Listes triées et listes doublement chaînées

Algorithmique et structures de données, 2022-2023

P. Albuquerque (B410), P. Künzli et O. Malaspinas (A401), ISC, HEPIA

2023-01-11

En partie inspirés des supports de cours de P. Albuquerque

Les listes triées

Une liste chaînée triée est:

- une liste chaînée
- dont les éléments sont insérés dans l'ordre.

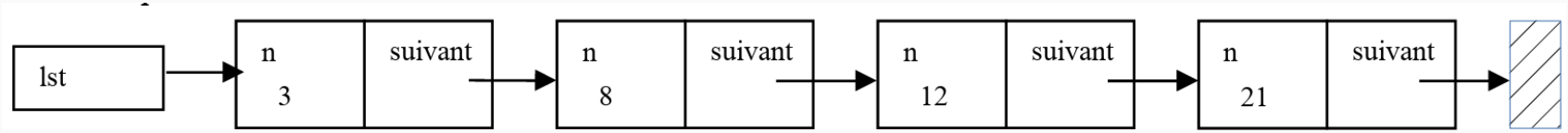

Figure 1: Exemple de liste triée.

Les listes triées

Une liste chaînée triée est:

- une liste chaînée
- dont les éléments sont insérés dans l'ordre.

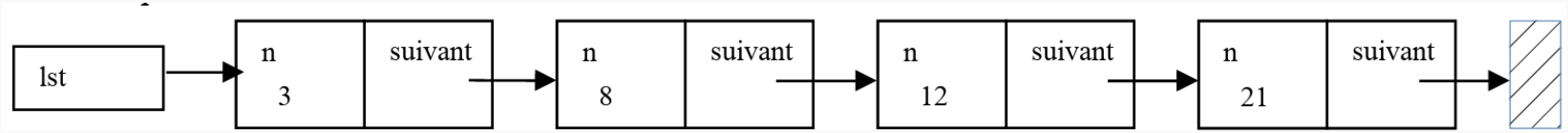

Figure 1: Exemple de liste triée.

- L'insertion est faite telle que l'ordre est maintenu.

Quelle structure de données?

Les listes triées

Quel but?

- Permet de retrouver rapidement un élément.
- Utile pour la recherche de plus court chemin dans des graphes.
- Ordonnancement de processus par degré de priorité.

Comment?

- Les implémentations les plus efficaces se basent sur les tableaux.
- Possibles aussi avec des listes chaînées.

Les listes triées

Quelle structure de données dans notre cas?

Une liste chaînée bien sûr (oui c'est pour vous entraîner)!

```
typedef struct _element { // chaque élément
    int data;
    struct _element *next;
} element;
typedef element* sorted_list; // la liste
```

Fonctionnalités

```
// insertion de val
sorted_list sorted_list_push(sorted_list list, int val);
// la liste est-elle vide?
bool is_empty(sorted_list list); // list == NULL
// extraction de val (il disparaît)
sorted_list sorted_list_extract(sorted_list list, int val);
 // rechercher un élément et le retourner
element* sorted_list_search(sorted_list list, int val);
```

L'insertion

Trois cas

1. La liste est vide.

L'insertion

Trois cas

1. La liste est vide.

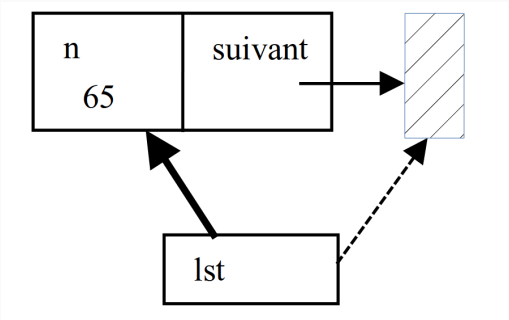

Figure 2: Insertion dans une liste vide, `list == NULL`.

