Hachage Statique

- Introduction
- Principe du hachage
- Terminologie
- Fonctions de Hachage
- Méthodes de résolution de collisions
- Estimation des débordements
- Conclusion

Introduction

- Problématique

  Supposons que nous avons n enregistrements (données) stockés dans un tableau. On désire:

  - rechercher une donnée x
  - insérer une donnée x

Solutions

* Si tableau ordonné Alors

  - Recherche: Recherche Dichotomique → O(Log(n)) → Rapide
  - Insertion: Décalage des éléments du tableau → Lent

* Si tableau non ordonné Alors

  - Recherche: Recherche Séquentielle → O(n) → Lent
  - Insertion: Insertion fin du tableau → Rapide

Introduction

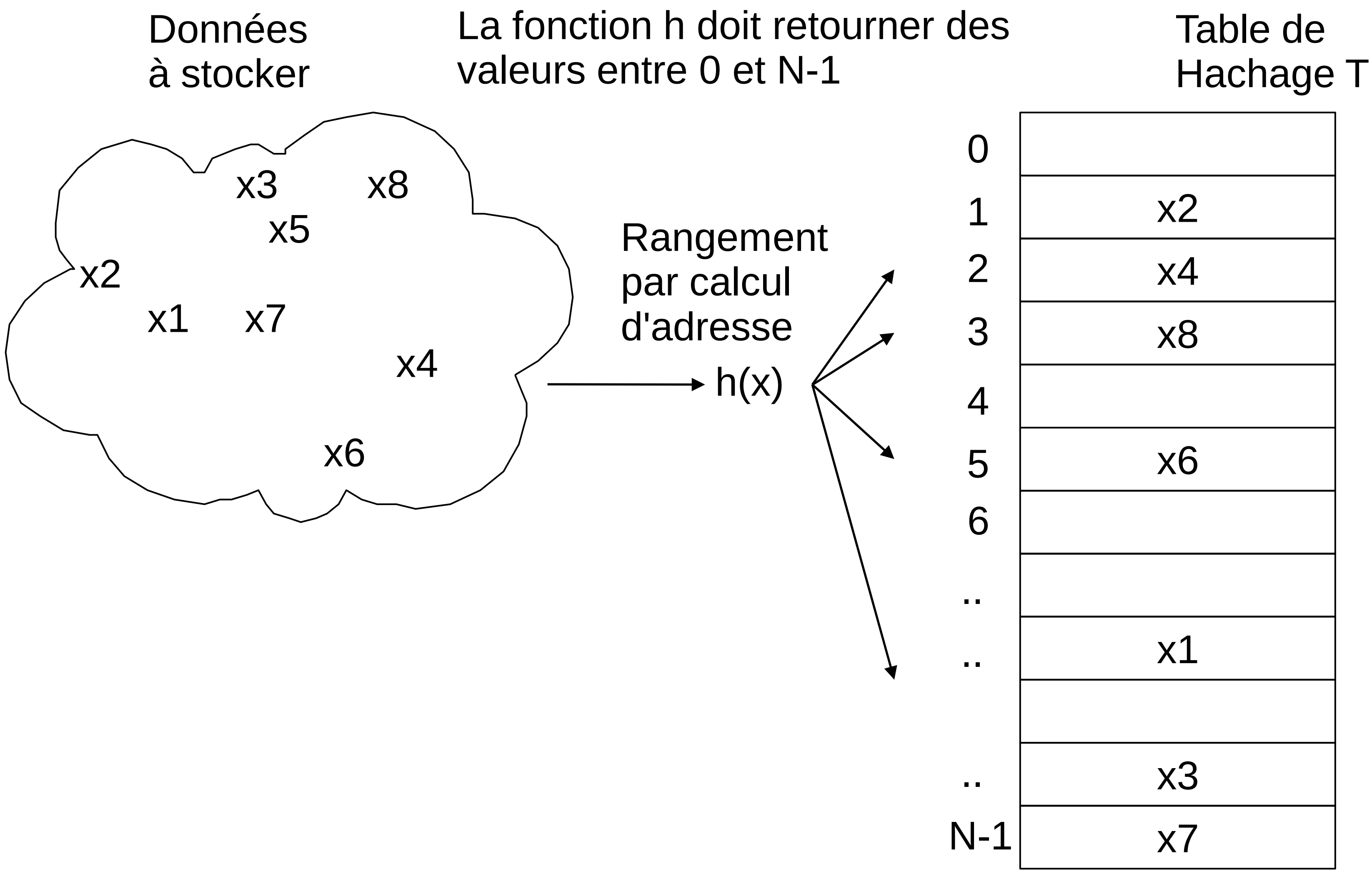
<u>Autre solution</u>

Tableau est une table de hachage dont la complexité de ces opérations (recherche et insertion), en moyenne, peut se faire en un temps constant

Hachage = Hashing (en anglais)

= Technique de rangement dispersé

Principe du hachage



Terminologie

- Stocker des données (x) dans une table (T) de taille N, en utilisant une fonction (h) pour la localisation rapide

  ➔ **calcul d'adresse**

- La fonction (h) calcule l’emplacement de (x) et retourne la case d'indice

  ➔ **adresse primaire**

- Si la case est déjà occupée (**collision**), on insère (x) à un autre emplacement

  ➔ **adresse secondaire ➔ Débordement**

- L’adresse secondaire est déterminé par un algorithme donné

  ➔ **méthode de résolution de collisions**

- Si des données ont la même image par la fonction de hachage

  ➔ **synonyme**

En résumé, pour utiliser une table de rangement dispersé (hachage), on doit donc définir:

- une fonction de hachage
- une méthode de résolution des collisions

Fonctions de hachage

- Il s’agit de trouver une fonction h tels que:

  $0 <= h(x) < N$

