Rapport du module
Théorie de l'information

Jonathan Druart
(Responsable : Philippe Langevin)

Année universitaire 2007-2008

Table des matières

1 Introduction

Le but de ce rapport est d'analyser le programme de génération de textes aléatoires implanté en scéances de travaux pratiques. Pour cela, nous approcherons le sujet de manière théorique dans un premier temps en mettant en évidence les notions vues en cours, et de manière pratique en précisant les méthodes d'implantation et en analysant les résultats numériques trouvés.

2 Approche théorique

2.1 Génération de texte aléatoire

La notion d'aléatoire est une notion difficile à appréhender. Dans notre cas, nous allons générer du texte aléatoire, en partant d'un texte pré-existant. Faire l'analyse de ce texte correspond à calculer la fréquence des mots d'une certaine longueur qui le composent. En fait, une fenêtre d'une certaine longueur glisse sur le texte de départ de manière à donner une approximation de la fréquence d'apparition d'une séquence de caractères. Si la fenêtre est de taille 1, nous avons donc la fréquence de chaque caractère dans le texte. Cette méthode nous donne, en plus de la distribution des caractères dans le texte de départ, la probabilité qu'une chaine de caractère soit un mot du texte.

Le texte initial étant stocké dans un arbre, lors de la création du texte aléatoire, le choix du noeud suivant ne dépend que du noeud courant.La prédiction du futur ne dépend donc pas des résultats précédents mais bien uniquement du noeud sur lequel on se trouve dans le présent. Ce processus nous fait penser aux chaînes de Markov.

2.2 Entropie

Dans cette partie, nous allons nous baser sur les travaux de Shannon sur ce sujet. En vulgarisant, Shannon définit l'entropie d'un évènement comme la quantité de surprise $-logp_i$ qu'un observateur aurait quand il découvre que cet

évènement s'est produit. En faisant le moyenne de tous les évènements possibles, on obtient l'entropie du système complet :

$$H_D(p) = -\sum_{i=1}^{n} p_i log_D p_i$$

C'est en 1948 que Claude Shannon introduit cette notion d'entropie, en effet, il cherche à construire un canal de communication le plus efficace possible sans toucher à la technologie de transmission. Il doit donc pour cela trouver le meilleur codage possible. Un canal de transmission étant composé d'une source et d'un récepteur à chaque extrémité de ce canal, le but est de transmettre une quantité d'information dans l'espace de l'un à l'autre. L'entropie correspond à la fonction mathématique de cette quantité d'information que la source contient ou veut émettre. Dans notre cas, la source correspond au fichier de caractères initial. Plus ce fichier contiendra des chaines de caractères redontantes, moins l'information sera intéressante. L'entropie sera maximale lorsque tous les caractères auront la même équiprobabilité. A contrario, l'entropie deviendra nulle lorsque un seul et unique choix sera possible.

3 Approche pratique

3.1 Implantation

Afin d'implanter le programme qui génère aléatoirement du texte, nous avons stocké le texte initial (texte qui sert à la génération aléatoire) dans un arbre binaire. En effet, chaque noeud de l'arbre contient un fils droit et un fils ainé, le premier pointe vers un noeud correspondand à un niveau inférieur, le second à un noeud se situant au même niveau que celui ci. De cette manière, il devient aisé de parcourir l'arbre. La création de l'arbre dépend de la taille de la fenêtre. Cette fenêtre correspond à la taille de la séquence de caractère qui sera utilisée pour générer le texte en sortie. À la lecture du fichier, chaque séquence de caractères de longueur de la fenêtre est lue et insérée dans l'arbre, en faisant glisser la fenêtre jusqu'à la fin de la lecture du fichier. À chaque parcours d'un noeud, le compteur qui lui est associé est incrémenté de 1 afin d'avoir à l'arrivée une pondération des noeuds correspondant au nombre de passage dans chacun. Un fois l'arbre créé, c'est la fonction rand() du langage C qui est utilisée pour faire le choix du noeud. Cette méthode est la limite de ce programme, en effet, la fonction rand() ne génère que des nombres pseudos-aléatoires et non pas réellement aléatoires. En effet, elle est utilisée avec une graine correspond à la date, et il est bien évidemment difficile de générer de l'aléatoire avec une fonction qui ne peut retourner que du pseudo-aléatoire.