L'insertion

Trois cas

1. La liste est vide.

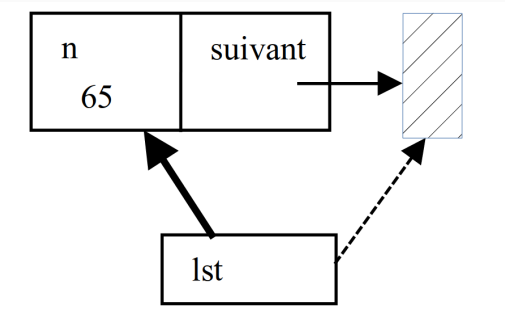

Figure 2: Insertion dans une liste vide, `list == NULL`.

```
sorted_list sorted_list_push(sorted_list list, int val) {
    if (sorted_list_is_empty(list)) {
        list = malloc(sizeof(*list));
        list->data = val;
        list->next = NULL;
        return list;
    }
}
```

L'insertion

2. L'insertion se fait en première position.

L'insertion

2. L'insertion se fait en première position.

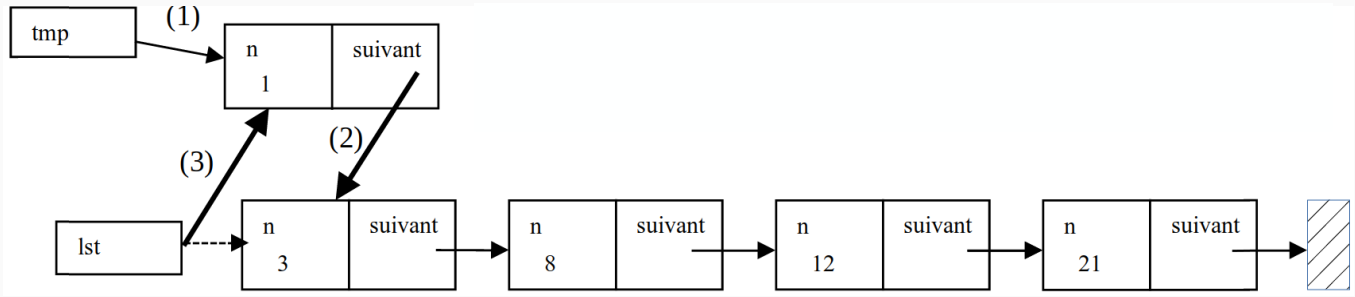

Figure 3: Insertion en tête de liste, `list->data >= val`.

L'insertion

2. L'insertion se fait en première position.

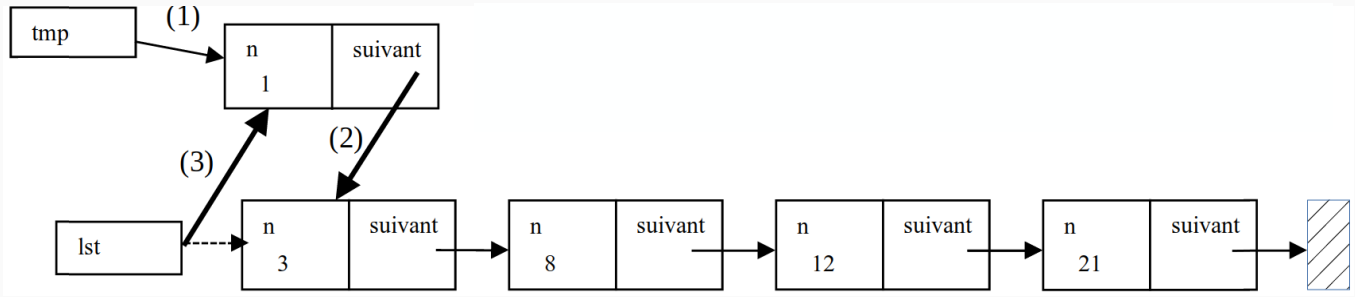

Figure 3: Insertion en tête de liste, `list->data >= val`.

```
sorted_list sorted_list_push(sorted_list list, int val) {
    if (list->data >= val) {
        element *tmp = malloc(sizeof(*tmp));
        tmp->data = val;
        tmp->next = list;
        list = tmp;
        return list;
    }
}
```

