Visualisation des arbres

Dans ce TP, on va s'intéresser à la visualisation d'arbres (binaires).

Dans ce TP et dans tous les TP qui suivront, on pourra mettre à profit les algorithmes et fonctions déjà codés dans les TP précédent. En particulier, on pourra réutiliser le type proposé au TP n°2 pour implémenter les arbres binaires.

Ici, on va utiliser le logiciel de représentation de graphes `dot` pour représenter des arbres. Pour ce faire, il suffit d'indiquer les liens entre les différents nœuds. La description de l'arbre est contenue dans un fichier suffixé par `.dot`. À partir d'un tel fichier, le logiciel `dot` génère un fichier suffixé par `.pdf` et visualisable directement, par exemple avec le logiciel `evince`.

On va donc, ci-dessous, intégrer dans nos programmes de gestion d'arbres des fonctions générant le fichier `.dot` et lancer directement la transformation de ce fichier en un fichier `.pdf` visualisable.

0. **Sortir un papier et un crayon** — Si vous ne réussissez pas cet exercice, votre enseignant ne viendra pas vous aider en cas de problème lors d'une des questions qui suivent.
1. **Étude d'un exemple** — Nous allons tout d'abord étudier l'arbre que nous représenterons en suivant le format ci-dessous :

```
1  digraph arbre {
2    node [shape=record, height=.1]
3    edge [tailclip=false, arrowtail = dot, dir=both];
4
5    n0 [label="<gauche> | <valeur> 6 | <droit>"];
6    n0:gauche:c -> n1:valeur;
7    n1 [label="<gauche> | <valeur> 7 | <droit>"];
8    n0:droit:c -> n2:valeur;
9    n2 [label="<gauche> | <valeur> 3 | <droit>"];
10   n2:droit:c -> n3:valeur;
11   n3 [label="<gauche> | <valeur> 1 | <droit>"];
12 }
```

   Dans cet exemple les noms `n0`, `n1`, `n2` et `n3` sont les noms qui désignent les quatre nœuds de l'arbre. Les trois premières lignes précisent la forme des nœuds et des flèches.

   En ligne 5, le nœud de nom `n0`, qui contient l'entier 6, est déclaré. Ce nœud est formé de trois champs, dont les noms sont `gauche`, `valeur` et `droite`, et sont déclarés entre des chevrons `<` et `>`. Le champ `valeur` contient donc l'entier 6.

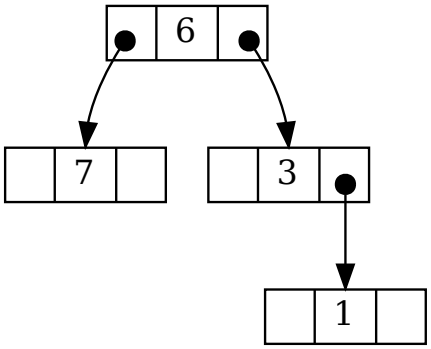
   Puis, en ligne 6, on annonce que le nœud `n0` aura pour fils gauche le nœud `n1`. Le nœud `n1` lui-même n'est déclaré qu'en ligne 7. Il contient l'entier 7, et n'a aucun fils.

   Plus précisément, la ligne 6 relie par une flèche → le nœud `n0` et le nœud `n1`. Les paramères indiqués après chaque symbole "`:`" précisent où placer ce lien. Celui-ci a pour origine le centre (`c`) de la partie `gauche` du nœud `n0`, et pour destination la partie `valeur` du nœud `n1`.

a. Copier le code ci-dessus dans un fichier intitulé `arbre.dot`.

b. Lancer, dans un terminal, la commande `dot -Tpdf arbre.dot -o arbre.pdf` pour créer le fichier `arbre.pdf`, puis lancer la commande `evince arbre.pdf` pour visualiser le fichier `arbre.pdf`.

c. Vérifier que l'arbre ainsi visualisé ressemble bien à



2. **Génération d'un fichier `.dot` par un programme** — On souhaite créer un fichier `nom.dot` en utilisant trois fonctions :

   — une fonction `void ecrireDebut(FILE *f)` qui écrit les trois premières lignes du fichier `*f` ;

   — une fonction `void ecrireArbre(FILE *f, Arbre a)` qui écrit les lignes décrivant l'arbre `a` : dans l'exemple ci-dessus, la fonction générera les lignes 5 à 11 ;

   — une fonciton `void ecrireFin(FILE *f)` qui écrit la dernière ligne du fichier `*f`, c'est-à-dire qu'elle ferme l'accolade.

