Algorithmes et structures de données : TD 8

Listes linéaires simplement chaînées - Complexité asymptotique

Rappel :

`New(element)` est de complexité O(1)
`SetLength(tableau, n)` est de complexité O(n)
`SetLength(tableau, 1)` est de complexité O(1)

Exercice 8.1 *Listes linéaire simplement chaînée*

Considérer l'algorithme 1 suivant qui crée une liste de n éléments :

```
{Algorithme 1}
type
        p_t_liste_simple = ^t_liste_simple;
        t_liste_simple = record
                cle     : integer;
                suivant : p_t_liste_simple;
        end;
var element : p_t_liste_simple;
var temp    : p_t_liste_simple;
var premier : p_t_liste_simple;
var i : integer;

début
        New(element);
        element^.suivant := NIL;
        element^.cle := 0;
        premier := element;

        temp := element;
        pour i de 1 à n-1 faire
        début
                New(element);
                element^.cle := i*i;
                element^.suivant := NIL;
                temp^.suivant := element;
                temp := element;
        fin
fin
```

1. Quelle est la complexité de cet algorithme (notation Grand-O) ?
2. Ebaucher l'occupation de la mémoire **d'une manière compacte**.
3. Ecrire un algorithme qui parcours la liste et affiche la `cle` de chaque élément à l'écran. Qu'est-ce qui est affiché à l'écran ?
4. Ecrire un algorithme qui rajoute un élément supplémentaire avec la clé 1000 **au début** de la liste (la liste aura n+1 éléments). Quelle est la complexité de votre algorithme ? Comparer avec l'exercice précédente sur les tableaux dynamiques.
5. Ecrire un algorithme qui rajoute un élément supplémentaire avec la clé 1000 **à la fin** de la liste. Quelle est la complexité de votre algorithme ? Comparer avec l'exercice précédente sur les tableaux dynamiques.
6. Que faut-il rajouter à l'algorithme 1 ci-dessus pour réduire la complexité de rajouter un élément supplémentaire **à la fin** de la liste ? Ecrivez l'algorithme !
7. Ecrire un algorithme qui parcours la supprime la première occurence d'un élément avec la clé `clesup` (il faudra changer des pointeurs ainsi qu'utiliser `Dispose`). Quelle est la complexité de votre algorithme.
8. Ecrire un algorithme qui parcours la liste et libère la mémoire occupé par la liste en utilisant `Dispose`.

Exercice 8.2 *Pointeurs*

Rappeler vous de l'exercice sur les musiciens de la dernière séance (TD 6.3), et considérer maintenant l'algorithme suivant :

```
type p_t_musicien = ^t_musicien;
     t_musicien = record
                          cle     : integer;
                          nom     : string;
                          annee   : integer;
                          suivant : p_t_musicien;
                  end;

var guitariste  : p_t_musicien;
var bassiste    : p_t_musicien;
var chanteur    : p_t_musicien;
var musicien    : p_t_musicien;

var i : integer;

begin

  New(guitariste);
  New(bassiste);
  New(chanteur);

  guitariste^.cle := 1;
  guitariste^.nom := 'Robert';
  guitariste^.annee := 1982;
  guitariste^.suivant := NIL;

  bassiste^.cle := 2;
  bassiste^.nom := 'Sebastian';
  bassiste^.annee := 1979;
  bassiste^.suivant := NIL;

  chanteur^.cle := 3;
  chanteur^.nom := 'Rainer';
  chanteur^.annee := 1984;
  chanteur^.suivant := NIL;

  guitariste^.suivant := bassiste;
  bassiste^.suivant := chanteur;
  chanteur^.suivant := guitariste;
```

```
  musicien := guitariste;

  { Endroit 1 }

  for i := 1 to 10 do
  begin
    write('nom ', musicien^.nom);
    writeln('  annee  ', musicien^.annee);
    musicien := musicien^.suivant;
  end;

  readln;

  Dispose(guitariste);
  Dispose(bassiste);
  Dispose(chanteur);

end.
```

1. Ebaucher l'occupation de la mémoire à l'endroit 1 (version étendu).
2. Quelle est la différence entre cet algorithme et celui de la dernière séance ?
3. Comparé à l'algorithme de la dernière séance (TD 6.3), lequel des deux algorithme occupe moins de la place dans la mémoire et est donc plus efficace en terme de mémoire ? Pourquoi ?
4. Faites tourner cet algorithme dans un tableau.
5. Qu'est-ce qui affiche cet algorithme à l'écran ?