3.2 Utilisation du programme

Le code source se compile à l'aide de l'option `-lm` afin de faire les liens avec la librairie `math.h` utilisée. Le programme s'utilise de la manière suivante :

$$./txtaleatoire\ nomFic\ tailleFenetre\ tailleAGenerer$$

avec

- nomFic correspond au nom du fichier qui sera utilisé pour créer l'arbre
- tailleFenetre qui correspond à la taille de la fenêtre
- tailleAGenerer qui correspond à la taille du texte à générer

Le graphe de l'arbre est généré automatique dans le fichier graph.dot du répertoire courant, ce fichier peut vite devenir important. Pour créer l'image png correspondante utiliser la commande :

$$\$dot\ -Tpng\ graph.dot\ -o\ graph.png$$

Un exemple d'un arbre généré de cette manière est disponible en `annexe B`.

Le programme retourne également l'entropie des mots correspondant à chaque feuille de l'arbre (voir section suivante) pour la fenêtre donnée.

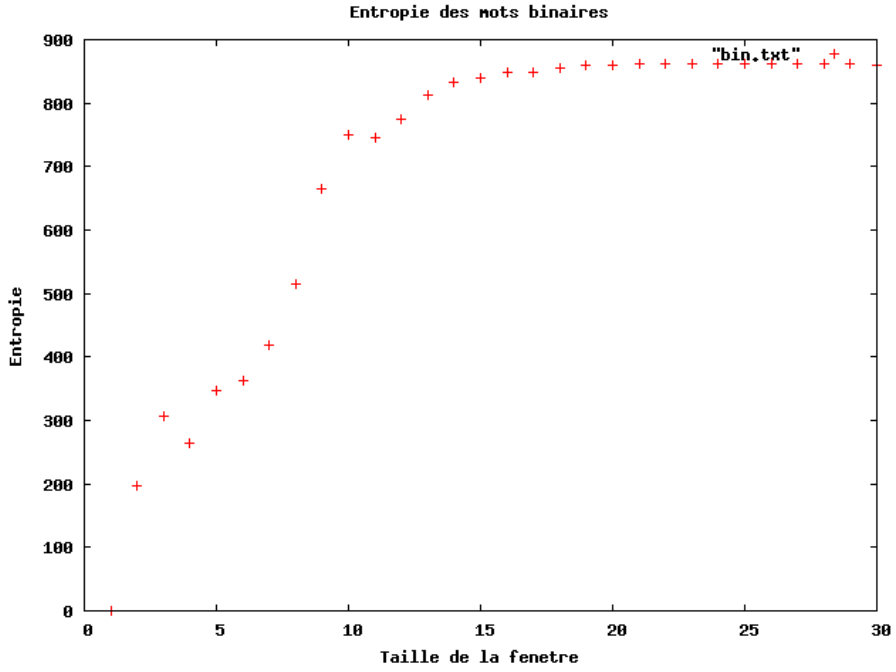
3.3 Analyse des résultats

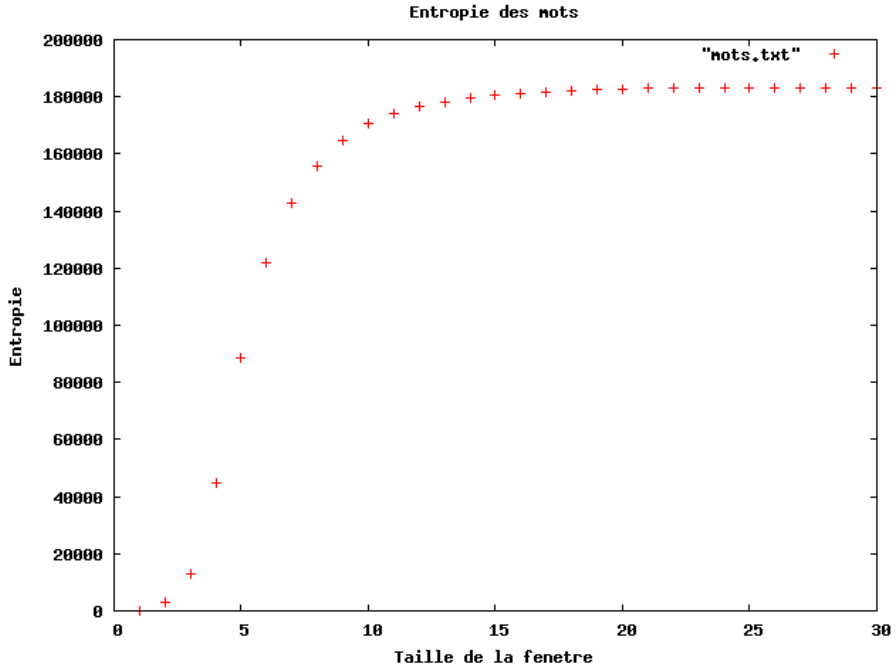
Les résultats directs de ce programme sont satisfaisants à partir d'une fenêtre de taille variant de 8 à 10 pour un texte en français. En effet, la syntaxe de la langue du texte en sortie devient correcte, tandis que le sens de ne l'est pas. Pour un texte en anglais, la taille de cette fenêtre descend à 7, 8.

