DM Enveloppe convexe

Bartolomeo Ryan

Janvier 2023

1 Fonction plus bas

```
int plus_bas(int tab[][2],int n){
    int j = 0;
    for(int i = 1; i < n; i++){
        if(tab[i][1]<tab[j][1] || (tab[i][1]==tab[j][1] && tab[i][0]<tab[j][0] ))
            j = i;
    }
    return j;
}
```

2 Correction

Cette fonction termine car la seule boucle est une boucle `for`. On veut montrer la spécification :
$E : T \subset \mathbb{Z}^2$,
$S : j \in \mathbb{N}$, $T[j] = min_{\lessdot} T$ avec $a \lessdot b \Leftrightarrow e_y < f_y \vee (e_y = f_y \wedge e_x < f_x)$

Pour cela, on peut prouver l'invariant :
$I(j, i) : T[j] = min_{\lessdot} T_{[\![0,i]\!]}$

2.1 Initialisation

Avant la boucle, $j = 0$ et T est un single-ton donc :
$T[0] = min_{\lessdot} T_{[\![0,0]\!]}$
donc $I(j, 0)$ est vérifié.

2.2 Itération

Montrons que pour tout rang i, si $I(j, i-1)$ est vérifié, alors $I(j, i)$ l'est aussi.

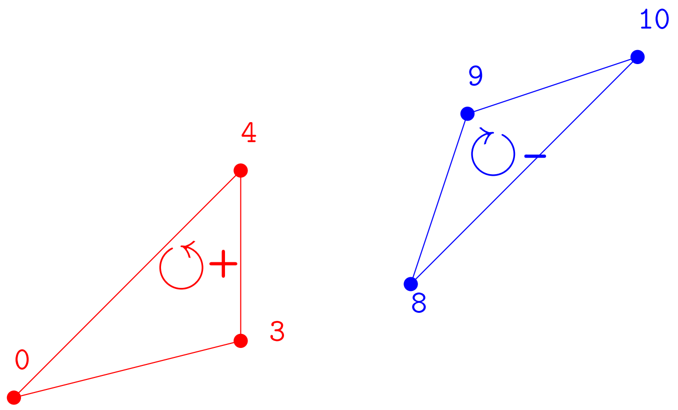
- Si $T[i]$ n'est pas plus petit par la relation d'ordre que $T[j]$, donc j reste l'indice du minimum, il n'est donc pas modifié.
- Si au contraire, $T[i]$ est plus petit, alors on entre dans la condition donc $j = i$

Ainsi, par hypothèse d'itération, à la fin de chaque itération, $I(j, i-1) \equiv I(j, i)$ donc en sortie de boucle, $I(j, |T|)$ sera vérifié.

Ainsi, puisque la boucle termine et que j contient la valeur du point le plus bas à la fin de la boucle, **la fonction est correcte**

3 Exemples d'orientation

i	j	k	Orientation
0	3	4	1
8	9	10	-1



4 Fonction orientation

```
int orient(int tab[][2],int i, int j, int k){
  int cross=(tab[k][1]-tab[i][1])*(tab[j][0]-tab[i][0])
           -(tab[k][0]-tab[i][0])*(tab[j][1]-tab[i][1]);
  return cross ? cross/abs(cross): 0;
}
```

5 Relation d'ordre de $\preceq$

$\preceq$ est une relation d'ordre car c'est une relation est réflexive, antisymétrique et transitive sur l'ensemble P.

6 Fonction prochain point

```
int prochain_point(int tab[][2],int n,int i){
    int j = 0;
    for(int k = 0; k < n; k++){
        if(orient(tab,i,j,k)<=0 && k!=i)
            j = k;
    }
    return j;
}
```

7 Prochain point pour $i = 10$

itération n°	0	1	2	3	4	5	6	7	8	9	10	11
valeur du minimum	0	1	2	2	2	5	5	5	5	5	5	5

8 Fonction Enveloppe Convexe de Jarvis

```
int conv_jarvis(int tab[][2], int n, int *env){
    int i = plus_bas(tab,n);
    int ind = 0;
    int p = i;
    do{
        env[ind] = p;
        p = prochain_point(tab,n,p);
        ind++;
    }while(i!=p);
    return ind;
}
```

9 Temps d'exécution

Cette fonction prendrai dans l'ordre de $n + n * n$ opérations pour récupérer l'enveloppe dans le pire des cas (fonction plus bas + prochain point * nombre de points), on estime donc sa complexité temporelle à $O(n^2)$

10 Tri des éléments

Parmi les algorithmes de tri répandus, on peut penser au tri fusion (ou *merge sort*) qui a une complexité de $O(n \log(n))$

11 Fonction Mise à jour de l'enveloppe supérieure

```
void maj_es(int tab[][2],stack es,int i){
    int t; // top
    int n; // next (juste en dessous de top)
    bool ordered = true;
    while(es->size >= 2 && ordered){
        // mise à jour des variables
        t = pop(es);
        n = top(es);
        push(t,es);

        if(orient(tab,i,t,n)<0)
            pop(es);
        else
            ordered = false;
    }
    push(i,es);
}
```

12 Fonction Mise à jour de l'enveloppe inférieure

On opère au même raisonnement mais en prenant une orientation négative

```
void maj_ei(int tab[][2],stack ei,int i){
    int t;
    int n;
    bool ordered = true;
    while(ei->size >= 2 && ordered){
        t = pop(ei);
        n = top(ei);
        push(t,ei);

        if(orient(tab,i,t,n)>0)
            pop(ei);
        else
            ordered = false;
    }
    push(i,ei);
}
```

13 Fonction Enveloppe convexe de Graham (variante A.Andrew)

```
stack conv_graham(int tab[][2],int n){
    stack es = new_stack();
    stack ei = new_stack();
    for(int i = 0; i < n; i++){
        maj_es(tab,es,i);
```

```
        maj_ei(tab,ei,i);
    }
    stack s = new_stack();
    pop(es); // doublon
    stack reversed_es = new_stack();
    // on vide es dans un autre tas pour le renverser
    for(int i=es->size;i--;)push(pop(es),reversed_es);

    for(int i=reversed_es->size;i--;)push(pop(reversed_es),s);
    for(int i=ei->size;i>1;i--)push(pop(ei),s);

    return s;
}
```

14 Analyse du temps d'exécution

Les fonctions de mises à jour prennent au pire des cas $\log n$ opérations pour être exécutées. Puisqu'on exécute ces procédures n fois, on peut estimer qu'on devra attendre l'exécution de $2n \log n$ opérations, d'où la complexité de l'algorithme est de l'ordre de $O(n \log n)$