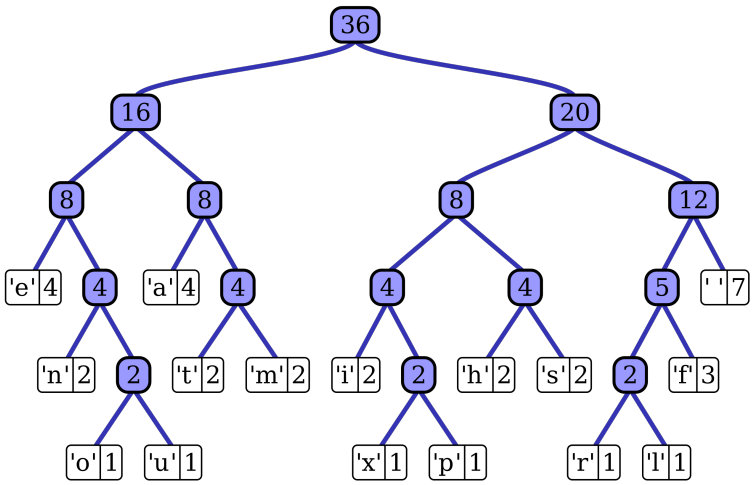
Cette *vingt-troisième* colle vous fera travailler sur les algorithmes du codage et décodage de Huffman, sans écrire de code (tout au brouillon).

Ex.1 Algorithme de codage de Huffman

1. Écrire proprement (en pseudo-code) l'algorithme qui prend un texte `t` écrit sur un alphabet Σ et construit l'arbre binaire de Huffman $A_H(\texttt{t})$.
2. Justifier (rapidement) la terminaison de cet algorithme.
3. Exécuter cet algorithme sur le texte `t` suivant : `t = "A BAS BABAR"` :
   - Construire l'arbre $A_H(\texttt{t})$ à partir du texte `t`, étape par étape.
   - En déduire la clé de codage binaire $c_{A_H(\texttt{t})}$ calculée à partir de l'arbre, qui à chaque lettre $a \in \Sigma$ de l'alphabet associe son codage binaire $c_{A_H(\texttt{t})}(a) \in \{0,1\}^*$.
   - En déduire l'encodage binaire du texte initial, sans chercher à encoder l'arbre dans ce message.
   - Si on code chaque lettre de l'alphabet $\Sigma = \{\texttt{'A'}, \texttt{'B'}, \texttt{' '}, \texttt{'S'}, \texttt{'R'}\}$ sur le même nombre de bits b, quel est le nombre minimal de bits b suffisant pour tous les coder et décoder sans ambiguïté ?
     En déduire la taille de la représentation binaire du texte initial, comme `|t| * b`.
   - Avec la clé de codage binaire $c_{A_H(\texttt{t})}$, quelle est la taille occupée par l'encodage binaire (de Huffman) du texte `t` ?
   - Conclure : est-ce que la compression a été utile ici ? (encore une fois, sans chercher à encoder l'arbre A_H dans le codage binaire produit)

Ex.2 Algorithme de décodage de Huffman

4. On considère l'arbre binaire de Huffman suivant :



   - En utilisant cet arbre, donner la clé de codage binaire associée (c'est-à-dire la fonction c_{A_H} qui à chaque lettre $a \in \Sigma$ associe un encodage binaire $c_{A_H}(a) \in \{0,1\}^*$).
   - En utilisant cet arbre, décoder le (long) message suivant : `0110 1010 1000 1011 111 1000 1011 111 010 0010 111 000 10010 010 0111 10011 11001 000`.

Ex.3 Analyse de complexité mémoire du codage de Huffman

5. À partir du pseudo-code de l'encodage de Huffman que vous avez donné en question 1, analysez la complexité mémoire de cet algorithme, en fonction de `|t|` la taille du mot d'entrée et/ou de $|\Sigma|$ la taille de l'alphabet, et/ou $f(\texttt{t})$ le nombre de lettres différentes de `t` (que l'on peut borner par $f(\texttt{f}) \leqslant \min(|\Sigma|, |\texttt{t}|)$ mais pour lequel on ne peut pas dire plus).

