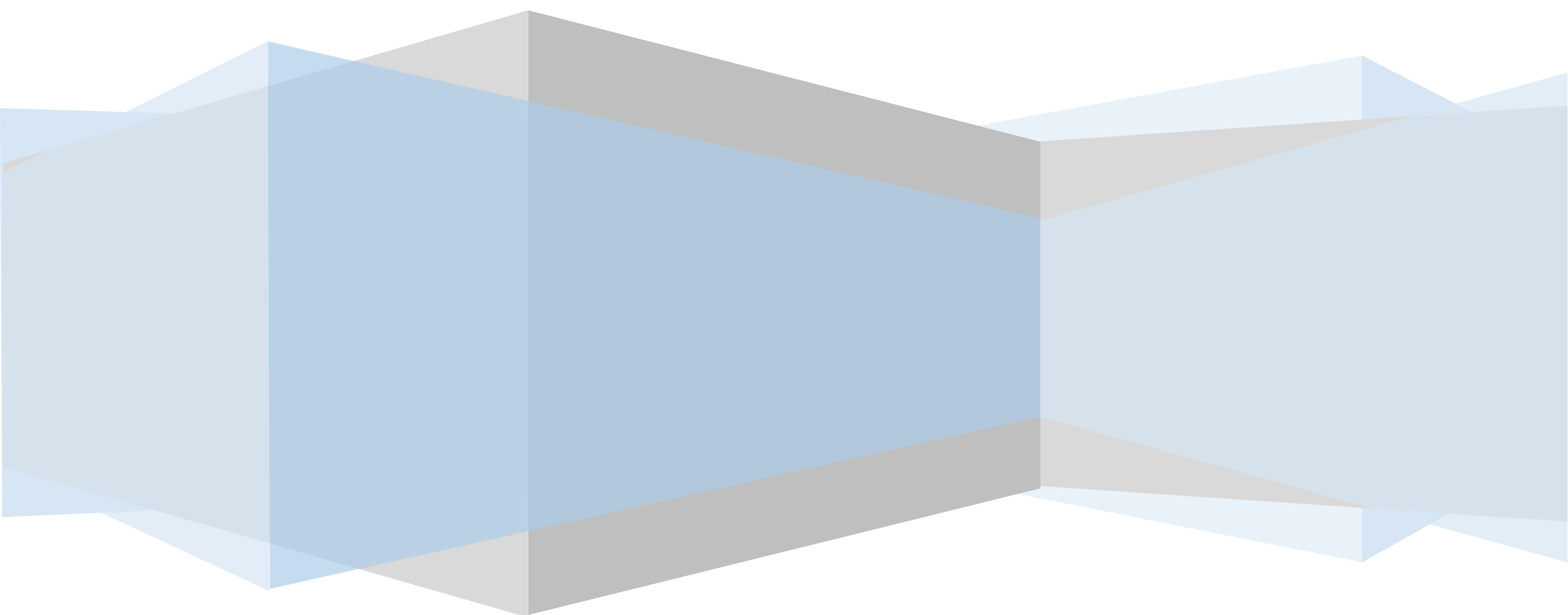
Algorithmique moderne : Analyse et complexité

Professeur Habiba Drias



Plusieurs algorithmes pour un même énoncé de problème

Considérons l'exemple de calcul des fréquences de nombres entiers positifs fournis dans un tableau. Les fréquences calculées seront affichées pour chacun de ces nombres. Les trois algorithmes suivants sont développés pour solutionner le problème.

Algorithme Calcul-Fréquences1

```
const   n=500 ;
var     T : tableau [1..n] d'entiers ;
        i, j, fréquences : entiers ;
        trouve : booléen ;
début
        pour    i := 1 à n faire
                j :=1 ;
                trouve := faux ;
                tantque (j<i) et (non trouve) faire
                        si T[i] = T[j] alors trouve := vrai
                                                Sinon j := j+1 ;
                        fsi
                fintantque
                si non trouve alors
                        fréquences := 1 ;
                        pour j := i+1 à n faire
                                si T[i] = T[j] alors fréquences := fréquences + 1
                                fsi
                        finpour
                        écrire ('fréquence de' T[i] 'est égale à' fréquences) ;
                fsi
        finpour
fin
```