Pour approfondir la notion d'entropie, il m'a semblé intéressant de calculer en fonction de la taille de la fenêtre, l'entropie des mots formés par l'arbre. Pour chaque feuille de l'arbre, j'ai donc récupéré le mot qui lui correspondait et calculé la somme de ses poids. En sommant encore une fois tous ces résultats et en multipliant par $log_2(alpha)$, on trouve l'entropie, soit la probabilité de prédire quel mot sera trouvé. Alpha correspondant au cardinal de l'alphabet utilisé.

3.3.1 Résultats trouvés

Les tests ont été effectués sur deux fichiers. L'un n'était composé que de l'alphabet binaire, l'autre est tiré d'un article sur l'informatique et contenait donc le plupart des caractères alpha-numériques utilisés dans la langue française (5000 mots).





On remarque que ces deux courbes sont logarithmiques et que le palier arrive pour une fenêtre de 15 pour l'alphabet binaire et de 10 pour l'alphabet alphanumérique.

Il y a une présence de discontinuité dans la courbe correspondant au texte binaire, pour une fenêtre égale à 3, 5 et 10 caractères, cela est dû au fait que le texte utilisé pour ce test n'était certainement pas assez important.

On constate également que plus la fenêtre est grande et moins on peut prédire avec certitude le mot qui peut être généré. Contrairement, plus la taille de la fenêtre est petite et plus l'entropie est faible et donc plus le mot généré est prédictible. Donc plus la taille de la fenêtre est grande et plus le phénomène aléatoire est présent. Au bout d'un certain point, (pour une largeur de fenêtre égale à 15 pour l'alphabet binaire, sinon 10) l'entropie est maximale, on ne peut plus prédire le mot, ce qui est normal étant donné qu'il y a plus d'incertitude car la présence d'informations est plus élevée. En atteignant cette valeur maximale, on a attend le phénomène le plus aléatoire possible, on ne pourra pas avoir d'entropie plus importante. De la même manière, on peut penser que l'espace mémoire de l'arbre généré atteindra également un maximum et que la courbe le représentant aura le même comportement.

4 Conclusion

La génération de texte aléatoire est une méthode très intéressante pour comprendre et apprendre les phénomènes liés à l'entropie. En effet, nous avons pu voir les relations entre les résultats découlants de cette génération de texte aléatoire et la notion d'entropie. Les résultats trouvés sur l'entropie des mots qui sont générés nous permettent de connaître la probabilité qu'un mot ressorte plus que tel ou tel autre. La taille de la fenêtre que l'on fait glisser à un impact important sur cette entropie, effectivement, à partir d'un certain seuil la prédiction est beaucoup moins grande qu'avec une fenêtre toute petite (de 1 ou 2 caractère(s)). On peut par ailleurs se demander quel phénomène intervient pour que cette valeur stagne à partir d'un certain temps et que l'on ne puisse pas obtenir une entropie encore plus forte et donc obtenir des résultats plus aléatoires. Peut-être est-ce dû à la fonction rand() du langage C qui a été utilisé et qui est basée sur le temps qui ne correspond pas à un phénomène entropique très important. Il aurait été intéressant de tester le même programme avec une fonction générant des nombres aléatoires plus efficacement.