  qui réduit au maximum le nombre de collisions

  - L'idéal, c'est d'avoir une fonction de hachage bijective.
  - Le pire des cas, c'est lorsque toute donnée est hachée en une même adresse.
  - Une solution acceptable est une solution où certaines données partagent la même adresse (f est surjective).

- Quelques fonctions de hachage usuelles
  - La fonction de division
  - La fonction du « middle square »
  - La fonction du « transformation radix »

Fonctions de hachage

1) La fonction de division

h(x) = x MOD N

retourne le reste de la division par N (la taille de la table)

- C'est une fonction facile et rapide à calculer mais sa qualité dépend de la valeur de N.

- Il est préférable que N soit **<u>premier</u>** et ne doit pas être une puissance de 2

- <u>Exemple</u>:

Soit N = 10 Alors:

h(5) = 5 MOD 10 = 5

h(55) = 55 MOD 10 = 5 ➔ collision

h(23) = 23 MOD 10 = 3

h(453) = 453 MOD 10 = 3 ➔ collision

Soit N = 11 Alors:

h(5) = 5 MOD 11 = 5

h(55) = 55 MOD 11 = 0

h(23) = 23 MOD 11 = 1

h(453) = 453 MOD 11 = 2

➔ Pas de collisions dans ce cas car N=11 est **<u>premier</u>**

➔ **minimiser les collisions**

Fonctions de hachage

2) La fonction du milieu du carré « middle square »

On élève la donnée x au carré x^2 et on prend les chiffres du milieu

- Cette méthode donne de bons résultats si le nombre au carré n'a pas de zéros.

- Exemple:

Soit N = 100

h(500) = 0 car $(500)^2 = 250000$

h(12) = 14

ou bien h(12) = 44 car $(12)^2 = 144$

Fonctions de hachage

3) La fonction dite « transformation radix »

On convertit la donnée x dans une base de numération et on prend le reste de la division.

- <u>Exemple</u>:

Soit: $x = 453$ ➔ base 10

$x = 382$ ➔ base 11

$h(x) = (x)_{11}$ MOD N

Fonctions de hachage

<u>En conclusion</u>:

Il n’y a pas de fonction de hachage universelle.