Algorithme Calcul-Fréquences2

```
const   n=500 ;
var     T : tableau [1..n] d'entiers ;
        i, j, fréquences : entiers ;
début   pour    i := 1 à n faire
                si T[i] <> -1 alors
                        fréquences := 1 ;
                        pour j := i+1 à n faire
                                si T[i] = T[j] alors fréquences := fréquences + 1 ;
                                                                T[j] := -1 ;
                                fsi ;
                        finpour ;
                        écrire ('fréquence de' T[i] 'est égale à' fréquences) ;
                fsi ;
        finpour ;
fin
```

Algorithme Calcul-Fréquences3

```
const   n=500 ;
var     T : tableau [1..n] d'entiers ;
        fréquences : tableau[1..n] d'entiers ;
        i : entiers ;
début
        pour    i := 1 à n faire
                fréquences[i] := 0 ;
        finpour ;
        pour i := 1 à n faire
                fréquences[T[i]] := fréquences[T[i]] + 1 ;
        finpour ;
        pour    i := 1 à n faire
                si fréquences[i] <> 0 alors
                écrire ('\nfréquence de' i 'est égale à' fréquences[i]) ;
        finpour ;
fin
```

Analyse et calcul de la complexité par l'exemple

Nous calculons dans ce qui suit la complexité de chacun des trois algorithmes en considérant la donnée extrême qui nécessite le plus de temps d'exécution. Ce cas est appelé 'cas le plus défavorable' ou 'pire cas'. Pour le premier algorithme par exemple, le pire cas correspond à la donnée où tous les nombres fournis en entrée sont distincts les uns des autres.

Algorithme Calcul-Fréquences1

Le principe de l'algorithme est de considérer chaque élément du tableau un par un et de :

- vérifier d'abord en consultant les éléments qui se trouvent à gauche de l'élément courant si ce dernier existe, auquel cas il a déjà été traité et donc sa fréquence a déjà été calculée,
- ensuite dans le cas où l'élément courant n'existe pas à gauche, parcourir tous les éléments qui se trouvent à droite pour calculer sa fréquence.

L'algorithme possède trois boucles : deux boucles internes séquentielles imbriquées dans une boucle externe. Dans le pire cas, l'algorithme procède à un balayage du tableau T n fois et pour chaque élément du tableau, il parcourt tout le tableau une deuxième fois. Le nombre total d'exécutions des instructions des boucles internes est égal à n*n. La complexité est de l'ordre de n^2 et est notée $O(n^2)$.

Algorithme Calcul-Fréquences2

Cet algorithme est une version améliorée du premier algorithme. A chaque fois qu'un élément est traité, sa valeur est mise à -1. De cette manière, la recherche de l'élément courant à gauche ne sera plus nécessaire et par conséquent la première boucle interne est supprimée. Seule la deuxième boucle interne reste et est exécutée (n-i) fois à chaque itération de la boucle externe. La complexité est donc égale à $\sum_{i=1}^{i=n}(n-i) = \frac{n(n-1)}{2} = \frac{n^2-n}{2}$ et est donc en $O(n^2)$.

Algorithme Calcul-Fréquences3

Dans cet algorithme, nous utilisons un deuxième tableau appelé *fréquences* dans lequel nous calculons les fréquences des nombres. Nous supposons que les nombres sont compris entre 1 et n. L'algorithme commence par mettre à 0 tous les éléments du tableau *fréquences* puis il parcourt le tableau T en incrémentant la fréquence de l'élément courant *i* qui se trouve dans le tableau fréquences à la position T[i].

C'est clair que la complexité dans ce cas est en O(n) du moment que l'algorithme contient trois boucles séquentielles (une pour l'initialisation du tableau, une deuxième pour le calcul des fréquences et la dernière pour l'impression des résultats) avec un nombre d'itérations égal à n.

En conclusion, nous avons développé trois algorithmes différents pour le même problème avec des complexités différentes, les deux premières solutions ont une complexité du même ordre à savoir $O(n^2)$ et la troisième une complexité plus réduite qui est O(n) mais gagnée au détriment de l'espace mémoire.

