Le projet consiste à développer une application Clients/Serveur qui permet à des clients de récupérer des fichiers enregistrés sur le serveur en utilisant le protocole de communication sous TCP-IP. Le client se connectera à un serveur de fichier, le serveur proposera une liste de fichiers à télécharger et le client en choisira certains. Une fois le choix validé le serveur compressera le fichier en utilisant le code de Huffman ou de Lempel Ziv Welch puis transmettra le fichier compressé et le dictionnaire de codage au client qui devra le décompresser le fichier à la réception et l’enregistrer en local.  
  
  
Partie 1 - Code de Huffman   
  
Le code de Huffman est un code permettant de comprimer un texte : l’idée de base est de coder les caractères avec un code de longueur variable ; les caractères les plus fréquents sont codés avec un code plus court que les caractères les moins fréquents. Pour coder un texte, on commence par calculer l’occurrence des différents caractères du texte. Le code de Huffman est ensuite construit en calculant un arbre de la façon suivante (voir descriptif) :  
"construction de la liste de Huffman" : Les caractères sont rangés dans une liste ordonnée par ordre croissant des occurrences.  
"construction de l’arbre de Huffman" : Tant que la liste contient au moins deux éléments on retire les deux premiers éléments puis on ajoute dans la liste un nouvel élément dont l’occurrence est la somme des occurrences, et qui possède le premier élément en tant que sous arbre droit et le second en tant que sous arbre gauche.   
  
Enfin le code de Huffman est une suite de 1 et de 0 que l’on obtient pour chaque caractère en parcourant l’arbre depuis la racine jusqu’à la feuille en question (qui contient le caractère). Par convention, on aura un code 1 quant on descend à gauche dans l’arbre et 0 lorsqu’on descend à droite.  
  
  
  
  
Autre Exemple : "tata et lo" Occurrences : ((e,1) (l,1) (o,1) (a,2) (‘ ‘ ,2) (t,3))  
  
Construction de l’arbre de huffman  
  
Création des classes Element, Feuille et Noeud  
  
Trois Classes permettront de traiter au sein de l’arbre des objets différents types. En effet la structure que nous venons de présenter peux contenir deux types d’éléments : les « feuilles » de l’arbre qui sont les éléments terminaux et contiennent deux informations tels que l’occurrence du caractère et le caractère lui-même. Les « nœuds » qui sont les éléments intermédiaires et qui ont une occurrence et deux enfants qui à leur tour pourront être des feuilles ou des nœuds.  
Créez une première classe Element qui sera la classe mère des deux sous classes Feuille et Nœud. Cette classe contiendra en partie privée les données communes aux feuilles et aux nœuds (occurrence). Créez pour cette classe un constructeur qui initialise l’occurrence à partir d’une valeur passée en argument ;   
Proposer un accesseur pour accéder en lecture/écriture à l’occurrence de Element et une méthode permettant de comparer les occurrences des éléments. Pensez à rendre la classe comparable ;   
Ajoutez maintenant les deux sous classes Feuille et Nœud qui hériteront de Element et qui auront pour la première un champ caractère et pour la seconde deux champs vers les éléments de droite et de gauche. Définissez pour ces champs des accesseurs en lecture ;  
Dans la classe Feuille implémenter un constructeur pour créer une instance avec la valeur du caractère et son occurrence. Proposez également un accesseur en lecture sur le caractère ;  
Pour la classe Nœud implémentez deux accesseurs en lecture pour les branches gauche et droite, et un constructeur pour la création du Nœud avec une fréquence et les deux Element de gauche et de droite ;   
Surchargez pour ces classes les fonctions d’affichage.  
  
Construction de la liste de Huffman  
  
A partir d’une nouvelle classe nous nous proposons de créer le code de Huffman.  
Cette classe contient trois champs, une chaine de caractères qui correspond au fichier à coder, une liste d’Elements qui contiendra l’arbre de Huffman et un dictionnaire qui contiendra le code de Huffman correspondant au fichier ;  
Implémentez la fonction qui permet de lire le fichier à coder ;  
Surchargez maintenant la méthode Equals pour les Feuille. Cette méthode retournera True si le caractère contenu dans la feuille courante est égal à celui de la feuille passée en argument ;  
Implémentez maintenant la fonction qui permet de générer la liste de Huffman. Cette fonction devra parcourir tout le texte à coder, pour chaque caractère vous testerez ci celui-ci est présent dans la liste. Vous utiliserez pour cela la méthode IndexOf des List. Si le caractère est présent dans la liste, il faut incrémenter son occurrence et retrier la liste, sinon vous devez ajoutez cette Feuille en début de liste ;  
  
Construction de l’arbre de Huffman  
  
Dans cette partie vous devez retirer les deux premiers éléments de la liste tant que celle-ci n’est pas réduite à un seul élément, puis vous devez créer un Nœud à partir de ces deux éléments, le réinsérer dans la liste et la trier.  
Implémentez cette fonction ;  
  
