TD9 - Structure de données

Code	LU2IN006
Type de cours	Travaux dirigés
Complété ?	☐
Jour du cours	@03/04/2023

`PINHO FERNANDES Enzo - L2 Mono-Info S4 - GR6`

TD9 : Skip list et arbre rouge-noir

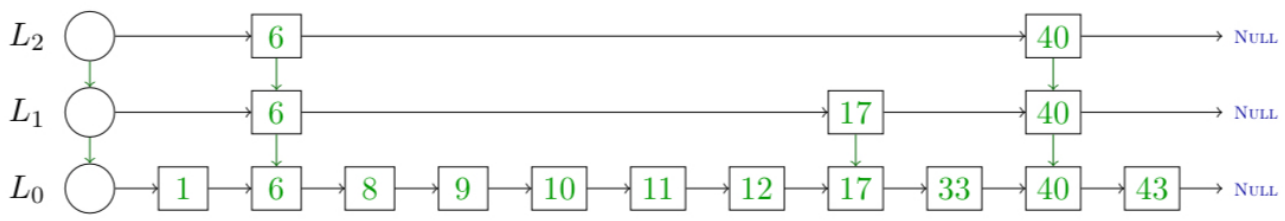
…	Recherche	Ajout	Suppression
Tableau	$O(log(n))$	$O(n)$	$O(n)$
Liste chaînée	$O(n)$	$O(1)$	$O(1)$

▼ Exercice 1 : Skip list

📖 **Enoncé :**

Exercice 1 – Skiplist

Une skip list (ou liste à enjambements, ou liste à saut) est une structure de données *probabiliste* permettant d'implémenter une liste triée. Cette structure de données est organisée en couches, où chaque couche est une liste chaînée. La première couche, notée L_0, correspond à la liste triée. Les couches supérieures $(L_n, n \geq 0)$ sont construits récursivement ainsi : un élément e de la couche L_n a une probabilité p de faire partie de la couche L_{n+1}. Le nombre de couches d'une skip list est donc aléatoire. Voici une skip list représentant la liste triée $(1, 6, 8, 9, 10, 11, 12, 17, 33, 40, 43)$:



Les couches supérieures permettent des *raccourcis* dans la liste quand on cherche un élément, comme le montre l'exemple suivant (recherche de la valeur 33) :

Q 1.1 À quoi correspond une skip list avec $p = 0$? avec $p = 1$?

Q 1.2 On pourrait imaginer une skip list déterministe, où les raccourcis ne seraient pas choisis aléatoirement mais plutôt de manière fixe. Quelle serait la forme d'une skip list déterministe qui simulerait une recherche par dichotomie ? En pensant principalement à l'insertion de nouveaux éléments, quel problème poserait une telle structure ?

Q 1.3 Proposer une structure C pour représenter une skip list.

Q 1.4 Implémenter une fonction `void skiplist_print(Skiplist *sl)` qui affiche le contenu de la skip list. Par exemple, pour la skip list de l'exemple, la sortie serait :

```
3:->6->40
2:->6->17->40
1:->1->6->8->9->10->11->12->17->33->40->43
```

Q 1.5 Décrire succinctement l'algorithme d'insertion d'un entier `val` dans une skip list. Pour simplifier, on supposera que l'entier `val` n'appartient pas à la skip list.

Q 1.6 Écrivez une fonction `void skiplist_add(skiplist *sl, int val)` qui insère un élément dans une skip list. On supposera existante une fonction `int isRandomOK(float p)` qui rend `1` si l'évènement de probabilité p s'est réalisé et `0` sinon.

