L3 SID APU
Cours 4
Méthodes de conception d'algorithmes
Algorithmes gloutons

Thomas Pellegrini, équipe SAMoVA, IRIT,
thomas.pellegrini@irit.fr

IRIT - UPS

2016-2017

Note

Ce cours est (largement) inspiré des sources suivantes :

- Cormen, Leiserson, Rivest, Stein. Algorithmique, Dunod, 3e édition
- Slides d'Ana Busic, `http://www.di.ens.fr/~busic/cours/LI325/slidesCAAC8_1213.pdf`
- `https://openclassrooms.com/`

IRIT CNRS - INPT - UPS - UT1 - UT2J

UNIVERSITÉ TOULOUSE III PAUL SABATIER Université de Toulouse

Exemples de problèmes courants

Rendu de monnaie

- Un caissier doit vous rendre la monnaie sur un achat, par exemple 2,63 EUR
- Beaucoup de possibilités s'offrent à lui : 263 pièces de 1 cent, ou 4 pièces de 50 cents, 2 pièces de 20 cents, 2 pièces de 10 cents, 1 pièce de 2 cents et une pièce de 1 cent
- Problème : minimiser le nombre de pièces rendues pour un montant fixé
- Solution naïve (force brute) : énumérer toutes les possibilités et choisir celle qui utilise le minimum de pièces $\rightarrow$ pas très efficace...

Exemples de problèmes courants

Rendu de monnaie

- Un caissier doit vous rendre la monnaie sur un achat, par exemple 2,63 EUR
- Beaucoup de possibilités s'offrent à lui : 263 pièces de 1 cent, ou 4 pièces de 50 cents, 2 pièces de 20 cents, 2 pièces de 10 cents, 1 pièce de 2 cents et une pièce de 1 cent
- Problème : minimiser le nombre de pièces rendues pour un montant fixé
- Solution naïve (force brute) : énumérer toutes les possibilités et choisir celle qui utilise le minimum de pièces $\rightarrow$ pas très efficace...

IRIT CNRS - INPT - UPS - UT1 - UT2J

UNIVERSITÉ TOULOUSE III PAUL SABATIER Université de Toulouse

Exemples de problèmes courants

Rendu de monnaie

- Un caissier doit vous rendre la monnaie sur un achat, par exemple 2,63 EUR
- Beaucoup de possibilités s'offrent à lui : 263 pièces de 1 cent, ou 4 pièces de 50 cents, 2 pièces de 20 cents, 2 pièces de 10 cents, 1 pièce de 2 cents et une pièce de 1 cent
- Problème : minimiser le nombre de pièces rendues pour un montant fixé
- Solution naïve (force brute) : énumérer toutes les possibilités et choisir celle qui utilise le minimum de pièces $\rightarrow$ pas très efficace...

IRIT CNRS - INPT - UPS - UT1 - UT2J

UNIVERSITÉ TOULOUSE III PAUL SABATIER

Exemples de problèmes courants

Rendu de monnaie

- Un caissier doit vous rendre la monnaie sur un achat, par exemple 2,63 EUR
- Beaucoup de possibilités s'offrent à lui : 263 pièces de 1 cent, ou 4 pièces de 50 cents, 2 pièces de 20 cents, 2 pièces de 10 cents, 1 pièce de 2 cents et une pièce de 1 cent
- Problème : minimiser le nombre de pièces rendues pour un montant fixé
- Solution naïve (force brute) : énumérer toutes les possibilités et choisir celle qui utilise le minimum de pièces $\rightarrow$ pas très efficace...

Exemples de problèmes courants

The knapsack problem ou problème du sac à dos fractionnaire

- On dispose d'un ensemble S de n objets qui possèdent chacun une valeur v_i et un poids w_i
- Nous sommes obligés de nous restreindre à emporter seulement une partie T de ces objets ou des fractions de ces objets car notre sac à dos a une capacité limitée W
- Problème : maximiser la somme des valeurs des objets qu'on va emporter avec soi

Exemples de problèmes courants

The knapsack problem ou problème du sac à dos fractionnaire

- On dispose d'un ensemble S de n objets qui possèdent chacun une valeur v_i et un poids w_i
- Nous sommes obligés de nous restreindre à emporter seulement une partie T de ces objets ou des fractions de ces objets car notre sac à dos a une capacité limitée W
- Problème : maximiser la somme des valeurs des objets qu'on va emporter avec soi

IRIT CNRS - INPT - UPS - UT1 - UT2J

UNIVERSITÉ TOULOUSE III PAUL SABATIER Université de Toulouse

Exemples de problèmes courants

The knapsack problem ou problème du sac à dos fractionnaire

- On dispose d'un ensemble S de n objets qui possèdent chacun une valeur v_i et un poids w_i
- Nous sommes obligés de nous restreindre à emporter seulement une partie T de ces objets ou des fractions de ces objets car notre sac à dos a une capacité limitée W
- Problème : maximiser la somme des valeurs des objets qu'on va emporter avec soi