L'insertion

3. L'insertion se fait sur une autre position que la première.

L'insertion

3. L'insertion se fait sur une autre position que la première.

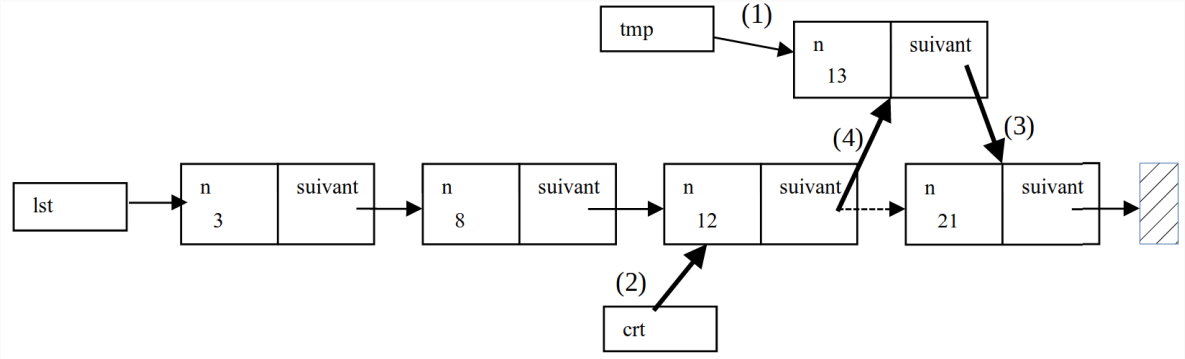

Figure 4: Insertion sur une autre position, list->data < val.

L'insertion

3. L'insertion se fait sur une autre position que la première.

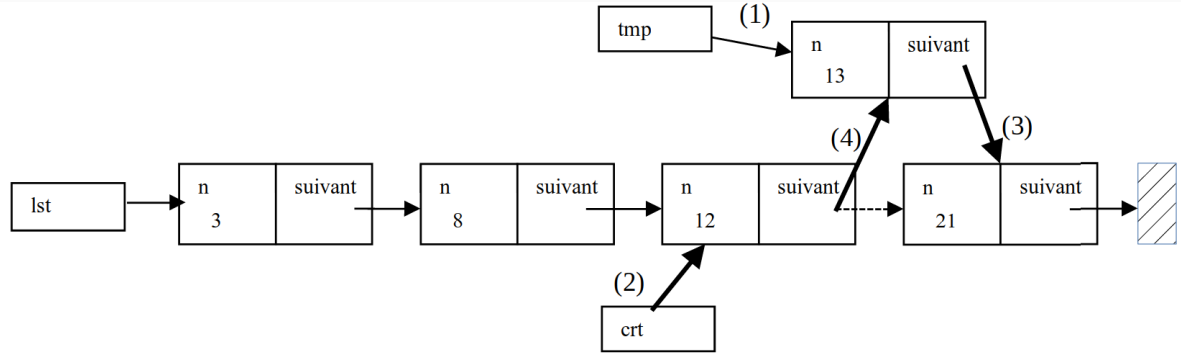

Figure 4: Insertion sur une autre position, list->data < val.

```
sorted_list sorted_list_push(sorted_list list, int val) {
    element *tmp = malloc(sizeof(*tmp));
    tmp->data = val;
    element *crt = list;
    while (NULL != crt->next && val > crt->next->data) {
        crt = crt->next;
    }
    tmp->next = crt->next;
    crt->next = tmp;
    return list;
}
```

L'extraction

Trois cas

1. L'élément à extraire n'est **pas** le premier élément de la liste

L'extraction

Trois cas

1. L'élément à extraire n'est **pas** le premier élément de la liste

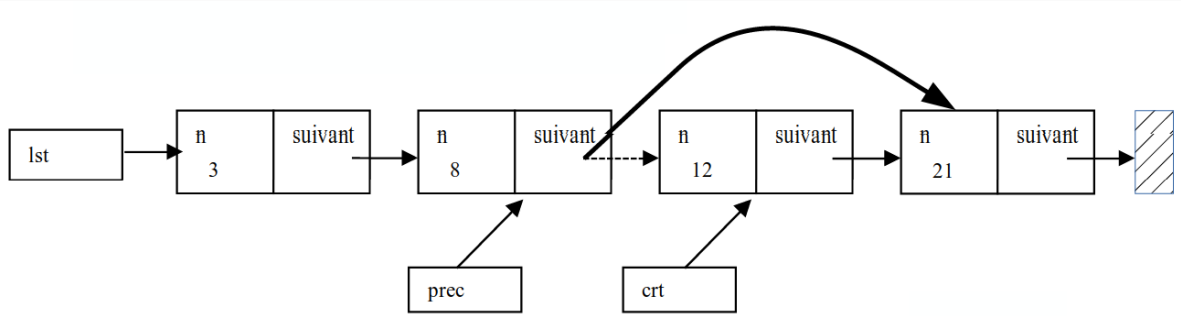

Figure 5: Extraction d'un élément qui n'est pas le premier.