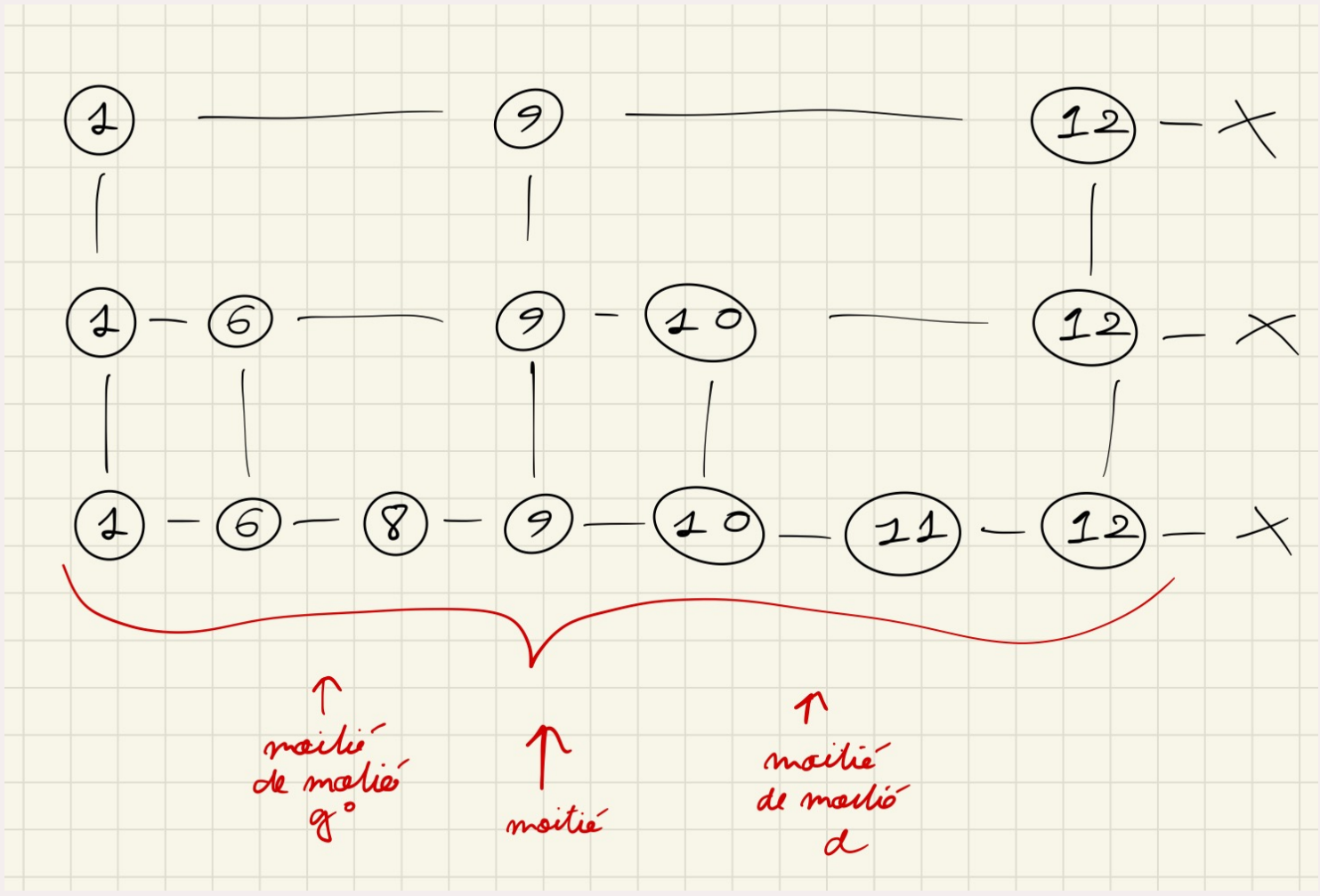
Réponses :

Question 1 :

- $p = 0$: Liste chaînée simple.
- $p = 1$: Infinité de couches donc impossible.

Question 2 :

- Insérer un élément demande de modifier les autres éléments sur les couches supérieures.