Exemples de problèmes courants

The knapsack problem ou problème du sac à dos fractionnaire

- On dispose d'un ensemble S de n objets qui possèdent chacun une valeur v_i et un poids w_i
- Nous sommes obligés de nous restreindre à emporter seulement une partie T de ces objets ou des fractions de ces objets car notre sac à dos a une capacité limitée W
- Problème : maximiser la somme des valeurs des objets qu'on va emporter avec soi

$$\max_{T \subseteq S} \sum_{i \in T} v_i$$

$$\text{avec} \sum_{i \in T} w_i \leq W$$

Exemples de problèmes courants

The knapsack problem ou problème du sac à dos

- Problème : maximiser la somme des valeurs des objets ou des fractions d'objets qu'on va emporter avec soi

$$\max_{T \subseteq S} \sum_{i \in T} v_i$$

$$\text{avec} \sum_{i \in T} w_i \leq W$$

- Solution naïve (force brute) : trouver toutes les combinaisons d'objets possibles qui satisfont à la capacité maximale du sac ou qui s'en rapprochent $\rightarrow$ pas très efficace...

IRIT
CNRS - INPT - UPS - UT1 - UT2J

UNIVERSITÉ TOULOUSE III PAUL SABATIER

Exemples de problèmes courants

Rendu de monnaie

- Solution gloutonne :
- On va répéter le choix de la pièce de plus grande valeur qui ne dépasse pas la somme restante
- Remarque : l'algo fonctionne correctement avec le système monétaire européen, mais qu'en est-il si l'on ne dispose que de pièces de 2, 3 et 4 euros seulement et que la somme à recevoir est de 9 euros ?

Exemples de problèmes courants

Rendu de monnaie

- Solution gloutonne :
- On va répéter le choix de la pièce de plus grande valeur qui ne dépasse pas la somme restante
- Remarque : l'algo fonctionne correctement avec le système monétaire européen, mais qu'en est-il si l'on ne dispose que de pièces de 2, 3 et 4 euros seulement et que la somme à recevoir est de 9 euros ?

CNRS - INPT - UPS - UT1 - UT2J

UNIVERSITÉ TOULOUSE III PAUL SABATIER

Exemples de problèmes courants

Rendu de monnaie

- Solution gloutonne :
- On va répéter le choix de la pièce de plus grande valeur qui ne dépasse pas la somme restante
- Remarque : l'algo fonctionne correctement avec le système monétaire européen, mais qu'en est-il si l'on ne dispose que de pièces de 2, 3 et 4 euros seulement et que la somme à recevoir est de 9 euros ?

Exemples de problèmes courants

The knapsack problem ou problème du sac à dos

- Solution gloutonne :
- On va répéter l'ajout de l'objet ou d'une fraction de l'objet qui a le plus grand ratio valeur/poids, avec un poids qui ne fait pas dépasser la contenance du sac, jusqu'à saturation du sac
- Pour faire ce choix, on a besoin de trier les objets par leur valeur par kilogramme $\rightarrow$ cet algo glouton s'exécute en $O(n \log n)$
- Remarque : l'algo ne fonctionne pas correctement si l'on ne permet pas de pouvoir prendre une fraction des objets mais uniquement les objets entiers (problème du sac à dos variante 0-1). Illustrez pourquoi avec trois objets : (10 kg, 60 EUR), (20 kg, 100 EUR) et (30 kg, 120 EUR), avec un sac à dos pouvant contenir 50 kg maximum. Quelle sont les solutions optimales dans les deux variantes du problème et que trouve l'algo dans chaque cas ?

IRIT CNRS - INPT - UPS - UT1 - UT2J

UNIVERSITÉ TOULOUSE III PAUL SABATIER

Exemples de problèmes courants

The knapsack problem ou problème du sac à dos

- Solution gloutonne :
- On va répéter l'ajout de l'objet ou d'une fraction de l'objet qui a le plus grand ratio valeur/poids, avec un poids qui ne fait pas dépasser la contenance du sac, jusqu'à saturation du sac
- Pour faire ce choix, on a besoin de trier les objets par leur valeur par kilogramme $\rightarrow$ cet algo glouton s'exécute en $O(n \log n)$
- Remarque : l'algo ne fonctionne pas correctement si l'on ne permet pas de pouvoir prendre une fraction des objets mais uniquement les objets entiers (problème du sac à dos variante 0-1). Illustrez pourquoi avec trois objets : (10 kg, 60 EUR), (20 kg, 100 EUR) et (30 kg, 120 EUR), avec un sac à dos pouvant contenir 50 kg maximum. Quelle sont les solutions optimales dans les deux variantes du problème et que trouve l'algo dans chaque cas ?