L'extraction

Trois cas

1. L'élément à extraire n'est **pas** le premier élément de la liste

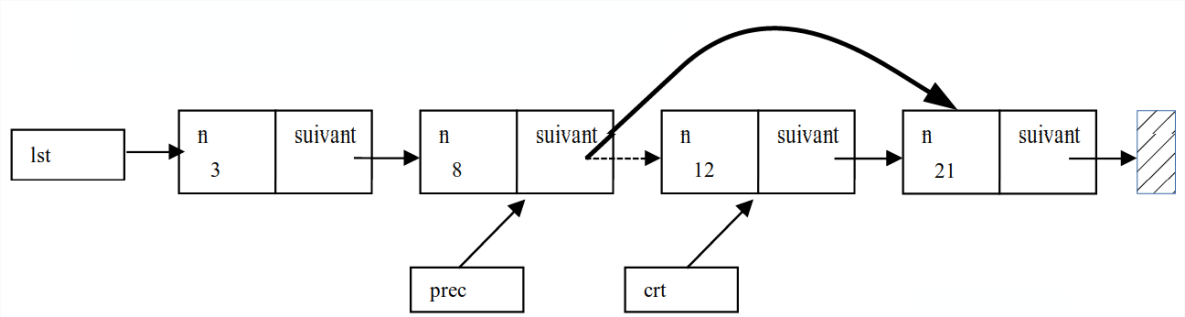

Figure 5: Extraction d'un élément qui n'est pas le premier.

```
sorted_list sorted_list_extract(sorted_list list, int val) {
    element *prec = *crt = list; // needed to glue elements together
    while (NULL != crt && val > crt->data) {
       prec = crt;
       crt = crt->next;
    }
    if (NULL != crt && prec != crt && crt->data == val) { // glue things together
        prec->next = crt->next;
        free(crt);
    }
    return list;
}
```

L'extraction

2. L'élément à extraire est le premier élément de la liste

L'extraction

2. L'élément à extraire est le premier élément de la liste

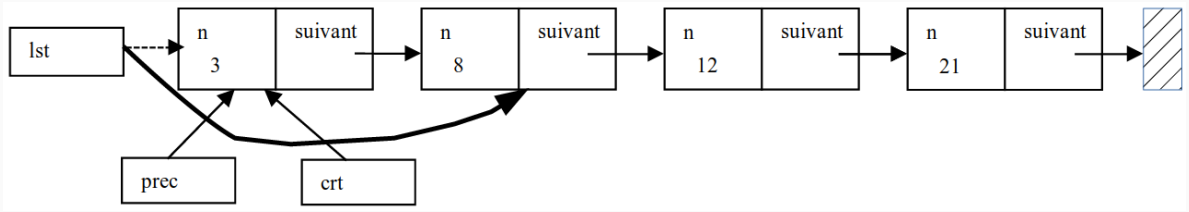

Figure 6: Extraction d'un élément qui est le premier.