Machine de Turing

Une machine de Turing est un outil théorique élémentaire utilisé dans la théorie de la calculabilité et également dans la théorie de la complexité. Elle est définie formellement comme un septuplé :

(Q,T,I,D,b,q_0,q_f) où

Q est l'ensemble des états de la machine

T est l'alphabet du langage reconnu par la machine

I est l'alphabet des données

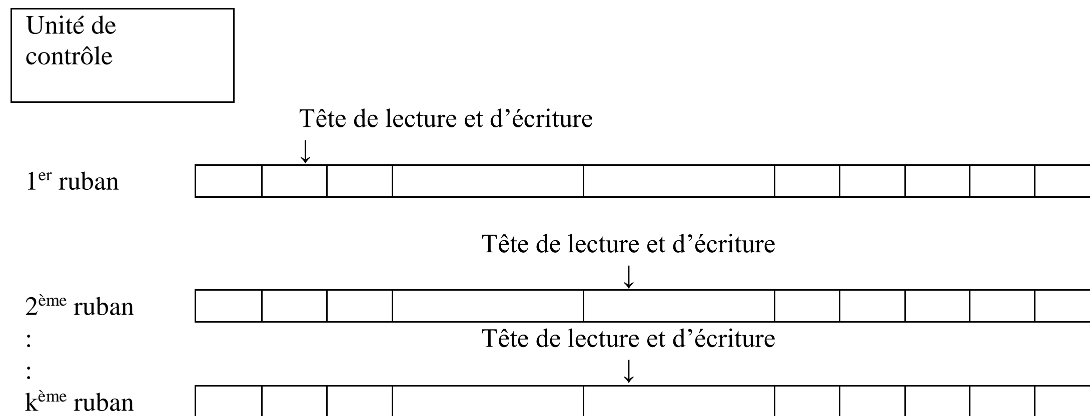
D est la fonction de transition

b est le caractère blanc

q_0 est l'état initial et

qf est un état final. On peut avoir plusieurs états finaux.

La machine de Turing est schématisée comme suit :



Elle est constituée de k rubans sur lesquels on peut lire et écrire des symboles qui appartiennent à T. Les k rubans sont structurés en plusieurs cases et peuvent servir de mémoire pour la machine ainsi que d'unités d'entrée/sortie. En effet, les entrées sont stockées initialement sur un ou plusieurs rubans et les résultats peuvent être récupérés sur une ou plusieurs cases des rubans. Les autres rubans peuvent servir pour le traitement effectué par la machine. A chacun des rubans est associée une tête de lecture et/ou d'écriture permettant à la fonction de transition de modifier le contenu des cases.

Pour illustrer le fonctionnement de la machine de Turing, considérons l'exemple 4 qui construit une machine de Turing pour le langage 0^k1^k de l'exemple 1.

Exemple 4 :

Pour le développement de la machine de Turing, il faut définir l'ensemble des états Q, l'alphabet T, l'alphabet I, la fonction de transition et les états initial et finaux.

1ère solution : utilisation d'un seul ruban

Initialement, la donnée se présente sur le ruban suivie de blancs. La machine proposée effectue des va et vient tout le long de la chaine en remplaçant les extrémités par des blancs. L'extrémité gauche doit faire apparaître un 0 tandis que l'extrémité droite doit faire figurer un 1. Dans le cas où cette règle n'est pas respectée, la machine transite vers un état indiquant que le mot présenté en entrée n'est pas valide. Le résultat est présenté sur le ruban à la fin du traitement, la tête de lecture pointe vers une case contenant 0, toutes les autres cases sont à blanc. Et si tout se passe bien, la machine s'arrête au milieu du mot en transitant vers un autre état final signalant l'appartenance du mot au langage. Dans ce cas, la case vers laquelle est positionnée la tête de lecture contient 1, toutes les autres cases sont à blanc. La machine est décrite comme suit :

Rôles des états de la machine

q0 : état initial, remplacer 0 par b
q1 : se déplacer vers l'extrémité droite
q2 : se déplacer d'un cran vers la gauche et mettre le dernier 1 à blanc
q3 : se déplacer vers l'extrémité gauche
q4 : état final signalant un échec : un 1 est rencontré à l'extrémité gauche
q5 : état final signalant un échec : un 0 est rencontré à l'extrémité droite
q6 : état final signalant l'appartenance du mot au langage