Exemples de problèmes courants

The knapsack problem ou problème du sac à dos

- Solution gloutonne :
- On va répéter l'ajout de l'objet ou d'une fraction de l'objet qui a le plus grand ratio valeur/poids, avec un poids qui ne fait pas dépasser la contenance du sac, jusqu'à saturation du sac
- Pour faire ce choix, on a besoin de trier les objets par leur valeur par kilogramme $\rightarrow$ cet algo glouton s'exécute en $O(n \log n)$
- Remarque : l'algo ne fonctionne pas correctement si l'on ne permet pas de pouvoir prendre une fraction des objets mais uniquement les objets entiers (problème du sac à dos variante 0-1). Illustrez pourquoi avec trois objets : (10 kg, 60 EUR), (20 kg, 100 EUR) et (30 kg, 120 EUR), avec un sac à dos pouvant contenir 50 kg maximum. Quelle sont les solutions optimales dans les deux variantes du problème et que trouve l'algo dans chaque cas ?

IRIT
CNRS - INPT - UPS - UT1 - UT2J

UNIVERSITÉ TOULOUSE III PAUL SABATIER

Exemples de problèmes courants

The knapsack problem ou problème du sac à dos

- Solution gloutonne :
- On va répéter l'ajout de l'objet ou d'une fraction de l'objet qui a le plus grand ratio valeur/poids, avec un poids qui ne fait pas dépasser la contenance du sac, jusqu'à saturation du sac
- Pour faire ce choix, on a besoin de trier les objets par leur valeur par kilogramme $\rightarrow$ cet algo glouton s'exécute en $O(n \log n)$
- Remarque : l'algo ne fonctionne pas correctement si l'on ne permet pas de pouvoir prendre une fraction des objets mais uniquement les objets entiers (problème du sac à dos variante 0-1). Illustrez pourquoi avec trois objets : (10 kg, 60 EUR), (20 kg, 100 EUR) et (30 kg, 120 EUR), avec un sac à dos pouvant contenir 50 kg maximum. Quelle sont les solutions optimales dans les deux variantes du problème et que trouve l'algo dans chaque cas ?

Principe général

- Technique pour résoudre des problèmes d'optimisation

Principe général

- Technique pour résoudre des problèmes d'optimisation
- Solution optimale obtenue en effectuant une suite de choix : à chaque étape on fait le choix localement optimal

Principe général

- Technique pour résoudre des problèmes d'optimisation
- Solution optimale obtenue en effectuant une suite de choix : à chaque étape on fait le choix localement optimal
- Pas de retour en arrière : à chaque étape de décision dans l'algorithme, le choix qui semble le meilleur à ce moment est effectué

CNRS - INPT - UPS - UT1 - UT2J

UNIVERSITÉ TOULOUSE III PAUL SABATIER

Principe général

- Technique pour résoudre des problèmes d'optimisation
- Solution optimale obtenue en effectuant une suite de choix : à chaque étape on fait le choix localement optimal
- Pas de retour en arrière : à chaque étape de décision dans l'algorithme, le choix qui semble le meilleur à ce moment est effectué
- On espère que ce choix mènera à une solution globalement optimale

Principe général

- Technique pour résoudre des problèmes d'optimisation
- Solution optimale obtenue en effectuant une suite de choix : à chaque étape on fait le choix localement optimal
- Pas de retour en arrière : à chaque étape de décision dans l'algorithme, le choix qui semble le meilleur à ce moment est effectué
- On espère que ce choix mènera à une solution globalement optimale
- Progression descendante : choix puis résolution d'un problème plus petit

Principe général

- Technique pour résoudre des problèmes d'optimisation
- Solution optimale obtenue en effectuant une suite de choix : à chaque étape on fait le choix localement optimal
- Pas de retour en arrière : à chaque étape de décision dans l'algorithme, le choix qui semble le meilleur à ce moment est effectué
- On espère que ce choix mènera à une solution globalement optimale
- Progression descendante : choix puis résolution d'un problème plus petit
- Les algos gloutons sont très puissants et fonctionnent bien pour toutes sortes de problèmes

Plan

Introduction

- Principe général
- Exemple : location de camion

Codage de Huffman

IRIT CNRS · INPT · UPS · UT1 · UT2J

UNIVERSITÉ TOULOUSE III PAUL SABATIER

Exemple : Réservation d'une salle de cours

- Une salle de cours à partager entre plusieurs enseignants et maximiser le nombre d'enseignants satisfaits
- La secrétaire reçoit un ensemble de n demandes de réservation de la salle : $S = \{a_1, a_2, \ldots, a_n\}$
- Chaque cours a_i a une heure de début d_i et un heure de fin f_i connues
- Deux demandes a_i et a_j sont compatibles si elles ne se chevauchent pas : $d_i > f_j$ ou $d_j > f_i$
- Problème : trouver un sous-ensemble de taille maximale de cours mutuellement compatibles

IRIT CNRS - INPT - UPS - UT1 - UT2J

UNIVERSITÉ TOULOUSE III PAUL SABATIER

Algorithme glouton

Exemple

i	1	2	3	4	5	6	7	8	9
d_i	1	2	4	1	5	8	9	13	11
f_i	8	5	7	3	9	11	10	16	14

Algorithme glouton : trier selon critère et satisfaire les demandes par ordre croissant
Quel critère ?

