Ali NAJEM et Lucas SOLTIC

Projet “Algorithmique et Programmation”

Résolution d’équations sur les arbres

Document d’analyse

Sommaire

I. Description du projet

Le projet “Résolution d’équations sur les arbres” s’inscrit dans le cadre du projet d’algorithmique et structure de données de 1ère année au département IRM à l’ESIL.

Le but du projet est de réaliser un résolveur de système d’équations basé sur un algorithme donné et s’appuyant sur une structure d’arbre. Les équations seront inscrites en dur dans le programme (éventuellement saisies selon le temps disponible). Le programme affichera le système de départ et les solutions qu’il a trouvées. Si le système est insoluble, le programme l’indiquera.

Le programme sera écrit en C et tirera parti d’une architecture modulaire (voir IV.).

II. Algorithmes

```
construire_systeme()
{
      créer chaque équation
      créer le système à partir des équations
}

traiter_systeme()
{
      tant qu’il reste des équations à traiter
            prendre une équation du système

            // à l’aide de obtenir_representant() :
            obtenir les représentants de l’équation

            // à l’aide de traiter_equation() :
            étudier l’équation

            ranger l’équation après traitement
      fin tantque
}



obtenir_representant(Terme u) (récursif)
{
      u <- terme gauche de e

      si u est une variable alors
```

```
            si S contient une équation e2 de la forme u = v alors
                  obtenir_representant(v);
            sinon
                  le répresentant de u dans S est lui-même.
            finsi
      sinon
            si u est une constante alors
                  le représentant de u dans S est lui-même
            sinon
                  // u est fm(q1,q2,...,qn)
                  le representant de u dans S est
            fm(obtenir_representant (q1), obtenir_representant(q2), ...,
            obtenir_representant(qn))
            fin si
      finsi
}

// après l'exécution de la fonction, dansSys contient l'équation qui
// doit être rajoutée au système, et dansSolu l'équation qui doit
// être rajoutée aux solutions. Si l'un ou les deux paramètres
// contiennent 'rien', il n'y a pas à les considérer
traiter_equation(Equation e, Equation dansSys, Equation dansSolu)
{
      dansSys <- rien
      dansSolu <- rien
      soit s terme gauche de e
      soit t terme droit de e

      si s est une variable et t est une variable alors
            si indice de s = indice de t alors
                  // rien à faire, on supprimera cette équation
            sinon
                  si indice de s < indice de t alors
                        dansSolu <- e // on rangera l'éq. dans S
                  sinon
                        intervertir s et t
                        // on réinsère l'éq inversée dans le système
                        dansSys <- e
                  fin si
            fin si
      sinon si s est une variable et t est une constante alors
            dansSolu <- e // on rangera l'éq. dans S
      sinon si s est une variable et t est une fonction dépendant de S
            S <- insoluble
      sinon
```

```
            dansSolu <- e
      fin si
}


afficher_systeme(Equation *pe)
{
// pe un pointeur qui pointe sur la premiere équation qu'on le passe
par paramètre.
      // tant qu'on a des equation à afficher
      tantque (p!=NULL) faire
                  afficher_equation(p);
            p= p->suivant;
      fin tanque
}

afficher_equation(Terme *p)
{
      si (p->terme_gauche->type_terme==1)
            // c'est le cas où le terme gauche est une variable
            //récuperer l'indice de la variable
            i= p->terme_gauche->contenu_terme.val
            donc le terme est : xi
      sinon
            si (p->terme_gauche->type_terme==2)
                  // c'est le cas où le terme gauche
                  // est une constante
                  on prend la constante elle même
            sinon
                  // Le cas où le terme est de la forme
                  // fk(x1,x2,...,xn)
                  // On récupère l'indice i de la fonction
                  i = p->terme_gauche->type_terme - 30 ;
                  donc le terme est : fi(liste des arguments)
                  //pour afficher les arguments d'une fonction :
                  //on a pa = p->terme_gauche->contenu_terme.arguments
                  //qui pointe aux arguments
                  tantque(pa!=NULL) faire
                        afficher_systeme(pa);
                        pa= pa->suivant;
                  fin tantque
            fin si
      fin si
}
```

```
afficher_solution()
{
      soit S l'ensemble des solutions
      soit p <- S
      soit i un entier = 1

      si S non nul alors
            tant que p non nul faire
                  afficher "Xi =" + en_texte(p->terme_droit.)
                  i = i+1
                  p = p->suivant
            fin tantque
      sinon
            afficher "Système insoluble"
      fin si
}
```

III. Structures de données

Le système d'équations T sera implémenté selon un arbre d'arité variable et avec un nombre de champs variable.