(q0,0) → (q1, b, R)
(q0,1) → (q4, b, R)
(q1,0) → (q1, 0, R)
(q1,1) → (q1, 1, R)
(q1,b) → (q2, b, L)
(q2,0) → (q5, b, L)
(q2,1) → (q3, b, L)
(q2,b) → (q6, 1, S)
(q3,1) → (q3, 1, L)
(q3,0) → (q3, 0, L)
(q3,b) → (q0, b, R)
(q4,1) → (q4, b, R)
(q4,0) → (q4, b, R)
(q4,b) → (q4, 0, S)
(q5,1) → (q5, b, L)
(q5,0) → (q5, b, L)
(q5,b) → (q5, 0, S)

Dans la partie gauche d'une transition, on note l'état dans lequel se trouve la machine suivi du symbole se trouvant sous la tête de lecture. Dans le triplet de droite, on trouve l'état vers lequel transitera la machine suivi du symbole à écrire et de la direction du mouvement de la tête de lecture et/ou d'écriture. Trois directions sont possibles pour le mouvement de la tête : S pour stationnaire, L pour gauche et R pour droite.
La complexité temporelle se déduit par le nombre de transitions effectuées pendant le traitement d'une donnée. Le pire cas se produit lorsque la chaine présentée en entrée est traitée complètement. Dans cette situation, les transitions se décomposent comme suit :

- n+1 déplacements vers la droite
- n déplacements vers la gauche
- n-1 déplacements vers la droite
- …
- 1 déplacement

Le nombre total de transitions est donc évalué à $\sum_1^{n+1} i$ = (n+1)(n+2)/2= (n^2+3n+2)/2. La complexité est donc en O(n^2).

La complexité spatiale s'exprime par le nombre de cases utilisées au total. Ce nombre est égal à n et sert à stocker le mot du langage à vérifier. Aucune autre case n'est exploitée par la suite. La complexité spatiale est donc O(n).

2ème solution : utilisation de deux rubans

On suppose que le mot se présente initialement sur le premier ruban. Le deuxième ruban sert à copier la séquence des 0, tout en remplaçant ceux du premier ruban par des blancs. La machine passe ensuite dans une seconde étape qui consiste à comparer les longueurs respectives des sous chaines des 0 et celles des 1. Nous aurons besoin de définir 5 états avec les rôles suivants :

- q0 : état initial
- q1 : sert à reporter la séquence des 0 sur le 2ème ruban et de préparer la comparaison avec la séquence des 1
- q2 : sert à comparer la longueur des séquences respectives des 0 et des 1
- q3 : état final servant à déclarer un résultat négatif

- q4 : état final servant à annoncer le succès

la fonction de transition est alors la suivante :
(q0,0,b) → (q1, (0,S), (b,R))
(q0,1,b) → (q3, (1,S), (b,S))
(q1,0,b) → (q1, (b,R), (0,R))
(q1,b,b) → (q3, (0,S), (b,S))
(q1,1,b) → (q2, (1,S), (b,L))
(q2,1,0) → (q2, (1,R), (0,L))
(q2,1,b) → (q3, (b,R), (b,S))
(q2,b,0) → (q3, (0,S), (b,S))
(q2,0,b) → (q3, (0,S), (b,S))
(q2,b,b) → (q4, (1,S), (b,S))
(q3,1,b) → (q3, (b,R), (b,S))
(q3,0,b) → (q3, (b,R), (b,S))
(q3,b,b) → (q3, (0,S), (b,S))

Dans le premier triplet gauche d'une transition, on note l'état dans lequel se trouve la machine suivi du symbole se trouvant sous la tête de lecture du 1er ruban, ensuite le symbole qui se trouve au niveau du 2ème ruban.
Dans le triplet de droite, on trouve l'état vers lequel transitera la machine suivi d'une paire concernant le 1er ruban et constituée du symbole à écrire sur le 1er ruban et de la direction du mouvement de la tête de lecture et/ou d'écriture du 1er ruban. La dernière paire porte le symbole à écrire sur le 2ème ruban suivi de la direction du mouvement de la tête de lecture et/ou d'écriture du 2ème ruban. Trois directions sont possibles pour le mouvement de la tête : S pour stationnaire, L pour gauche et R pour droite.
La complexité temporelle de la machine se mesure par le nombre de transitions effectuées pendant le traitement de la donnée. Dans le cas de notre exemple, le pire cas se produit lorsque le mot est traité en entier, deux cas se présentent : le mot est correct ou bien le mot contient plus de 1 que de 0. Dans ces deux situations, les transitions effectuées sont :