Algorithme glouton

Exemple

i	1	2	3	4	5	6	7	8	9
d_i	1	2	4	1	5	8	9	13	11
f_i	8	5	7	3	9	11	10	16	14

Algorithme glouton : trier selon critère et satisfaire les demandes par ordre croissant
Quel critère ?

- Durée ?
- Heure de début ?
- Heure de fin ?

IRIT CNRS - INPT - UPS - UT1 - UT2J

UNIVERSITÉ TOULOUSE III PAUL SABATIER

Algorithme glouton

Exemple

i	1	2	3	4	5	6	7	8	9
d_i	1	2	4	1	5	8	9	13	11
f_i	8	5	7	3	9	11	10	16	14

Algorithme glouton : trier selon critère et satisfaire les demandes par ordre croissant
Quel critère ?

- Durée ? NON
- Heure de début ? NON
- Heure de fin ? OUI

Algorithme glouton

Demandes

i	1	2	3	4	5	6	7	8	9
d_i	1	2	4	1	5	8	9	13	11
f_i	8	5	7	3	9	11	10	16	14

Algorithme glouton

Demandes

i	1	2	3	4	5	6	7	8	9
d_i	1	2	4	1	5	8	9	13	11
f_i	8	5	7	3	9	11	10	16	14

Etape 1 : tri selon la date de fin

i	4	2	3	1	5	7	6	9	8
d_i	1	2	4	1	5	9	8	11	13
f_i	3	5	7	8	9	10	11	14	16

Algorithme glouton

Demandes

i	1	2	3	4	5	6	7	8	9
d_i	1	2	4	1	5	8	9	13	11
f_i	8	5	7	3	9	11	10	16	14

Etape 1 : tri selon la date de fin

i	4	2	3	1	5	7	6	9	8
d_i	1	2	4	1	5	9	8	11	13
f_i	3	5	7	8	9	10	11	14	16

Etape 2 : satisfaire les demandes par ordre croissant

i	4	2	3	1	5	7	6	9	8
d_i	1	2	4	1	5	9	8	11	13
f_i	3	5	7	8	9	10	11	14	16

Solution : cours numéros 4, 3, 7 et 9

IRIT
CNRS - INPT - UPS - UT1 - UT2J

UNIVERSITÉ TOULOUSE III PAUL SABATIER

Algorithme glouton : version récursive

Entrées : tableaux *d* et *f* triés convention : $f[0] \leftarrow 0$

1: FONCTION R(d, f, i, n)
2: m $\leftarrow$ i + 1
3: TANT QUE $m \leq n$ ET $d[m] < f[i]$ FAIRE
4: m $\leftarrow$ m + 1
5: SI $m \leq n$ ALORS
6: RETOURNER $\{m\} \cup R(d, f, m, n)$
7: SINON
8: RETOURNER $\emptyset$

Appel : R(d,f,0,n)

IRIT CNRS - INPT - UPS - UT1 - UT2J

UNIVERSITÉ TOULOUSE III PAUL SABATIER Université de Toulouse

Algorithme glouton

Entrées : tableaux d et f triés

version récursive
convention : $f[0] \leftarrow 0$

1: FONCTION R(d, f, i, n)
2: m ← i + 1
3: TANT QUE $m \leq n$ ET $d[m] < f[i]$ FAIRE
4:     m ← m + 1
5: SI $m \leq n$ ALORS
6:     RETOURNER $\{m\} \cup R(d, f, m, n)$
7: SINON
8:     RETOURNER $\emptyset$

Appel : R(d,f,0,n)

version itérative

1: FONCTION glouton(d, f, n)
2: A ← {1}
3: k ← 1
4: POUR i DE 2 à n FAIRE
5:     SI $d[i] \geq f[k]$ ALORS
6:         $A \leftarrow A \cup \{i\}$
7:         $k \leftarrow i$
8: RETOURNER A

Appel : glouton(d,f,n)

Algorithme glouton

Entrées : tableaux d et f triés

version récursive
convention : $f[0] \leftarrow 0$

```
1: FONCTION R(d, f, i, n)
2: m ← i + 1
3: TANT QUE m ≤ n ET d[m] < f[i] FAIRE
4:     m ← m + 1
5: SI m ≤ n ALORS
6:     RETOURNER {m} ∪ R(d, f, m, n)
7: SINON
8:     RETOURNER ∅
```

Appel : R(d,f,0,n)

version itérative

```
1: FONCTION glouton(d, f, n)
2: A ← {1}
3: k ← 1
4: POUR i DE 2 à n FAIRE
5:     SI d[i] ≥ f[k] ALORS
6:         A ← A ∪ {i}
7:         k ← i
8: RETOURNER A
```

Appel : glouton(d,f,n)

Complexité ?