A Code source

Listing 1 –

```
1
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <string.h>
6 #include <math.h>
#define log2(x) (log(x)*1.44269504088896340735992468100 19)

typedef struct Arbre{
 char car;
11 int cpt;
 struct Arbre *fd, *fa; //fils droit, fils ain
}Arbre;

int id=0; // Variable globale pour la cration du graphe
16

/*********************************************************************
 * Retourne le squence decaractre se situant la position donne *
 *********************************************************************/
21
char* CarPosition(FILE* fic, int position, int longueur){
 char* tabChar = (char*)malloc(longueur * sizeof(char));
 int i = 0;

26 for (i = 0 ; i < longueur ; i++){
  fseek(fic, position + i, SEEK_SET);
 if (fscanf(fic, "%c", &tabChar[i]) == EOF)
  return NULL;
 }
31 return tabChar;
}


/*********************************************************************
36 * Ajoute une chaine une chaine de caractre l'arbre *
 *********************************************************************/

Arbre* AjouterChaine(Arbre* arbre, char* tabChar, int taille){
 if (arbre == NULL){
41  Arbre* racine = (Arbre*)malloc(sizeof(Arbre));
  racine->fd = NULL;
  racine->fa = NULL;
  racine->cpt = 1;
  racine->car = tabChar[0];
46  taille --;
```

```
    arbre=racine;
    if ( taille > 0)
     racine->fa = AjouterChaine(arbre->fa, tabChar + sizeof(char), taille);
    return racine;
  }else{
    Arbre* marqueur = arbre;

    while(1){
     if (marqueur->car == tabChar[0]){
      marqueur->cpt ++;
       taille --;
      if ( taille > 0)
       marqueur->fa = AjouterChaine(marqueur->fa, tabChar + sizeof(char), taille);
       return arbre;
     }else{
      if (marqueur->fd == NULL)
       break;
      else
       marqueur = marqueur->fd;
     }
    }

    Arbre* noeud = (Arbre*)malloc(sizeof(Arbre));
    noeud->fd = NULL;
    noeud->fa = NULL;
    noeud->cpt = 1;
    noeud->car = tabChar[0];

    marqueur->fd = noeud;

     taille --;

    if ( taille > 0)
     marqueur->fd->fa = AjouterChaine(marqueur->fd->fa, tabChar + sizeof(char), taille);
    return arbre;
   }
  }


 /*****************************************************************
  * Initialisation de l'arbre *
  *****************************************************************/

  Arbre* InitArbre(char* nomFic, int tailleFenetre){
   FILE* fic;
   char* tabChar;
   Arbre* racine = NULL;
   int pos = 0;
   fic = fopen(nomFic, "r");
   if ( fic != NULL){
```

```
  while((tabChar = CarPosition(fic, pos, tailleFenetre )) != NULL){
   racine = AjouterChaine(racine, tabChar, tailleFenetre);
   pos = pos + 1;
  }
   fclose ( fic );
  return racine;
 }else{
  printf("Le␣fichier␣%s␣n'a␣pas␣ t ␣ouvert,␣pb␣de␣droit␣ou␣n'existe␣pas\n",nomFic);
  return NULL;
 }
 free(tabChar);
}


/*********************************************************************
 * Retourne le caractre se situant la position donne *
 *********************************************************************/

char CarCorrespondant(Arbre* arbre, int pos){
 if (pos < arbre->cpt)
  return (arbre->car);
 else{
  pos = pos - arbre->cpt;
  return (CarCorrespondant(arbre->fd, pos));
 }
}


/*********************************************************************
 * Retourne le caractre suivant *
 *********************************************************************/

char CarSuivant(char* tabChar, int tailleFenetre, Arbre* arbre){
 Arbre* arbreRec = arbre;

 if ( tailleFenetre == 0){
  int somme = 0;
  do{
   somme = somme + arbreRec->cpt;
   arbreRec = arbreRec->fd;
  }while(arbreRec != NULL);
   int rand = random() % somme;
   return CarCorrespondant(arbre, rand);
 }else{
 do{
  if (tabChar[0] == arbreRec->car)
   return CarSuivant(&tabChar[1], tailleFenetre - 1, arbreRec->fa);
  else arbreRec = arbreRec->fd;

  }while(arbreRec != NULL);
```