Ex.4 Analyse de la complexité temporelle pour quatre variantes du codage de Huffman

On va considérer les quatre variantes suivantes, qui diffèrent par la manière dont sont implémentées les deux opérations suivantes :

i. d'extraction des arbres (n_1, A_1) et (n_2, A_2) de nombres d'occurrence minimum (`Extraction`) dans la forêt d'arbres F,
ii. d'insertion du nouvel arbre $(n_1 + n_2, \texttt{Noeud}(A_1, A_2))$ dans la forêt F (`Insertion`).

1) Variante naïve : `Extraction` et `Insertion` naïves, mais un tri de la forêt après l'insertion

Comme en TP 19 la semaine dernière, on considère pour commencer la variante naïve du codage de Huffman : la forêt sera une liste simplement chaînée de couple (nb d'occurrence, arbre), qui sera *maintenue triée* (selon la première coordonnée) par des appels à un algorithme de tri à la fin de la boucle `while`, après l'`Insertion` du nouvel arbre.

6. Donner un exemple d'algorithme de tri qui soit efficace pour cet objectif là.
   S'il doit trier une forêt à k arbres, quelle est sa complexité temporelle en fonction de k ? $\mathcal{O}(k)$, $\mathcal{O}(k \log k)$ ou $\mathcal{O}(k^2)$?
7. Comment implémenter naïvement les deux opérations d'`Extraction` et d'`Insertion` ? (sans chercher à maintenir la forêt triée pour l'`Insertion`, vu que l'algorithme de tri est utilisé juste après)
8. Quelles seront les complexités temporelles des deux opérations `Extraction` et `Insertion` ?

On considère ensuite trois variantes améliorées, deux assez naïves et une plus compliquée, qui ne vont plus trier la forêt à la fin de la boucle `while`.

2) Première variante améliorée : `Extraction` pas triviale, `Insertion` triviale

On considère une première variante améliorée, qui ne va plus chercher à maintenir la forêt triée. Au lieu de ça, l'opération d'`Extraction` sera plus compliquée, alors que l'opération d'`Insertion` sera triviale et en temps constant.

9. Que proposez-vous pour ces deux opérations ?
10. Quelles sont leurs complexités temporelles, en fonction de k le nombre d'arbres dans la forêt F ?

3) Deuxième variante améliorée : `Extraction` triviale, `Insertion` pas triviale

On considère une deuxième variante améliorée, qui va de nouveau chercher à maintenir la forêt triée. L'opération d'`Insertion` sera alors plus compliquée que dans l'approche naïve 1), alors que l'opération d'`Extraction` sera triviale et en temps constant.

11. Que proposez-vous pour ces deux opérations ?
12. Quelles sont leurs complexités temporelles, en fonction de k le nombre d'arbres dans la forêt F ?

4) Variante optimale : `Extraction` et `Insertion` plus efficace, au prix d'une structure plus compliquée

Pour la dernière variante, que vous implémenterez pour le DM 06 pendant les vacances, on ne cherche plus non plus à maintenir la forêt triée. Les deux opérations seront toutes les deux plus compliquées, mais plus efficaces que l'opération la plus compliquée des deux, dans les deux variantes précédentes.

On stocke la forêt non plus comme une liste simplement chaînée de couples (nb occurrence, arbre) mais comme une file de priorité min, stockée par un tas binaire.

13. Sans rappeler les deux algorithmes, donner les complexités temporelles pour les opérations d'`Extraction` et d'`Insertion`, en fonction de k le nombre d'arbres dans la forêt F.
14. (bonus, à la fin) Rappeler le fonctionnement des deux algorithmes d'`Extraction` et d'`Insertion` sur un tas binaire.

Analyse de complexité temporelle dans les 4 cas

15. Pour chacune des quatre variantes 1) 2) 3) et 4) expliquées ci-dessus, analysez la complexité temporelle de l'algorithme du codage de Huffman, qui produit l'arbre $A_H(\texttt{t})$ à partir du texte `t`.
    - Astuce : analysez d'abord l'algorithme dans sa totalité, en utilisant $C_E(k)$ et $C_I(k)$ les complexités temporelles des deux opérations `Extraction` et `Insertion` pour une forêt à k arbres.
    - Ensuite, remplacez ces deux coûts $C_E(k)$ et $C_I(k)$ par ceux trouvés en 1) 2) 3) et 4) précédemment, et conclure.