Algorithme glouton

Entrées : tableaux d et f triés

version récursive
convention : $f[0] \leftarrow 0$

```
1: FONCTION R(d, f, i, n)
2: m ← i + 1
3: TANT QUE m ≤ n ET d[m] < f[i] FAIRE
4:     m ← m + 1
5: SI m ≤ n ALORS
6:     RETOURNER {m} ∪ R(d, f, m, n)
7: SINON
8:     RETOURNER ∅
```

Appel : R(d,f,0,n)

version itérative

```
1: FONCTION glouton(d, f, n)
2: A ← {1}
3: k ← 1
4: POUR i DE 2 à n FAIRE
5:     SI d[i] ≥ f[k] ALORS
6:         A ← A ∪ {i}
7:         k ← i
8: RETOURNER A
```

Appel : glouton(d,f,n)

Complexité : $O(n)$ si les tableaux d et f sont triés, $O(n \log n)$ sinon (complexité du tri)

IRIT CNRS - INPT - UPS - UT1 - UT2J

UNIVERSITÉ TOULOUSE III PAUL SABATIER

Correction (ou preuve) de l'algorithme

Théorème : Le résultat de l'algorithme est optimal

1. Il existe une solution optimale qui commence par c_1 : soit A une sol. opt., et soit c_{a_1} le premier cours ; si $c_{a_1} = c_1$, alors $A \setminus c_{a_1} \cup c_1$ est aussi une sol. opt.
2. Le problème se ramène à trouver une solution optimale d'éléments de S compatibles avec c_1. Donc si A est une solution optimale pour S, alors $A \setminus c_1$ est une solution optimale pour $S' = \{c_i | d_i \geq f_1\}$. En effet si l'on pouvait trouver B' une solution optimale pour S' contenant plus de clients que A', alors ajouter c_1 à B' offrirait une solution optimale pour E contenant plus de clients que A, ce qui contredirait l'hypothèse que A est optimale.

Correction (ou preuve) de l'algorithme

Théorème : Le résultat de l'algorithme est optimal

1. Il existe une solution optimale qui commence par c_1 : soit A une sol. opt., et soit c_{a_1} le premier cours ; si $c_{a_1} = c_1$, alors $A \setminus c_{a_1} \cup c_1$ est aussi une sol. opt.
2. Le problème se ramène à trouver une solution optimale d'éléments de S compatibles avec c_1. Donc si A est une solution optimale pour S, alors $A \setminus c_1$ est une solution optimale pour $S' = \{c_i | d_i \geq f_1\}$. En effet si l'on pouvait trouver B' une solution optimale pour S' contenant plus de clients que A', alors ajouter c_1 à B' offrirait une solution optimale pour E contenant plus de clients que A, ce qui contredirait l'hypothèse que A est optimale.

CNRS - INPT - UPS - UT1 - UT2J

UNIVERSITÉ TOULOUSE III PAUL SABATIER

Codage de Huffman

Problème : Compresser un fichier texte T (archivage, transfert)
Idée : deux lectures du texte T

1. 1ère passe : calculer la fréquence de chaque caractère dans T
2. 2ème passe : encoder chaque caractère en binaire : les caractères les plus fréquents ont les codes les plus courts

Exemple

Fichier de 100000 caractères (6 caractères distincts)

Caractère	a	b	c	d	e	f
Fréquence	45%	13%	12%	16%	9%	5%
Code fixe	000	001	010	011	100	101
Code variable	0	101	100	111	1101	1100

Taille du fichier compressé :

1. Code fixe (sur 3 bits) : $3 * 100000 = 300000$ bits
2. Code variable : $1 * 45000 + 3 * 13000 + 3 * 12000 + 3 * 16000 + 4 * 9000 + 4 * 5000 = 224000$ bits

Exemple

Fichier de 100000 caractères (6 caractères distincts)

Caractère	a	b	c	d	e	f
Fréquence	45%	13%	12%	16%	9%	5%
Code fixe	000	001	010	011	100	101
Code variable	0	101	100	111	1101	1100

Taille du fichier compressé :

1. Code fixe (sur 3 bits) : $3 * 100000 = 300000$ bits
2. Code variable : $1 * 45000 + 3 * 13000 + 3 * 12000 + 3 * 16000 + 4 * 9000 + 4 * 5000 = 224000$ bits

Code préfixe : l'encodage d'un caractère n'est le préfixe d'aucun autre

$\longrightarrow$ simple concaténation à l'encodage et pas d'ambiguïté au décodage.