Cependant, une bonne fonction:

- est rapide à calculer
- répartit uniformément les éléments

Elle dépend donc:

- de la machine
- des éléments

Mais aucune fonction n’évite les collisions, qu’il va falloir traiter.

Méthodes de résolution de collisions

- Lors de l'insertion de x, si l'adresse primaire h(x) est déjà utilisée par une autre donnée, la méthode de résolution de collision permet de trouver un autre emplacement (libre) pour x
- Pour résoudre les collisions, deux stratégies se présentent:
  - Les méthodes indirectes ou le hachage par chainage (hachage dynamique)
  - Les méthodes directes ou le hachage par calcul de l’emplacement (hachage statique):

    1) Essai linéaire

    2) Double hachage

1) Essai Linéaire

Principe:

- S'il se produit une collision sur la case h(x), on essaie les cases qui la précèdent : h(x)-1, h(x)-2, h(x)-3,..., 0, N-1, N-2,…, jusqu'à trouver une case vide.

- La rencontre d'une case vide indique que la donnée n'existe pas
  ➔ Il faudra sacrifier une case vide dans la table de hachage pour que la séquence de test soit finie

1) Essai Linéaire

Exemple:

L'insertion des données suivantes, donne la table ci-après. Le bit 'vide' à 1, indique une case est vide.

$h(a) = 5$

$h(b) = 1$

$h(c) = 3$

$h(d) = 3$ ➔ collision
Calcul de $h(d) - 1 = 2$ ➔ case vide

	Donnée	vide
0		1
1	b	0
2	d	0
3	c	0
4		1
5	a	0
6		1
7		1
8		1
9		1
10		1

1) Essai Linéaire

Exemple:

L'insertion des données suivantes, donne la table ci-après. Le bit 'vide' à 1, indique une case est vide.

$h(a) = 5$

$h(b) = 1$

$h(c) = 3$

$h(d) = 3$ ➔ collision

Calcul de $h(d) - 1 = 2$ ➔ case vide

	Donnée	vide
0		1
1	b	0
2	d	0
3	c	0
4		1
5	a	0
6		1
7		1
8		1
9		1
10		1

1) Essai Linéaire

h(e) = 0

h(f) = 2 ➔ collision
Calcul de h(f) - 1 = 1 ➔ case pleine
Calcul de h(f) - 2 = 0 ➔ case pleine
Calcul de h(f) - 3 = -1
// si (h(x) - i < 0) alors
calcul de (h(f)-i) + N
calcul de h(f) - 3 + 11 = 10 ➔ case vide

h(g) = 8

	Donnée	vide
0	e	0
1	b	0
2	d	0
3	c	0
4		1
5	a	0
6		1
7		1
8	g	0
9		1
10	f	0

1) Essai Linéaire

h(e) = 0

h(f) = 2 ➔ collision

Calcul de h(f) - 1 = 1 ➔ case pleine

Calcul de h(f) - 2 = 0 ➔ case pleine

Calcul de h(f) - 3 = -1

// si (h(x) - i < 0) alors

calcul de (h(f)-i) + N

calcul de h(f) - 3 + 11 = 10 ➔ case vide

h(g) = 8

	Donnée	vide
0	e	0
1	b	0
2	d	0
3	c	0
4		1
5	a	0
6		1
7		1
8	g	0
9		1
10	f	0

1) Essai Linéaire

h(a) = 5 => adresse primaire
h(b) = 1 => adresse primaire
h(c) = 3 => adresse primaire
h(d) = 3 => adresse secondaire (collision)
h(e) = 0 => adresse primaire
h(f) = 2 => adresse secondaire (collision)
h(g) = 8 => adresse primaire

	Donnée	vide
0	e	0
1	b	0
2	d	0
3	c	0
4		1
5	a	0
6		1
7		1
8	g	0
9		1
10	f	0

La recherche de x (tel que h(x) = 2) s'arrête avec un échec dans la case vide d'adresse 9
=> la séquence de test est : 2,1,0,10,9

Si on devait insérer x, la donnée serait affectée à la case 9 (si c'est pas la dernière case vide).