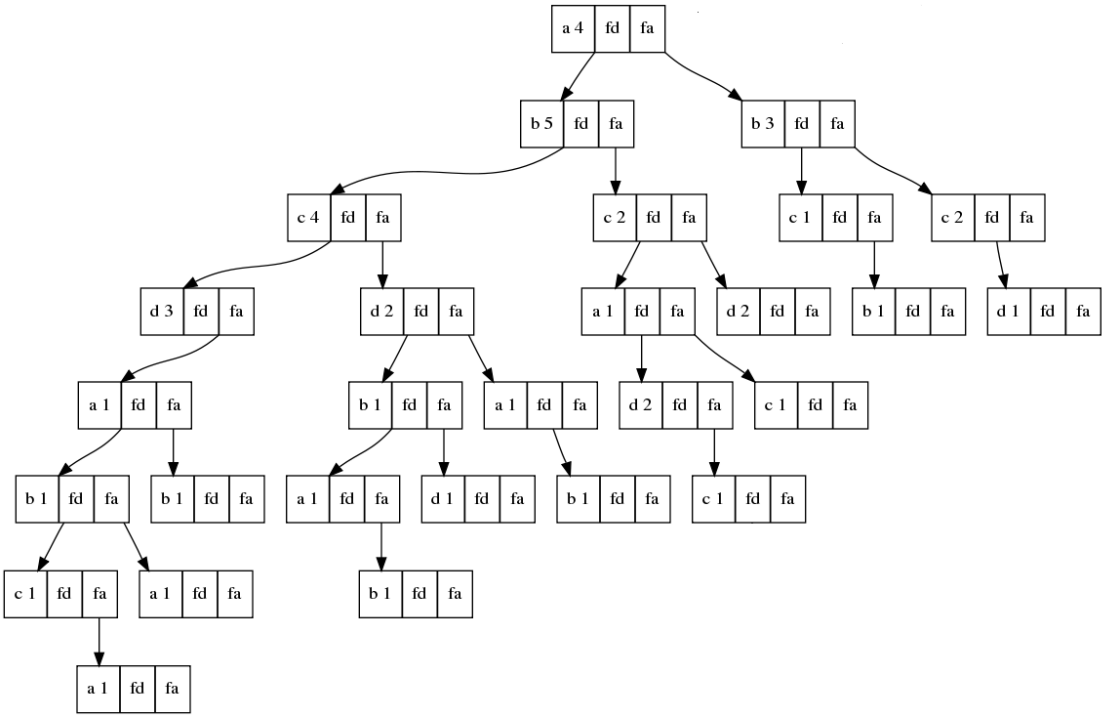
```
 }
 return '\0';
}

/*****************************************************************
 * Dcale la fenetre d'une position *
 *****************************************************************/

void DecalerFenetre(char* tabChar, int tailleFenetre){
 memmove(tabChar, &(tabChar[1]), tailleFenetre - 1);
 tabChar[tailleFenetre - 1] = '\0';
}

/*****************************************************************
 * Gnre le texte alatoire *
 *****************************************************************/

char* CreationTxtAleatoire(Arbre* arbre, int tailleFenetre, int tailleTxtAGenerer){
 char* tabChar = (char*)malloc(sizeof(char)*(tailleFenetre+1));
 char* txt = (char*)malloc(sizeof(char)*(tailleTxtAGenerer));
 int i = 0;
 for (i = 0 ; i < tailleFenetre ; i++){
  tabChar[i] = CarSuivant(tabChar, i, arbre);
 }
 tabChar[i] = '\0';
 strcpy(txt, tabChar);
 for (; i < tailleTxtAGenerer ; i++){
  DecalerFenetre(tabChar, tailleFenetre);
  tabChar[tailleFenetre-1] = CarSuivant(tabChar, tailleFenetre - 1, arbre);
  if (tabChar[tailleFenetre - 1] == '\0'){
   printf("Chaine de caractres introuvable\n");
   break;
  }
  txt[i] = tabChar[tailleFenetre - 1];
 }
 txt[i] = '\0';
 free(tabChar);
 return (txt);
}


/*****************************************************************
 * Somme les compteurs de chaque noeuds *
 *****************************************************************/

int SommeCpt(Arbre* arbre){
 if(arbre != NULL){
  if ((arbre->fd != NULL) && (arbre->fa != NULL)){
```

```
    return (arbre->cpt + SommeCpt(arbre->fa) + SommeCpt(arbre->fd));
   }else{
    if(arbre->fa != NULL){
     return (arbre->cpt + SommeCpt(arbre->fa));
    }else if(arbre->fd != NULL){
     return (arbre->cpt + SommeCpt(arbre->fd));
    }else{
     return arbre->cpt;
    }
   }
  }
}


/*********************************************************************
 * Trouve le mot correspondant une feuille donne *
 *********************************************************************/

int TrouverMot(Arbre* feuille){
 if ( feuille ->fa == NULL) return feuille->cpt;
 else return (feuille->cpt + TrouverMot(feuille->fa));
}

/*********************************************************************
 * Calcule l'entropie des mots *
 *********************************************************************/

int EntropieMots(Arbre* arbre){
 if(arbre != NULL){
  if ((arbre->fd != NULL) && (arbre->fa != NULL)){
   return (EntropieMots(arbre->fa) + EntropieMots(arbre->fd));
  }else{
   if(arbre->fa != NULL) return (EntropieMots(arbre->fa));
   else if(arbre->fd != NULL) return (EntropieMots(arbre->fd));
   else return TrouverMot(arbre);
  }
 }
}


/*********************************************************************
 * Libration de l'espace mmoire allou pour l'arbre *
 *********************************************************************/

void FreeArbre(Arbre* arbre){
 if (arbre != NULL){
  FreeArbre(arbre->fd);
  FreeArbre(arbre->fa);
  free(arbre);
 }
```

```
}


/*****************************************************************
 * Cree le fichier du graphe *
 *****************************************************************/

void CreerGraphe(Arbre* a, FILE* graph){
 if(a){
  fprintf(graph,"\"%p\" [label = \"<f0> %c %d|<f1> fd|<f2> fa\" shape = \"record\"];\n", a ,a->car, a
  if(a->fd){
   fprintf(graph,"\"%p\":f1 -> \"%p\":f0 [id = %d];\n", a, a->fd, id);
   id ++;
  }
  if(a->fa){
   fprintf(graph,"\"%p\":f2 -> \"%p\":f0 [id = %d];\n", a, a->fa, id);
   id ++;
  }
  CreerGraphe(a->fd, graph);
  CreerGraphe(a->fa, graph);
 }
}


/*****************************************************************
 * Fonction principale *
 *****************************************************************/

int main(int argc, char* argv[]){

 char* nomFic;
 int tailleFenetre = 0;
 int tailleTxtAGenerer = 0;
 FILE* graph;

 if (argc != 4){
  printf("Utilisation : \n\t./txtaleatoire nomFic tailleFenetre tailleTxtAGenerer\n");
  return -1;
 }else{
  nomFic = argv[1];
  tailleFenetre = atoi(argv[2]);
  tailleTxtAGenerer = atoi(argv[3]);
  srand(time(NULL));
  printf("--- Initialisation de L'arbre ---\n");
  Arbre* arbre = InitArbre(nomFic, tailleFenetre);
  printf("--- Arbre initialis ---\n");
  if (arbre != NULL){
   char* txt;
   printf("--- L'arbre a t construit avec succs ---\n");