Exemple

Fichier de 100000 caractères (6 caractères distincts)

Caractère	a	b	c	d	e	f
Fréquence	45%	13%	12%	16%	9%	5%
Code fixe	000	001	010	011	100	101
Code variable	0	101	100	111	1101	1100

Taille du fichier compressé :

1. Code fixe (sur 3 bits) : $3 * 100000 = 300000$ bits
2. Code variable : $1 * 45000 + 3 * 13000 + 3 * 12000 + 3 * 16000 + 4 * 9000 + 4 * 5000 = 224000$ bits

Code préfixe : l'encodage d'un caractère n'est le préfixe d'aucun autre

→ simple concaténation à l'encodage et pas d'ambiguïté au décodage.

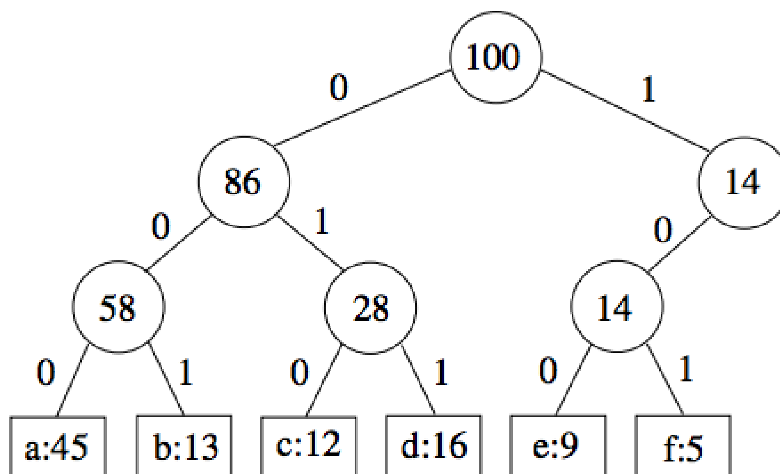
Ex. 001011101.
Code fixe : 001 011 101 → *bdf*.
Code variable : 0 0 101 1101 → *aabe*.

Code préfixe et arbre digital

Un code préfixe peut toujours être représenté à l'aide d'un arbre binaire (arbre digital).

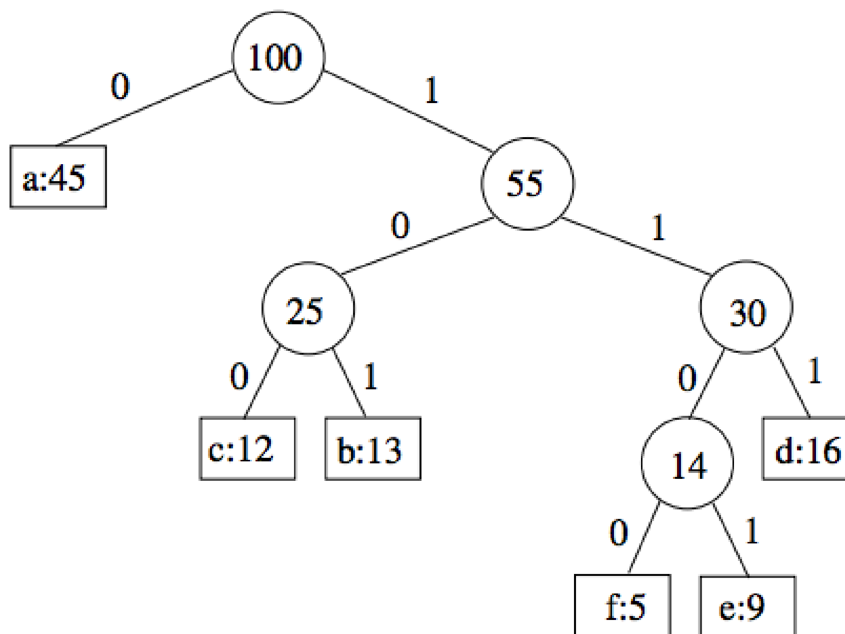
Si l'arbre n'est pas complet alors le code n'est pas optimal.