La table est remplie quand le nombre d'éléments insérés égale à N-1 ➔ sacrifice d'une case vide

1) Essai linéaire

- La **suppression** d'un élément x, **génère une case vide**

  ➔ **suppression physique**

- Cette nouvelle case vide risque de rendre d'autres **données inaccessibles**.

  Dans l'exemple précédent, si on supprime b en vidant la case 1, on perd du même coup la donnée f (car elle n'est plus accessible)

  ➔ **faire des tests avant de vider une case**

	Donnée	vide
0	e	0
1	b	0
2	d	0
3	c	0
4		1
5	a	0
6		1
7		1
8	g	0
9		1
10	f	0

1) Essai linéaire

- Le principe de la suppression d'une donnée x est donc :
  - Rechercher l'adresse j de x
  - Tester toutes les cases (i=j-1, j-2, ...) au dessus de j (circulairement) jusqu'à trouver une case vide et vérifier que les données qu'elles contiennent ne vont pas être perdues quand la case j sera vidée
  - Si c'est le cas, on vide la case j et on s'arrête
  - Sinon, dès qu'on trouve une donnée « qui pose problème » on la déplace dans la case j et on tente de vider son emplacement (en testant les cases au dessus qui n'ont pas encore été testées). C'est le même principe qu'on vient d'appliquer pour la case j.

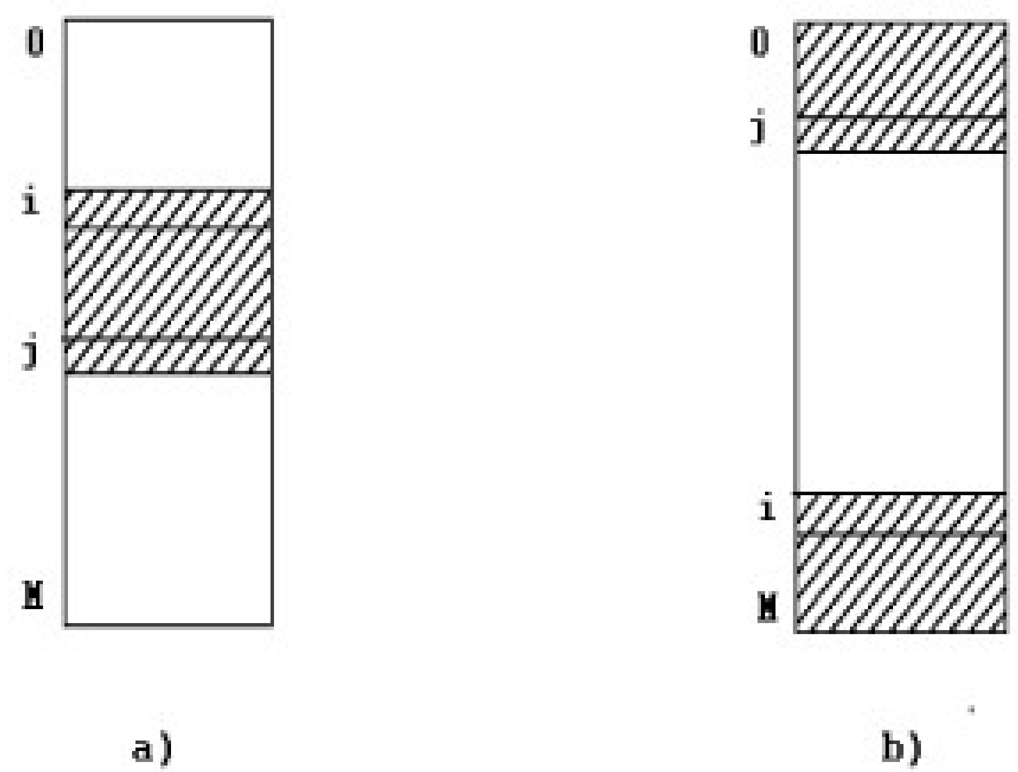
1) Essai linéaire

Soit i l’adresse de l’élément à supprimer:

1. Rendre T(i) vide
   Poser j=i
2. i=i-1
   si i<0 alors i = i + M
3. si T(i) est vide alors
   - Algo se termine

   sinon
   - soit r=h(T(i))
   - a) i<j
     - si r<i ou r>=j alors
       - déplacer l’élément (T(j) = T(i))

       finsi
   - b) i>j
     - si j<=r<i alors
       - déplacer l’élément (T(j) = T(i))
4. Recommencer à partir de 2.