Les équations sont des structures à 3 champs (3 pointeurs) : le premier est un pointeur sur le terme gauche de l'équation, le deuxième vers le terme droite de l'équation et le troisième champ pointe sur la prochaine équation:

```
typedef struct equation_s {
    Terme * terme_gauche;
    Terme * terme_droit;
    struct equation_s * suivant;
} Equation, *Systeme;
```

Un terme est une structure à deux champs. Un champ qui contient un entier égal à 1, 2 ou 30+k (k compris entre 0 et 3) suivant que le terme est une variable, une constante ou une fonction. Et un champ contenant le terme lui-même si le terme est une variable ou une constante ou un pointeur vers une liste d'argument si le terme est complexe (le cas où le terme est une fonction).

```
typedef struct {
```

```
    int type_terme;

    union {
       int val;
       Argument *arguments;
    } contenu_terme;
} Terme;
```

Une liste d'arguments est liste de structure à deux champs: le premier champ contient un pointeur vers un terme et le second champ contient un pointeur vers le prochain argument.

```
typedef struct argument_s {
    Terme *terme_argument;
    argument_s * suivant ;
} Argument;
```

L'ensemble des solutions est représenté par une liste de termes.

```
typedef struct subst_s {
    Terme * terme_droit;
    struct subst_s* suivant;
} *Solutions;
```

De manière schématique, cela donnerait par exemple quelque chose comme:

```
Solutions S;
S->terme_droit = F1(1,2);  /* x1 */
S->suivant->terme_droit = 2; /* x2 */
S->suivant->suivant->terme_droit = X4; /* x3 */
S->suivant->suivant->suivant->terme_droit = NULL; /* x4 */
```

IV. Modules

Modules requis:
- Racine du programme (main(), liaison entre les modules)
- Résolution (récupération du représentant et traitement d'une équation)
- Affichage (affichage du système et de la solution)

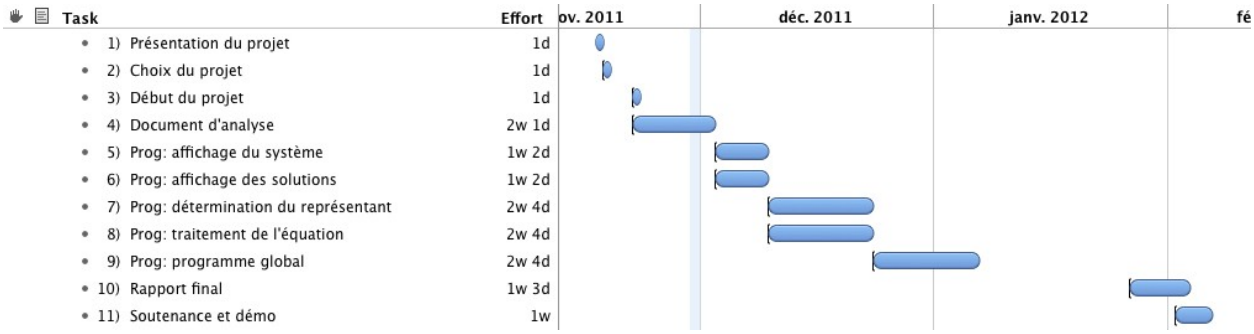
Modules en bonus selon le temps disponible:
- Saisie d'équation en console (de base l'équation est en dur dans le programme)
- Interface graphique

V. Planning

Détail des dates:

17/11/11	présentation du projet
18/11/11	choix
22/11/11	début du projet
02/12/11	rendu du document d'analyse
03/12/2011 - 09/12/2011	affichage du système
03/12/2011 - 09/12/2011	affichage des solution
10/12/2011 - 23/12/2011	détermination du représentant
10/12/2011 - 23/12/2011	traitement de l'équation
24/12/2011 - 06/01/2012	programme global
27/01/2012 - 03/02/2012	rapport final
02/02/2012 - 06/02/2012	soutenance et démonstration

Vue globale du planning:


Task
Effort
ov. 2011
déc. 2011
janv. 2012
fé
1) Présentation du projet
1d
2) Choix du projet
1d
3) Début du projet
1d
4) Document d'analyse
2w 1d
5) Prog: affichage du système
1w 2d
6) Prog: affichage des solutions
1w 2d
7) Prog: détermination du représentant
2w 4d
8) Prog: traitement de l'équation
2w 4d
9) Prog: programme global
2w 4d
10) Rapport final
1w 3d
11) Soutenance et démo
1w