 * @param T an array with N integer values
 * @param N table size T
 * @param max contains the value of the largest number present in T [N]
 * @return The number of occurrences of the maximum in T.
 */
int multiplicite ( int *T , const int N , int *max );

#endif MULTIPLICITE_H_
```

Codingstyle 2 – multiplicite.h

6.3 multiplicite.c

```c
#include <stdio.h>
#include <assert.h>
#include "multiplicite.h"

int multiplicite ( int *T , const int N , int *max ) {
   assert(T != NULL && N > 0);

   unsigned int i = 0;
   unsigned int j = N-1;
   unsigned int occurence = 0;
   *max = T[0];

   while (i <= j) {

      if (T[i] > *max || T[j] > *max){
         if( T[i] == T[j]){
            *max = T[i];
            occurence = 2;

         }else if(T[i] > T[j]){
            *max = T[i];
            occurence = 1;
         }else{
            *max = T[j];
            occurence = 1;
         }

      }else if(*max == T[i] || *max == T[j]){
         if(i == j)
            occurence++;
         else if(T[i] == T[j])
            occurence +=2;
         else
            occurence++;
      }
      i++;
      j--;
   }
   return(occurence);
}
```

Codingstyle 3 – multiplicite.c

7 Documentation

Pour plus d'information sur le code vous pouvez consulter le site internet contenant la documentation doxygen.