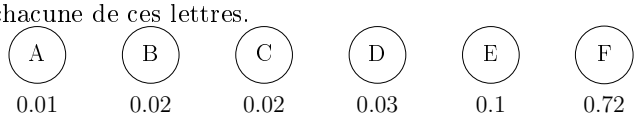
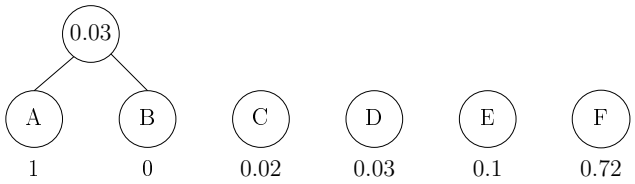
■ T.P. 8 ■
Codage de Huffman

L'ensemble du T.P. est consacré à l'étude de l'algorithme de Huffman (1952) qui permet de coder un texte caractère par caractère à l'aide d'une chaîne binaire (i.e. une suite de 0 et de 1) en minimisant la longueur totale de la chaîne obtenue ; cet algorithme permet de faire de la compression de données. Par exemple, si la lettre e est plus fréquente que les autres lettres du texte, elle sera codée à l'aide d'un code plus court.

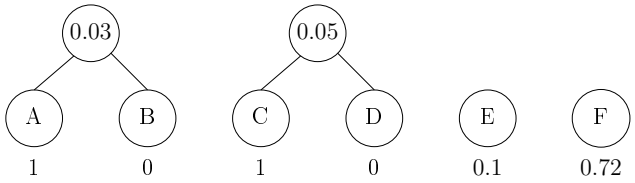
Exemple. On considère un texte t constitué des 6 lettres A, B, C, D, E, F de fréquences respectives $(0.01, 0.02, 0.02, 0.03, 0.1, 0.72)$. On construit itérativement un arbre qui sera ensuite utilisé pour déterminer le code associé à chacune de ces lettres.



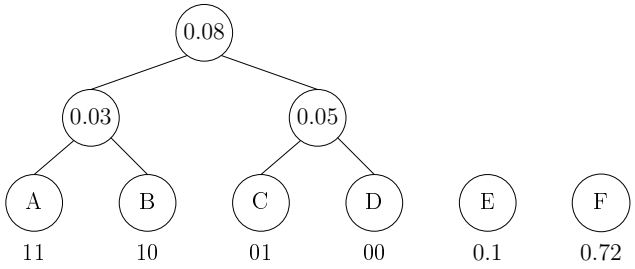
(i) On regroupe les 2 fréquences les plus faibles en un arbre au sommet duquel on inscrit la somme des fréquences. Les fréquences utilisées sortent alors du jeu, mais le nouveau somment entre dans le jeu, porteur de la somme des fréquences. Les feuilles de gauches se voient attribuées le bit 1, celles de droite le bit 0.



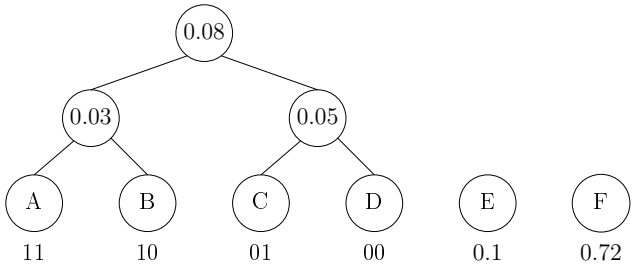
(ii) On recommence ensuite avec les nœuds restants.