2) Double Hachage

- Cette méthode est presque analogue à la précédente mais au lieu que la séquence soit linéaire, elle est construite par une autre fonction de hachage soit h’.
- Soient h(x) la fonction utilisée pour le calcul de l'adresse primaire et h'(x) la seconde fonction de hachage qui calcule le pas de la séquence:
  h(x), h(x)-h'(x), h(x)-2h'(x), h(x)-3h'(x),...
- Pour que la séquence soit circulaire, les soustractions se font modulo N (la taille de la table)

  // C-a-d quand on calcule le nouvel indice i := i – h'(x), on rajoute juste après le test: SI (i < 0)   i := i + N FSI

- Pour que la couverture soit totale (passer par toutes les cases), il faut choisir la taille N de la table un **nombre premier**
- Pour simplifier les algorithmes, on sacrifie une case de la table

  //C-a-d il reste toujours au moins une case vide (critère d'arrêt)

2) Double Hachage

<u>Exemple</u>:

Soit T une table de 6 éléments.
Le bit 'vide' à 1, indique une case vide.

L'insertion des données suivantes d’après la première fonction de hachage h:

h(a) = 3
h(b) = 2
h(c) = 3
h(d) = 2
h(e) = 1

et en plus avec h’ la deuxième fonction de hachage:

h'(c) = 3
h'(d) = 1
h'(e) = 3

2) Double Hachage

h(a) = 3 ➔ inséré à la case 3

h(b) = 2 ➔ inséré à la case 2

h(c) = 3 ➔ collision à la case 3
on a h’(c) = 3
calcul de h(c)-h’(c) = 0
inséré à la case 0

h(d) = 2 ➔ collision à la case 2
on a h’(d) = 1
calcul de h(d)-h’(d)=1
inséré à la case 1

h(e) = 1 ➔ collision à la case 1
on a h’(e) = 3
calcul de h(e)-h’(e)=1-3=-2
soit i=h(e)-h’(e)
si i<0 alors i=i+N donc i=4 (avec N=6)
inséré à la case 4

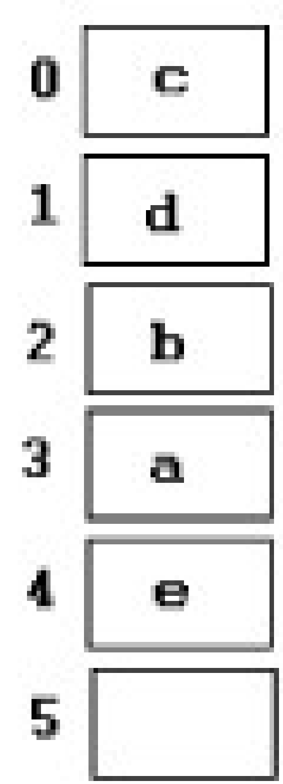
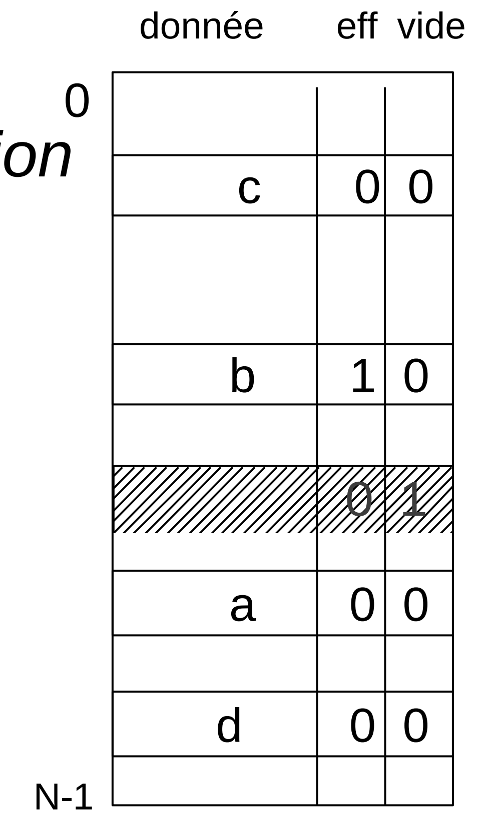