- 1 transition pour marquer l'extrémité droite du 2ème ruban, nécessaire pour la suite du traitement
- n0 transitions pour copier la séquence des 0, n0 étant la longueur de la séquence des 0
- n1 transitions pour comparer la séquence des 1 avec la séquence des 0
- 1 transition pour mettre le résultat sur le 1er ruban

Au total, le nombre de transitions est égal à n0 + n1 + 2 = n +2, où n est la longueur du mot présenté en entrée. La complexité est donc O(n).

Le nombre de cases utilisées au total étant de : n+k0 où k0 est le nombre de cases du 2ème ruban occupées par les 0 copiés à partir du 1er ruban. La complexité spatiale est O(n) puisque k0<=n et donc n+k0 <= 2n.

Listes chainées

Représentation à l'aide de tableau et opérations élémentaires

La liste L = {a1, a2, …, an} est représentée à l'aide des tableaux *élément* et *suivant* comme suit :

	élément	**suivant**
Tête=1	a1	2
2		3
	…..	…..
n	an	-1
libre		

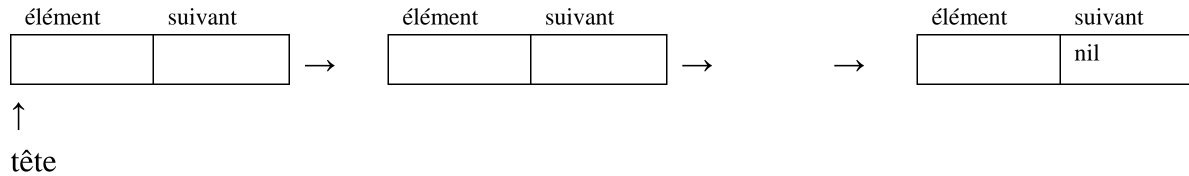
Pour rechercher un élément dans la liste, la manière la plus simple est de parcourir le tableau *élément* selon l'ordre indiqué par le vecteur *suivant* et de comparer à chaque fois l'élément recherché avec l'élément du tableau. Si les éléments sont triés, l'insertion d'un élément consiste à placer l'élément dans la liste à la position qui convient par respect à l'ordre préétabli. La suppression d'un élément consiste à rechercher d'abord l'élément qu'on veut supprimer s'il existe dans la liste, ensuite à procéder à la suppression proprement dite. Un algorithme d'insertion peut s'écrire comme suit :

```
procédure insertion ;
(* insertion d'un entier x dans une liste triée *)
entrée : tête, libre : -1..max ; x : entier ;
sortie : tête, libre : -1..max  ;

var p, q, temp : -1..max   ;
début
        p := tête ;
        élément [libre] := x ;
        temp := suivant[libre] ;
        si (non liste-vide) et (non liste-pleine) alors
          début tant que (élément[p] < x) et (suivant[p] ≠ -1) faire
                  début    q := p ;
                           p := suivant[p] ;
                  fin ;
                  si (élément[p] > x) alors
                     si (p = tête) alors début suivant[libre] := tête ;
                                               tête := libre ;
                                  fin
                            sinon début       suivant[libre] := p ;
                                              suivant[q] := libre ;
                                          fin
                  sinon début suivant[libre]:=  suivant[p];
                                 suivant[p] := libre ;
                         fin ;
                  libre := temp ;
          fin
          sinon si (liste-vide) alors début tête := libre ;
                                             suivant[tête] := -1 ;
                                             libre := temp ;
                                       fin
                                  sinon signaler ('liste pleine')
fin
```

Représentation à l'aide de structure dynamique et opérations élémentaires

Tous les traitements vus dans la section précédente peuvent être conduits sur des structures de données dynamiques, en l'occurrence sur des listes dynamiques. La représentation graphique d'une liste dynamique est la suivante :