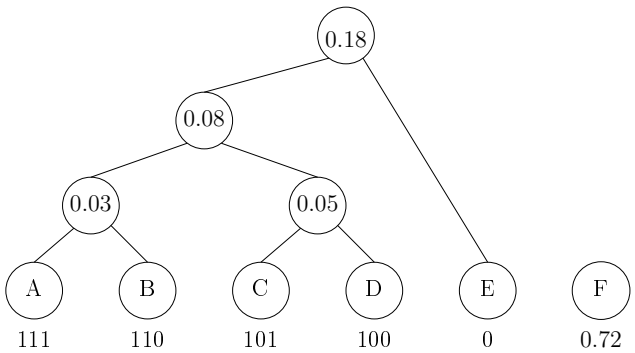
(iii) On réitère la procédure. Les feuilles du sous-arbre de gauche se voient ajoutées le bit 1, celles de droite le bit 0.



(iv) L'étape suivante permet de construire l'arbre suivant.



(v) Enfin, l'arbre final obtenu est le suivant.



Notion de tas. Le module `heapq` permet de manipuler des tas. Les tas sont des données structurées qui permettent d'obtenir rapidement la donnée de poids minimal.

- `heapify(liste)` transforme une liste en tas.
- `heappop(tas)` extrait (efficacement) un élément minimal de `tas`.
- `heappush(tas, nv_element)` ajoute à `tas` le nouvel élément `nv_element` en préservant la structure de tas.

Dictionnaires. Un dictionnaire exprime une relation entre un ensemble de clés et un ensemble de valeurs.

- `{}` est le dictionnaire vide.
- `{k1:v1,...,kn:vn}` permet de construire explicitement un dictionnaire. Par exemple, `{'a' : 0, 'b' : 10}` est le dictionnaire contenant les clés `'a'` et `'b'`. Les valeurs associées à ces clés sont respectivement 0 et 10.
- `<dico>[<cle>]` permet d'accéder à la valeur associée à la clé `<cle>`.
- `<dico>[<cle>] = <valeur>` permet de créer la clé `<cle>` dans le dictionnaire `<dico>` et d'y affecter la valeur `<valeur>`. Si la clé était déjà présente, sa valeur est écrasée.
- `<cle> in <dico>` teste l'appartenance de `<cle>` au dictionnaire `<dico>`.
- `for <cle> in <dico>` permet d'itérer sur les clés.
- `for (<cle>, <valeur>) in <dico>.items()` permet d'itérer sur les associations.

1. Écrire une fonction `occurrences(texte)` qui, étant donné un texte `texte`, renvoie un dictionnaire dont les clés sont les lettres de `texte` et les valeurs en sont les occurrences.

Pour modéliser l'arbre de Huffman, nous utilisons un tas. À chaque étape, les éléments du tas sont les arbres. On stocke le poids de l'arbre ainsi que le codage associé aux feuilles correspondantes. Pour un arbre ne contenant que la racine, le code sera `''`.
Dans l'exemple précédent, les valeurs successives de `tas` seront :

- `[(0.01, ['A','']), (0.02, [['B','']]), (0.02, [['C','']]), (0.03, [['D','']]), (0.1, [['E','']]), (0.72, [['F','']])]`
- `[(0.03, [['A','1'],['B','0']], (0.02, [['C','']]), (0.03, [['D','']]), (0.1, [['E','']]), (0.72, [['F','']])]`
- `[(0.03, [['A','1'],['B','0']], (0.05, [['C','1'], ['D','0']]), (0.1, [['E','']]), (0.72, [['F','']])]`
- ...
- `[(1, [['A','1111'],['B','1110'],['C','1101'], ['D','1100'], ['E','10'], ['F','0']])]`

2. Écrire une fonction `code_huffman(occurrences)` qui, étant donné un dictionnaire d'occurrences, retourne le dictionnaire des codes.

3. Écrire une fonction `encodage(texte, code)` qui, étant donné un texte `texte` et un dictionnaire de code `code`, renvoie le texte codé.

4. Écrire une fonction `decode(texte_bin, code)` qui, étant donné un texte binaire `texte_bin` et un dictionnaire de code `code`, renvoie le texte décodé.

5. Écrire des fonctions de codage / décodage utilisant le codage ASCII.
On pourra utiliser les fonctions `ord` *et* `bin`.

6. Évaluer la qualité de compression du codage de Huffman.