```

```
    printf("--- Cration du fichier graph.dot ---\n");
    graph = fopen("graph.dot","w");
    if (graph != NULL){
     fprintf(graph,"digraph g {\n");
301  CreerGraphe(arbre, graph);
     printf("--- Le fichier graph.dot  t   gnr   avec succs ---\n");
    }else printf("--- Le fichier graph.dot n' pas  t   gnr   , ( problme de droits ?) ---\n");

     printf("--- Cration du texte alatoire ---\n");
306  txt = CreationTxtAleatoire(arbre, tailleFenetre, tailleTxtAGenerer);
     if (txt != NULL){
      printf("--- Le texte a t   gnr   avec succs ---\n");
      printf("--------------------------------------------------\n");
      printf("--------------------TEXTE GNR--------------------\n");
311   printf("%s\n",txt);
      printf("--------------------------------------------------\n");
      int entropieMots = EntropieMots(arbre);
      float rlog = log2(45);
      printf("entropie des mots = %d\n -> *log2(45)=%f\n",entropieMots, rlog*entropieMots);
316   printf(" Utiliser la commande $dot -Tpng graph.dot -o graph.png pour gnrer le png correspo
     }else{
      printf("--- ERREUR ! Le texte n'a pas pu tre gnr ---\n");
     }
     fprintf(graph,"}\n");
321  fclose(graph);

    }else{
     printf("L'arbre n'a pu tre construit , ou est vide\n");
     exit(0);
326 }
    return 0;
  }
}
```

B Exemple de graphe



C Exemple d'exécution

./txtaleatoire InputFile 10 350
— Initialisation de L'arbre —
— Arbre initialisé —
— L'arbre a été construit avec succès —
— Création du fichier graph.dot —
— Le fichier graph.dot à été généré avec succès —
— Création du texte aléatoire —
— Le texte a été généré avec succès —
————————————————————————-
————————-TEXTE GÉNÉRÉ————————
lus en plus pressant de village armé de courts bâtons, d'arcs et de la communication et de l'utilisation des composants et la terminologie ont suivi l'évolution en gestion de la restaurants dits de spécialité [modifier]

Voir aussi consumérisme). Si certains cours dans les universités américaine Informatics (science du calculateur . En fr
————————————————————————-
total des poids = 334150
-> *log2(45)=1835102.759194
Utiliser la commande dot -Tpng graph.dot -o graph.png pour générer le png