   Dans la suite, on désigne par `out` un `FILE *` obtenu par l'ouverture en écriture d'un fichier `nom.dot`. On génèrera donc notre code en lançant la fonction `void dessine(FILE *out, Arbre a)` définie par :

```
1 void dessine(FILE *f, Arbre a) {
2   ecrireDebut(out);
3   ecrireArbre(out, a);
4   ecrireFin(out);
5 }
```

   Pour donner un nom unique à chaque nœud, vous pouvez construire des noms en utilisant les adresses mémoire des nœuds. Par exemple, vous pouvez utiliser `fprintf(out,"n%p", tmp)` pour écrire le nom d'un `Noeud *tmp`,. Dans un programme C, la ligne définissant un nœud `a` non `NULL` serait donc, par exemple : `fprintf(out,"n%p␣[label=\"<gauche>␣|␣<valeur>␣%d␣|␣<droit>\"];\n", a,a->valeur)`.
   Voici un exemple d'utilisation :

```
1 FILE *out;
2 /*construire l'arbre a*/
3 /* ... */
4 /*ouvrir le fichier monficher.dot*/
5 out=fopen("monficher.dot","w");
6 /*écrire sur le fichier monfichier.dot*/
7 dessine(out, a);
```

   Il ne reste plus qu'à générer le dessin et à le visualiser.

   a. Écrire le code des fonctions `void ecrireDebut(FILE *f)`, `void ecrireArbre(FILE *f, Arbre a)` et `void ecrireFin(FILE *f)`.

b. Écrire le core de la fonction `void dessine(FILE *f, Arbre a)`, qui écrit l'intégralité des lignes représentant l'arbre `a`, y compris les trois premières et la dernière, dans fichier `*f`.

c. Tester ces fonctions, par exemple en reconstruisant l'arbre représenté en question 1, ou encore les arbres vus au TP n°2.

3. **Lancement par programme et affichage de l'arbre pas à pas**

Le logiciel `evince` met à jour l'affichage dès que le fichier traité est changé. Si, à chaque transformation de l'arbre, votre programme change le fichier `monfichier.dot` et met à jour le fichier `monfichier.pdf`, on peut donc suivre l'évolution de la forme de l'arbre après chaque ajout ou suppression.

La fonction `int system("commande")` de la librairie standard (`stdlb`) permet de lancer l'instruction shell `commande`.

Écrire un programme permettant de visualiser un arbre binaire et de rafraîchir cet affichage après chaque modification de l'arbre. On pourra, par exemple, s'inspirer du canevas suivant :

```
1  void creePDF(char *dot, char *pdf, Arbre a) {
2    FILE *out=fopen(dot,w);
3    dessine(out,a);
4    fclose(out);
5    int len = strlen(dot) + strlen(pdf) + 15;
6    char cmd[len];
7    strcpy(cmd, "dot␣-Tpdf␣");
8    strcat(cmd, dot);
9    strcat(cmd, "␣-o␣");
10   strcat(cmd, pdf);
11   system(cmd);
12 }
13
14 Arbre a=NULL;
15 creePDF("visualise.dot", "visualise.pdf", a);
16 system("evince␣visualise.pdf␣&");
17 /* ouvre evince en background avec un arbre vide */
18
19 /* tant que l'on n'a pas fini nos modifications */
20   /* ajouter un element x à l'arbre a */
21   creePDF("visualise.dot", "visualise.pdf", a);
22   /* faire une pause ou appuyer sur une touche pour l'ajout suivant */
```

On pourra en outre penser à activer (manuellement) dans `evince` l'option **Ajuster à la largeur de la page : 100%** pour que la totalité de l'arbre apparaisse sur l'écran.