Caractère	a	b	c	d	e	f
Fréquence	45%	13%	12%	16%	9%	5%
Code fixe	000	001	010	011	100	101



Code préfixe et arbre digital

Caractère	a	b	c	d	e	f
Fréquence	45%	13%	12%	16%	9%	5%
Code variable	0	101	100	111	1101	1100



Taille du fichier compressé

Texte T sur l'alphabet A

- $f(c)$ dénote la fréquence du caractère $c \in A$ dans T
- $prof(c)$ dénote la profondeur du caractère c dans l'arbre digital (= taille du codage de c)

Le nombre de bits requis pour encoder le texte T
(i.e. la taille du fichier compressé) est

$$B(T) = \sum_{c \in A} f(c) prof(c).$$

Construction de l'arbre de Huffman

Entrées : Fréquences $(f_1, \ldots, f_n)$ des lettres (a_i) d'un texte T.

Sortie : Arbre digital donnant un code préfixe minimal pour T.

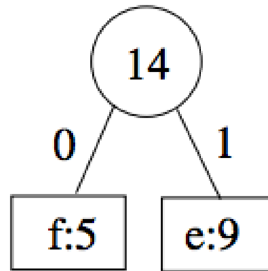
1. Créer, pour chaque lettre a_i, un arbre (réduit à une feuille) qui porte comme poids la fréquence $f(i)$.
2. Itérer le processus suivant :
   - choisir 2 arbres G et D de poids minimum
   - créer un nouvel arbre R, ayant pour sous-arbre gauche (resp. droit) G (resp. D) et lui affecter comme poids la somme des poids de G et D.
3. Arrêter lorsqu'il ne reste plus qu'un seul arbre : c'est l'arbre de Huffman