L'extraction

2. L'élément à extraire est le premier élément de la liste

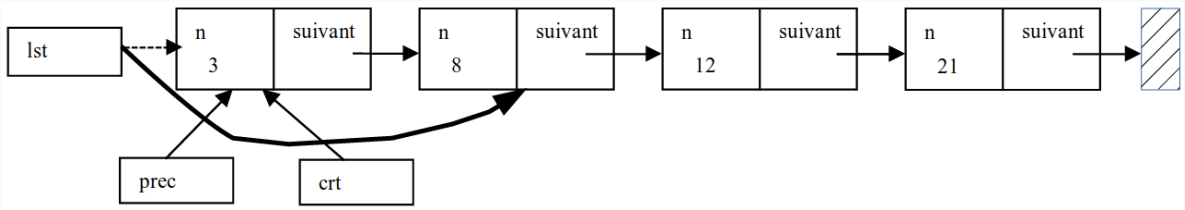

Figure 6: Extraction d'un élément qui est le premier.

```
sorted_list sorted_list_extract(sorted_list list, int val) {
    element *prec = *crt = list; // needed to glue elements together
    while (NULL != crt && val > crt->data) {
       prec = crt;
       crt = crt->next;
    }
    if (NULL != crt && crt->data == val && prec == crt) { // glue things togethe
        list = list->next;
        free(crt);
    }
    return list;
}
```

L'extraction

3. L'élément à extraire n'est **pas** dans la liste.
   - La liste est vide.
   - La valeur est plus grande que le dernier élément de la liste.
   - La valeur est plus petite que la valeur de `crt`.

L'extraction

3. L'élément à extraire n'est **pas** dans la liste.
   - La liste est vide.
   - La valeur est plus grande que le dernier élément de la liste.
   - La valeur est plus petite que la valeur de `crt`.

On retourne la liste inchangée.

L'extraction

3. L'élément à extraire n'est **pas** dans la liste.
   - La liste est vide.
   - La valeur est plus grande que le dernier élément de la liste.
   - La valeur est plus petite que la valeur de `crt`.

On retourne la liste inchangée.

```
sorted_list sorted_list_extract(sorted_list list, int val) {
    element *prec = *crt = list; // needed to glue elements together
    while (NULL != crt && val > crt->data) {
       prec = crt;
       crt = crt->next;
    }
    if (NULL == crt || crt->data != val) { // val not present
        return list;
    }
}
```

La recherche

```
element* sorted_list_search(sorted_list list, int val);
```

- Retourne `NULL` si la valeur n'est pas présente (ou la liste vide).
- Retourne un pointeur vers l'élément si la valeur est présente.

La recherche

```
element* sorted_list_search(sorted_list list, int val);
```

- Retourne NULL si la valeur n'est pas présente (ou la liste vide).
- Retourne un pointeur vers l'élément si la valeur est présente.

```
element* sorted_list_search(sorted_list list, int val) {
    // search for element smaller than val
    element* pos = sorted_list_position(list, val);
    if (NULL == pos && val == list->data) {
        return list; // first element contains val
    } else if (NULL != pos && NULL != pos->next && val == pos->n
        return pos->next; // non-first element contains val
    } else {
        return NULL; // well... val's not here
    }
}
```

La recherche

La fonction `sorted_list_position`

```
element* sorted_list_position(sorted_list list, int val);
```