↑

tête

La liste dynamique utilise la structure d'enregistrement et la notion de pointeurs. Un enregistrement est constitué de deux ou plusieurs champs et contenant l'ensemble de l'information de la liste. Dans notre cas, deux champs sont spécifiés : *élément* et *suivant*. Le champ *élément* contient un élément de la liste, en l'occurrence un entier et le champ *suivant* indique le chainage établi entre les éléments de la liste. La déclaration de la liste en algorithmique se fait de la manière suivante :

```
type
liste:    enregistrement
                élément : entier ;
                suivant : ↑liste ;
          fin ;
var
          tête : ↑liste ;
```

Un algorithme de suppression peut s'écrire comme suit :

```
procédure suppression ;
(* suppression de x d'une liste non triée *)
entrée : tête : ↑liste ;  x : entier ;
sortie : tête : ↑liste ;

var p, q : ↑liste  ;
début
          si (liste-vide) alors signaler ('liste vide')
          sinon début
                     p := tête ;
                     tant que (p↑.élément ≠ x) et (p↑.suivant ≠ nil) faire
                             début    q := p ;
                                      p := p↑.suivant;
                             fin ;
                    si (p↑.élément = x) alors
                        début   si p = tête alors tête := p↑.suivant
                                          sinon q↑.suivant :=  p↑.suivant ;
                                libérer (p) ;
                        fin ;
                   sinon signaler ('x n'existe pas dans la liste') ;
               fin
fin
```

Queues ou files d'attente

Représentation à l'aide de tableau et opérations élémentaires

L'initialisation de la queue, les algorithmes de recherche, d'insertion, de suppression ainsi que la procédure de test de l'état de la queue sont les suivants :

Initialisation de la queue :
n := 200 ; premier := 0 ; dernier := 0 ; libre := 0 ;

---

procédure *queue-vide* ***;***
entrée *: queue, premier, dernier, libre ;*
sortie *: vrai ou faux ;*

début
    si *(premier = libre) et (dernier = libre)*
      alors retourner *vrai* ***sinon retourner*** *faux ;*
fin

---

procédure *queue-pleine* ***;***
entrée *: queue, premier, dernier, libre ;*
sortie *: vrai ou faux ;*

début ***si*** *(libre = premier) et (dernier = (premier-1) modulo n)* ***alors retourner*** *vrai* ***sinon retourner*** *faux ;*
fin

---

procédure *recherche* ***;***
entrée *: queue, premier, dernier, x ;*
sortie *: position de x dans la queue ;*

var *p :* ***entier*** *;*
    trouve : ***booléen ;***
début *trouve :=* ***faux*** *;*
    p := premier ;
    tant que *(****non*** *trouve)* ***et*** *(p <= dernier)* ***faire***
      si *(queue[p] = x)* ***alors*** *trouve :=* ***vrai sinon*** *p :=(p+1)modulo n ;*
    si *(trouve)* ***alors*** *retourner(p)*
      sinon *retourner('x n'existe pas dans la queue') ;*
fin

```
procédure enfiler ;
entrée : queue, premier, dernier, libre, x ;
sortie : queue, premier, dernier libre;

début
        si queue-pleine alors signaler ('queue pleine')
        sinon
                début
                   si queue-vide alors premier := libre ;
                    queue [libre] := x ;
                   dernier := libre ;
                    libre := (libre +1) modulo n ;
                fin
fin
```

```
procédure défiler;
entrée : queue, premier, dernier, libre ;
sortie : le premier élément de la queue ;

var x ;
début
        si (queue-vide) alors  signaler ('queue vide')
                        sinon début
                                        x := queue[premier] ;
                                        premier := (premier+1) modulo n ;
                                        retourner (x) ;
                                fin
fin
```

Piles

Représentation à l'aide de tableau et opérations élémentaires

La pile contenant les éléments a1, a2, …, an, est représentée à l'aide du tableau suivant :

	pile
1	a1
2	
	…..
sommet→ n	an
max→	

```
Initialisation de la pile
Sommet := 0 ;
```

```
procédure empiler (var sommet : entier ; x : entier) ;
début
        si (sommet ≠ max) alors
        début sommet := sommet + 1 ;
                pile[sommet] := x ;
        fin
fin
```

```
procédure dépiler (var sommet : entier) ;
début
        si (sommet ≠ 0) alors
        début retourner ( pile[sommet]);
                sommet := sommet-1 ;
        fin
fin
```