Algorithmique et programmation

Listes chaînées

R.Gosswiller

Sommaire

Séquences ordonnées

Limites des tableaux

Détails

Allocation contraignante
Taille nécessaire à connaître
Difficulté à calculer la taille
Reallocation peu fiable

Limites des tableaux

Problème

Comment adapter au plus juste la mémoire consommée par un ensemble ?

Solution

Réallouer la mémoire au coup par coup
Travailler avec des ensembles adaptables

Structures auto-référencées

Principe

Il est possible de concevoir une structure contenant un membre de son propre type

Solution

```
1 struct str_ {
2         struct str_ * member;
3 };
```

Utilisation

Créer une chaîne de références

Structures auto-référencées

Attention

Un typedef ne peut faire référence à lui-même dans une structure !

Exemple (faux)

```
1 typedef struct str_ {
2         str member;
3 }str;
```

Listes chaînées

Les listes chaînées

Principe

Chaque élément est un couple (valeur, lien)
Chaque lien est un pointeur vers l'élément suivant
Modèle à l'image des tableaux dynamiques
Eléments non consécutifs en mémoire

Opérations

Lecture des maillons de la chaîne
Ajout, suppression, insertion, recherche, ...

Les listes chaînées

Syntaxe d'une liste élémentaire

```
1 struct liste_ {
2         int value;
3         struct liste_ * next;
4 };
5 typedef struct liste_ * liste;
```

Détails

Chaque élément contient une valeur et un pointeur

Créer une liste

```
1  liste createNew(int v) {
2          liste elt = malloc(sizeof(struct liste_));
3          elt->value = v;
4          elt->next = NULL;
5
6          return elt;
7  }
8
9  void add(liste l, int new) {
10         if(l==NULL) { printf("error"); return; }
11         while(l->next!=NULL) {
12                 l=l->next;
13         }
14         l->next=createNew(new);
15 }
16
17 liste l = createNew(0);
18 add(l,42);
19 add(l,35);
20 add(l,28);
```

Créer une liste

Attention

Il faut utiliser $l->next$ et non pas l

Rester connecté à la structure et anticiper les modifications

Sinon : Chaînes brisées

Exemple

```
1  void add(liste l, int new) {
2          while(l!=NULL) {
3                  l=l->next;
4          }
5          l=createNew(new);
6      printf("%x\n",&l); // Adresse 1
7  }
8
9  liste l = createNew(0);
10 printf("%x\n",&l); // Adresse 2
11 add(l,42);
```

Opérations sur les listes

Opérations sur les listes

- Affichage
- Ajout en tête, en queue, en n
- Suppression
- Recherche
- Longueur

Les listes chaînées

Affichage

```
1  void printList (liste p)
2  {
3          int i=0;
4          while (p != NULL) {
5                  printf("Element␣%d␣:␣%d\n", i, p->value);
6                  p = p->next;
7                  i++;
8          }
9  }
```

Les listes chaînées

Ajout et Suppression

- Ces procédures se font en agissant que sur deux ou trois maillons
- Pour free() une chaîne, il faut free() chaque élément séparément !
- La difficulté se rencontre généralement quand on agît sur les premiers et derniers éléments

Recherche

```
1  #include <stdbool.h>
2  bool search(liste p, int val) {
3          while (p->next != NULL) {
4                  if(p->value == val) {
5                          return true;
6                  }
7                  p = p->next;
8          }
9          return false;
10 }
```

Les listes chaînées

Listes et sous-structures

```
1  struct voiture {
2          int prix;
3          char * couleur;
4  }
5
6  typedef struct liste_ * {
7          voiture val;
8          struct liste_ * next;
9  } liste;
```

Détails

Chaque élément contient une valeur et un pointeur

On peut créer des listes de structures

Types de listes

Les listes chaînées

Principe

On peut décliner la notion de liste chaînée selon différentes variantes

Types de listes

Listes chaînées
Listes circulaires
Listes doublement chaînées/doublement circulaires

Listes chaînées circulaires

Principe

Liste chaînée où le dernier élément est connecté au premier
Pas de notion de tête/queue
Accès permanent à toute la liste
Nécessite un flag sur la première valeur pour certaines opérations

Listes chaînées circulaires

Structure

```
1 struct listCirc {
2   struct listCirc * next;
3   int value;
4 };
```

Ajout

```
1 void add(struct listCirc * element, struct listCirc * l) {
2     element->next = l->next;
3     l->next = element;
4 }
```

Listes doublement chaînées

Principe

Chaque maillon est lié au suivant et au précédent
Possibilité de parcours de la liste dans les deux sens
Opérations d'ajout/suppréssion à faire dans les deux sens

Détails

Le dernier élément nécessite d'être traité comme s'il étais également le premier et inversement

Listes doublement chaînées

Listes d'éléments

```
1 typedef struct element_{} element;
2 struct liste_ {
3     element * e;
4     struct liste_ * prev;
5     struct liste_ * next;
6 }
```

Listes chaînées et tableaux

Comparaison

- Listes chaînées : gestion de la mémoire moins efficace mais plus précise
- Listes chaînées : ajout et retrait d'élément simple/possible
- Tableaux : accès plus rapide à un élément i car connexe en mémoire
- Tableaux : meilleures recherches et fonctions de tri

Conclusion

- Les listes chaînées sont des ensembles dynamiques de maillons
- Plus pratiques que les tableaux sur la modification, mais moins sur la lecture
- Attention entre chaîne double ou circulaire