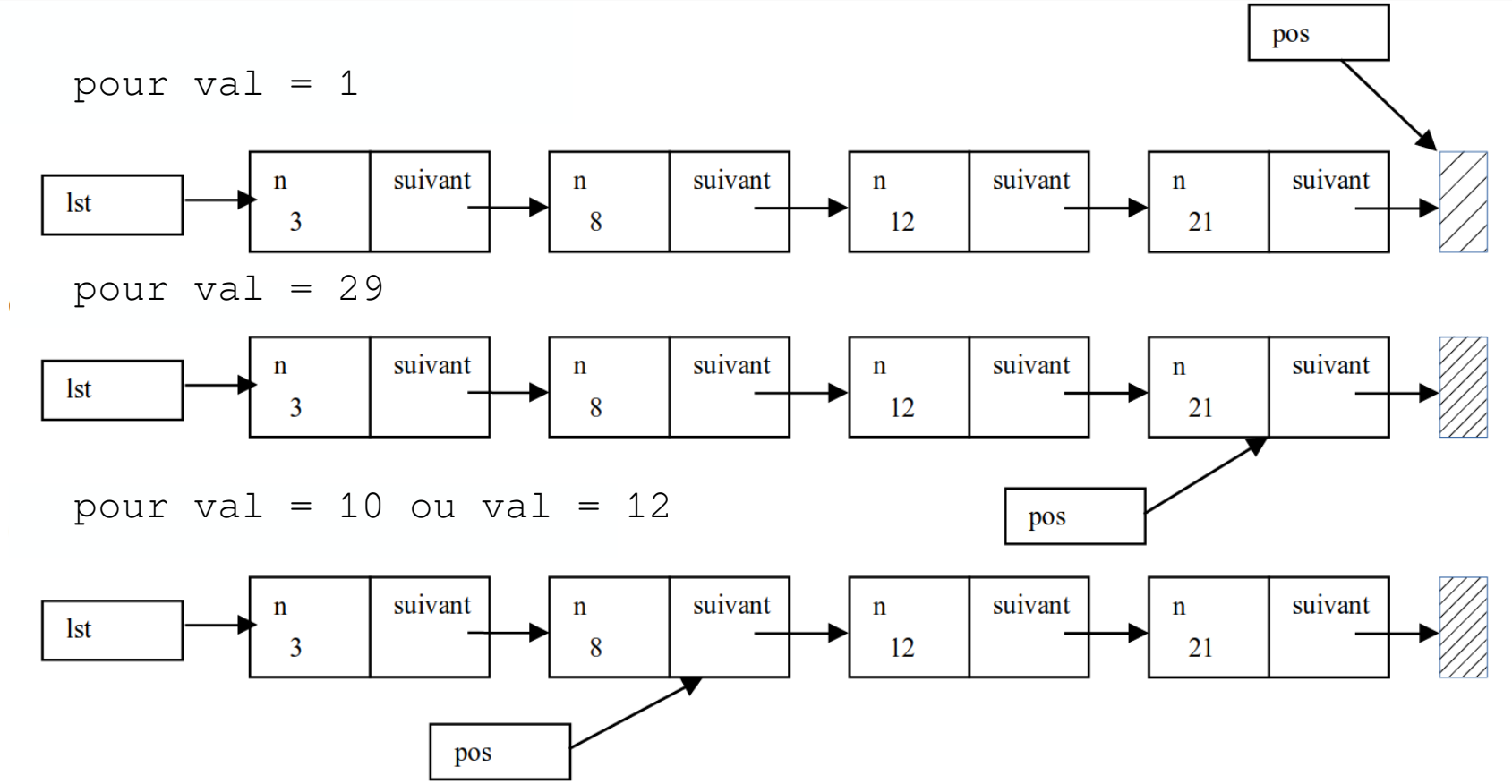

Figure 7: Trois exemples de retour de la fonction `sorted_list_position()`.

La recherche

Exercice: implémenter

```
element* sorted_list_position(sorted_list list, int val);
```

La recherche

Exercice: implémenter

```
element* sorted_list_position(sorted_list list, int val);
```

```
element* sorted_list_position(sorted_list list, int val) {
    element* pos = list;
    if (sorted_list_is_empty(list) || val <= list->data) {
        pos = NULL;
    } else {
        while (NULL != pos->next && val > pos->next->data) {
            pos = pos->next;
        }
    }
    return pos;
}
```

Complexité de la liste chaînée triée

L'insertion?

Complexité de la liste chaînée triée

L'insertion?

$$\mathcal{O}(N).$$

L'extraction?

Complexité de la liste chaînée triée

L'insertion?

$$\mathcal{O}(N).$$

L'extraction?

$$\mathcal{O}(N).$$

La recherche?

Complexité de la liste chaînée triée

L'insertion?

$$\mathcal{O}(N).$$

L'extraction?

$$\mathcal{O}(N).$$

La recherche?

$$\mathcal{O}(N).$$

Liste doublement chaînée

Application: navigateur ou éditeur de texte

- Avec une liste chaînée:
  - Comment implémenter les fonctions `back` et `forward` d'un navigateur?
  - Comment implémenter les fonctions `undo` et `redo` d'un éditeur de texte?

Liste doublement chaînée

Application: navigateur ou éditeur de texte

- Avec une liste chaînée:
  - Comment implémenter les fonctions `back` et `forward` d'un navigateur?
  - Comment implémenter les fonctions `undo` et `redo` d'un éditeur de texte?

Pas possible.

Solution?