Question 3 :

```
typedef struct Cell{
  int value;
  Cell *right;
  Cell *above;
  Cell *left;
  Cell *below;
} Cell;

typedef struct Layer{
  Cell *first;
  Layer *above;
  Layer *below;
} Layer

typedef struct Skiplist{
  Layer *top;
  Layer *bottom;
}
```

Question 4 :

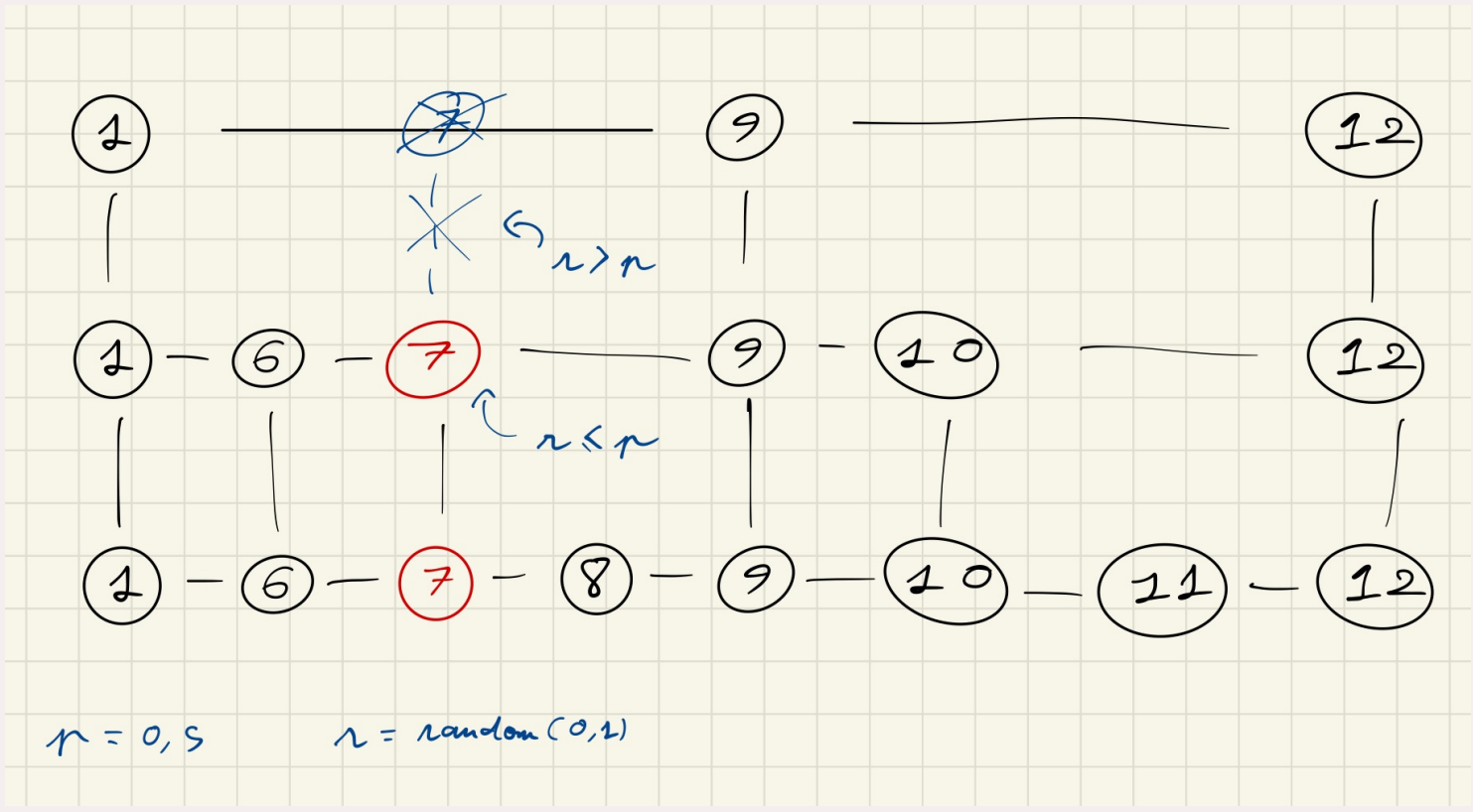
```
void skiplist_print(Skiplist *sl){
  Layer *curr = sl->top;
  while(curr != NULL){
    layer_print(curr);
    printf("\n");
    curr = curr->below;
  }
}

void layer_print(Layer *l){
  printf("%d: ", l->pos);
  Cell *curr = l->first;
  while(curr != NULL){
    printf("%d ", curr->value);
    curr = curr->right;
```

```
  }
}
```