Table T

2) Double Hachage

Le principe de la suppression est analogue à celui de l'essai linéaire.
Un moyen simple consiste à faire une *suppression logique*, c'est à dire le positionnement d'un bit qu'on rajoute au niveau des entrées de la table.



Chaque case renferme 2 bits :

vide : indiquant une case vide

eff : indiquant un effacement logique
(**la case n'est pas vide)**
ex: b est supprimée logiquement

Pour récupérer l'espace perdu à cause des effacements logiques, on effectue des **réorganisations périodiques**

➔ réinsérer les données non effacées dans une nouvelle table

Estimation des débordements

Soit une table de **N** cases, et on aimerait insérer **r** données

Le pourcentage de remplissage (la densité) est donc: **$d = r / N$**

Soit P(x) la probabilité que x données parmi r soient « hachées » vers la même case

$$P(x) = C_r^x \, (1 - 1/N)^{r-x} \, (1/N)^x$$

L'inconvénient de la formule est qu'elle est difficile à calculer pour N et r grands. La fonction de *POISSON* est une bonne approximation.

$$P(x) = (d^x * e^{-d}) / x! \qquad \text{(avec } d = r/N\text{)}$$

N*P(x) est donc une estimation du nombre de cases ayant été choisies x fois durant l'insertion des r données dans la table

Le nombre total de données en débordement est alors estimé à :

$$N(P(2) + 2*P(3) + 3*P(4) + 4*P(5) + ...)$$

Estimation débordements

<u>Exemple numérique</u>:

Lors de l'insertion de r = 1000 données dans une table de N = 1000 cases

densité = d = r/N = 1)

on estime que :
N.P(0) = 368 cases ne recevront aucune données
N.P(1) = 368 cases auront été choisies 1 seule fois
N.P(2) = 184 cases auront été choisies 2 fois
N.P(3) = 61 cases auront été choisies 3 fois
N.P(4) = 15 cases auront été choisies 4 fois
N.P(5) = 3 cases auront été choisies 5 fois
N.P(6) = 0 cases auront été choisies 6 fois

Estimation débordements

<u>Exemple numérique</u>:

Avec la densité d=1 (r = 1000 et N = 1000)

Le nombre de données en débordement est proche de :
1000(0.184 + 2*0.061 + 3*0.015 + 4*0.003) = 363

soit **36,3%** des données

(r/nombre de données en débordement = 1000/363 = 36.3)

contre 631 données dans leurs adresses primaires calculés:
1000(0.368 + 0.184 + 0.061 + 0.015 + 0.003) = 631

soit **63.1%** des données

Conclusion

Les méthodes de hachage donnent des résultats excellents en moyenne O(1), mais lamentables dans le pire cas O(n), car il n'est pas possible d'éviter les collisions.

En particulier, le choix de la fonction de hachage est fondamental.

Il existe plusieurs façons de réduire les collisions :

- trouver une fonction qui distribue bien les données c'est a dire de façon aléatoire.

- augmenter l'espace des adresses possibles.

<u>Exemple</u>: d = 0.5 (N = 1000 et r = 500 données)

Si la densité est de 50 %, on peut s'attendre à 79% de données rangées dans leur adresse primaire et 21 % rangées ailleurs.

- mettre plus d'une donnée par adresse possible (si on accepte par exemple b données par adresse donc d = r/(b*N).