Liste doublement chaînée

Application: navigateur ou éditeur de texte

- Avec une liste chaînée:
  - Comment implémenter les fonctions `back` et `forward` d'un navigateur?
  - Comment implémenter les fonctions `undo` et `redo` d'un éditeur de texte?

Pas possible.

Solution?

- Garder un pointeur supplémentaire sur l'élément précédent et pas seulement le suivant.

Liste doublement chaînée

Application: navigateur ou éditeur de texte

- Avec une liste chaînée:
  - Comment implémenter les fonctions `back` et `forward` d'un navigateur?
  - Comment implémenter les fonctions `undo` et `redo` d'un éditeur de texte?

Pas possible.

Solution?

- Garder un pointeur supplémentaire sur l'élément précédent et pas seulement le suivant.
- Cette structure de donnée est la **liste doublement chaînée** ou **doubly linked list**.

Liste doublement chaînée

Exercices

- Partir du dessin suivant et par **groupe de 5**

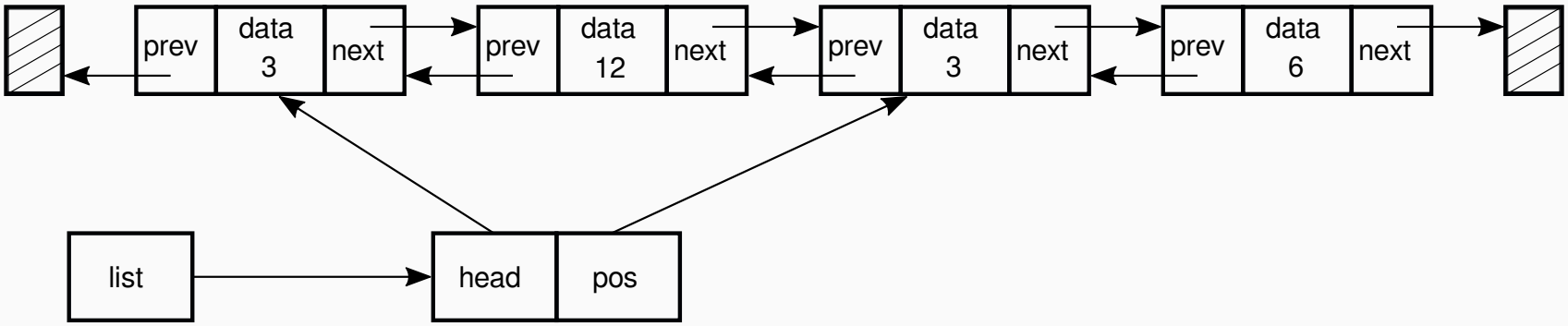

Figure 8: Un schéma de liste doublement chaînée d'entiers.

1. Écrire les structures de données pour représenter la liste doublement chaînée dont le type sera `dll` (pour `doubly_linked_list`)

Liste doublement chaînée

2. Écrire les fonctionnalités de création et consultation

```
// crée la liste doublement chaînée
dll dll_create();
// retourne la valeur à la position actuelle dans la liste
int dll_value(dll list);
// la liste est-elle vide?
bool dll_is_empty(dll list);
// Est-ce que pos est le 1er élément?
bool dll_is_head(dll list);
// Est-ce que pos est le dernier élément?
bool dll_is_tail(dll list);
// data est-elle dans la liste?
bool dll_is_present(dll list, int data);
// affiche la liste
void dll_print(dll list);
```

Liste doublement chaînée

3. Écrire les fonctionnalités de manipulation

```
// déplace pos au début de la liste
dll dll_move_to_head(dll list);
// déplace pos à la position suivante dans la liste
dll dll_next(dll list);
// déplace pos à la position précédente dans la liste
dll dll_prev(dll list);
```

Liste doublement chaînée

4. Écrire les fonctionnalités d'insertion

```
// insertion de data dans l'élément après pos
dll dll_insert_after(dll list, int data);
// insertion de data en tête de liste
dll dll_push(dll list, int data);
```

5. Écrire les fonctionnalités d'extraction

```
// extraction de la valeur se trouvant dans l'élément pos
// l'élément pos est libéré
int dll_extract(dll *list);
// extrait la donnée en tête de liste
int dll_pop(dll *list);
// vide la liste
void dll_destroy(dll *list)
```