▼ **Question 5 :**

- On insère val dans L_0 (à la bonne position)
- On tire un nombre aléatoire entre 0 et 1.
  - S'il est plus grand, on s'arrête.
  - S'il est plus petit :
    - On insère val dans L_1 et on recommence l'étape 2. Il faudra éventuellement créer la couche si elle n'existe pas.



▼ **Question 6 :**

```
void skiplist_insert(Skiplist *sl, int val){
  Cell *c1 = layer_insert(sl->bottom, val);
  Layer *curr = sl->bottom;
  while(isRandomOk(p)){
    if(curr->above == NULL){
      Layer *l_new = layerInit(curr->pos + 1);
      l_new->below = curr;
      curr->above = l_new;
    }

    Cell *c2 = layer_insert(curr-above, val);
    c1->above = c2;
    c2->below = c1;
    curr = curr->above;
    c1 = c2;
  }
}
```

▼ Exercice 2 : Arbre rouge-noir

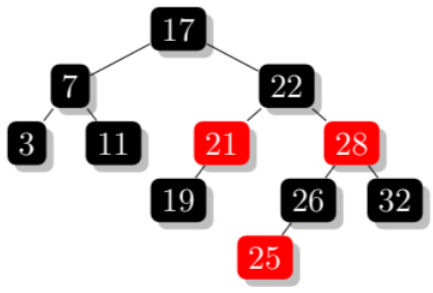
📖 **Enoncé :**

Exercice 2 – Arbre Rouge-Noir

Un arbre rouge-noir (ARN) :

- est un arbre binaire de recherche,
- dont les nœuds sont colorés en noir ou en rouge,
- la racine étant colorée en noir,
- les nœuds fils d'un nœud rouge étant colorés en noir,
- si un noeud a moins de deux fils, ont lui ajoute des fils fictifs noirs,
- et le nombre de nœuds noirs sur tous les chemins de la racine à une feuille est constant.

La figure ci-dessous représente un ARN (les seuls sommets rouges sont les sommets 21, 25 et 28).



Q 2.1 Définir une structure permettant de représenter un nœud d'ARN.

Q 2.2 Écrire une fonction C permettant de calculer le nombre de nœuds rouges dans un ARN.

Q 2.3 Écrire une ou des fonctions C permettant de vérifier qu'un `ARNtree` vérifie bien les propriétés d'un arbre rouge-noir.

Q 2.4 Rappelez brièvement comment on réalise une insertion dans un ARN. Insérer dans l'ARN de la figure les éléments 18, 24 et 23 dans cet ordre.

Q 2.5 Quelle est la complexité de cette fonction ? Quel peut-être l'intérêt de cette structure par rapport à un AVL ?

Réponses :

▼ Question 1 :

```
typedef enum {noir, rouge} Color;

typedef struct NodeRB{
  int val;
  Color color;
  NodeRB *parent;
  NodeRB *child_g;
  NodeRB *child_d;
} Noeud;
```

▼ Question 2 :

```
int nbRouges(NodeRB* root){
  if(root == NULL)
    return 0;
  return (root->color == rouge) + nbRouges(root->child_g) + nbRouges(root->child_d);
}
```

▼ Question 3 :

```
int p3(NodeRB *root) { return root->color == noir; }

int p4(NodeRB *tree){
  if(tree == NULL){
    return 1;
  }

  if(tree->color == rouge){
    if(tree->child_g->color == rouge || tree->child_d->color == rouge){
      return 0;
    }

  if(p4(tree->child_g) == 0 || p4(tree->child_d) == 0)
    return 0;

  return 1;
}
```

▼ Question 4 :

▼ Question 5 :