Code de Huffman  
  
Pour coder un texte, en utilisant le code de Huffman, nous avons besoin d’un dictionnaire qui permet de traduire chaque caractère en son code de Huffman. Un dictionnaire est une dictionary contenant au rang c le code de Huffman du caractère. Le code de huffman est une suite de 1 et de 0 que l’on obtient pour chaque caractère en parcourant l’arbre depuis la racine jusqu’à la feuille en question. Par convention, on aura un code 1 quant on descend à gauche dans l’arbre et 0 lorsqu’on descend à droite.  
Définissez une premier méthode creerCode qui initialisera le code d’un caractère à vider, se placera sur la racine de l’arbre et appellera une méthode coder ;   
Cette méthode parcourt l’arbre tant que l’on n’atteint pas les Feuilles. Si l’on se trouve sur un Nœud on appelle récursivement la fonction pour le sous arbre droit et pour le sous arbre gauche ;  
Avant l’appel du sous arbre droit on ajoutera un ‘0’ dans le code que l’on retirera une fois le parcours achevé. On fera la même chose avec ‘1’ pour le parcours du sous arbre gauche ;  
Le parcours s’arrête dès que l’on rencontre un objet de type Feuille. La méthode is permet de connaitre le type de l’Element courant ;  
Dès que l’on atteint une Feuille le caractère de la Feuille ainsi que le code obtenu seront enregistré dans le dictionnaire ;  
Ajoutez une méthode permettant l’affichage du code de Huffman.  
  
Codage/Décodage et affichage  
  
Surchargez les méthodes ToString de toutes les classes ;  
Implémenter les fonctions permettant d’afficher la liste des Feuilles avec leurs fréquences, le code de Huffman et le texte codé dans des fichiers intermédiaires ;  
Ajouter une méthode qui permet de coder le texte d’origine ;  
Ecrire un nouveau programme qui lit un texte codé, le code de Huffmane et le décode en utilisant le code de Huffman ;  
Utiliser les classes BinaryWriter et BinaryReader pour lire et écrire le dictionnaire sur un fichier.   
  
  
Partie 2 – Communication via TCP-IP   
  
Dans cette partie nous implémenterons un processus de communication entre des clients potentiels et un serveur. L’adresse localhost nous servira pour « simuler » le fonctionnement du réseau. La classe Sockets permet de spécifier le protocole à utiliser, le type de transfert de données, et toutes les informations utiles à la communication entre le client et le serveur. Afin de simplifier le travail nous utiliserons les classes TcpListener et TcpClient qui sont des objets dédiés à la communication en utilisant le protocole TCP.  
  
Structure de base du serveur  
  
Cette classe permet de démarrer le service et de mettre le serveur en attente des clients. Dès qu’un client se manifeste un flux est créé le traitement du client fait dans un thread afin que le serveur puisse traiter d’autre demandes.  
Implémenter une classe de base permettant de configurer le serveur en utilisant un objet de type TcpListener qui devra avoir l’adresse IP à laquelle se « binder » (ici localhost) et le port sur lequel on souhaite écouter ;  
Une fois l’objet créé lancez l’écoute via la méthode Start et attendre qu’un client se manifeste. Pour cela vous pourrez utiliser une boucle infinie qui se débloquera sur la méthode AcceptTcpClient, cette méthode retour un flux de type TcpClient qui permettra une communication par flux entre le client et le serveur. Ce flux, obtenu via partir de la méthode GetStream() est optimisé pour utiliser des String ce qui évite de transmettre des tableaux de Byte, comme dans le cas d’une configuration par Sockets ;  
Une fois le flux associé au client récupéré, vous démarrerez le service lié au traitement du client dans un thread afin que le serveur puisse servir en parallèle d’autres clients.   
  
Traitement du client  
  
La classe Service associé au traitement du client récupérera la liaison client ou son flux associé lors de sa création.   
Implémentez un constructeur permettant de récupérer le flux lié au client qui s’est connecté au serveur. Vous pourrez ensuite créer un flux de lecture et un flux d’écriture permettant de voir ce que le client demande et de lui répondre en conséquence ;   
Implémentez maintenant la fonction de traitement du client. Celle-ci attendra consistera à donner au client la liste des fichiers disponibles pour le téléchargement, à attendre que celui-ci fasse son choix puis à compresser les fichiers en utilisant le code de Huffman, à envoyer la table de codage et ensuite les fichiers ;  
Cette procédure se terminera à la demande du client ou si celui-ci n’est plus accessible, suite à une déconnexion ou à un problème de communication.  
  
Traitement coté client  
  
La classe coté client sera le pendant de celle coté serveur. Elle permettra la configuration d’un objet de type TcpClient vers le serveur sur le port choisi pour la communication.   
Créer une fonction permettant de se connecter au serveur ;  
Si le client est accepté à la réception du flux créer un flux de lecture et un d’écriture permettant de communiquer avec le serveur. A la réception de la liste des fichiers téléchargeables le client fera son choix et le communiquera au serveur, qui lui transmettra en retour la table de codage et les fichiers demandé ;  
Le client pourra décompresser les données et enregistrer les fichiers sur son espace mémoire.  
c