CNRS - INPT - UPS - UT1 - UT2J

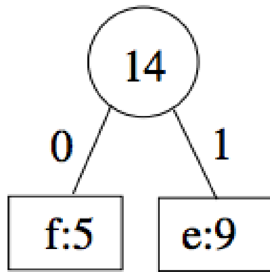
UNIVERSITÉ TOULOUSE III PAUL SABATIER

f:5 e:9 c:12 b:13 d:16 a:45

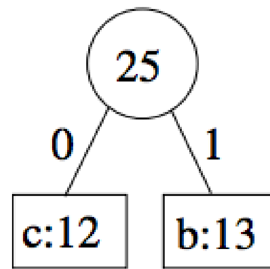
c:12 b:13



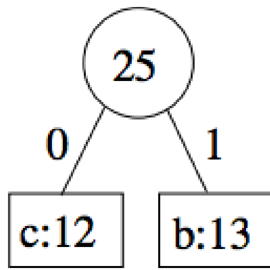
d:16 a:45

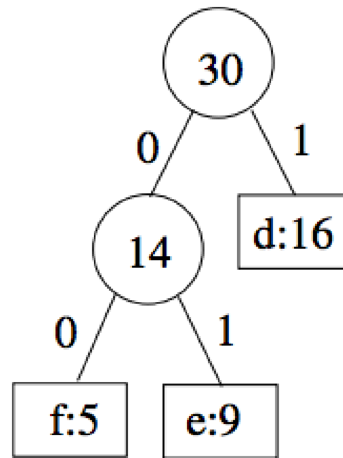


d:16

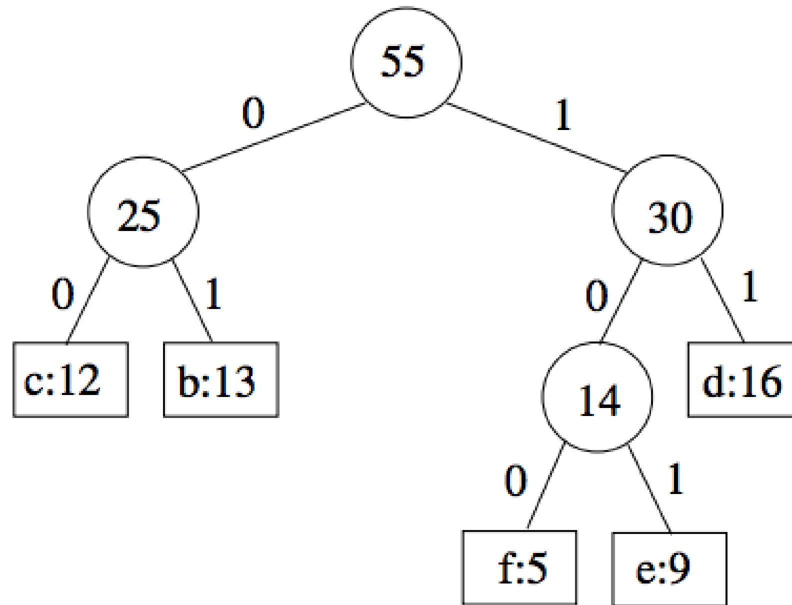


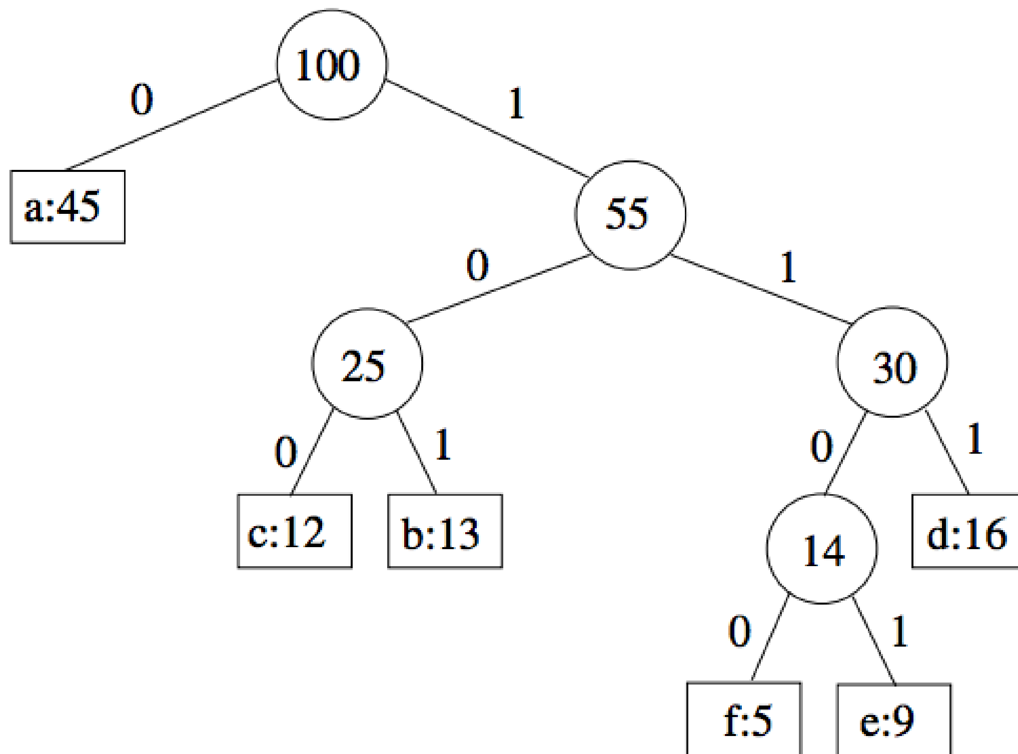
a:45





a:45


a:45
55
0
1
25
30
0
1
0
1
c:12
b:13
14
d:16
0
1
f:5
e:9


100
0
1
a:45
55
0
1
25
30
0
1
0
1
c:12
b:13
14
d:16
0
1
f:5
e:9

Complexité de la construction

1. Construction initiale et tri : $O(n \log n)$
2. A chaque itération :
   - extraire 2 fois l'arbre de poids min et fabriquer un nouvel arbre à partir des 2 précédents : $O(1)$
   - rajouter ce nouvel arbre à la liste triée : $O(\log n)$
3. Il y a $n - 1$ itérations

La construction de l'arbre de Huffman est donc en $O(n \log n)$ comparaisons

Optimalité

Optimalité : le code préfixe complet de l'arbre de Huffman est minimal, *i.e.* aucun arbre digital ne compresse mieux le texte T que *Huffman(A, f(i))*

- Propriété de choix glouton Soient x le caractère ayant la plus petite fréquence, et y le caractère ayant la seconde plus petite fréquence. Il existe un codage préfixe optimal tel que x et y aient le même père
- Propriété de sous-structure optimale On considère l'alphabet $A' = A \setminus \{x, y\} \cup \{z\}$, où z est une nouvelle lettre ayant pour fréquence $f(z) = f(x) + f(y)$
Soit T' l'arbre d'un codage optimal pour A', alors l'arbre T obtenu à partir de T' en remplaçant la feuille associée à z par un noeud interne ayant x et y comme feuilles représente un codage optimal pour A

IRIT CNRS · INPT · UPS · UT1 · UT2J

UNIVERSITÉ TOULOUSE III PAUL SABATIER Université de Toulouse

Optimalité

Optimalité : le code préfixe complet de l'arbre de Huffman est minimal, *i.e.* aucun arbre digital ne compresse mieux le texte T que *Huffman(A, f(i))*

- Propriété de choix glouton Soient x le caractère ayant la plus petite fréquence, et y le caractère ayant la seconde plus petite fréquence. Il existe un codage préfixe optimal tel que x et y aient le même père
- Propriété de sous-structure optimale On considère l'alphabet $A' = A \setminus \{x, y\} \cup \{z\}$, où z est une nouvelle lettre ayant pour fréquence $f(z) = f(x) + f(y)$
  Soit T' l'arbre d'un codage optimal pour A', alors l'arbre T obtenu à partir de T' en remplaçant la feuille associée à z par un noeud interne ayant x et y comme feuilles représente un codage optimal pour A