TP3 - Tas binaires

Ce qu'on voit lors de ce TP

- L'implémentation des files de priorités avec des tas binaires
- L'utilité des files de priorités dans certains problèmes algorithmiques

Téléchargez le fichier `file_priorite.h` . Déplacez-le dans un répertoire personnel, ainsi que vos deux fichiers `liste.c` et `liste.h` .

I) Tas binaires minimum

Bon vous commencez à comprendre le fonctionnement des TP !

Créez un fichier `file_priorite.c` et codez toutes les fonctions dont le prototype se trouve dans `file_priorite.h` .

II) Application : destruction de cairns

Vous vouez un culte de détestations des cairns, ces empilements de cailloux qui jonchent les chemins de randonnées.

Vous vous résolvez à renverser le maximum de cailloux sur votre chemin. Chaque

fois que vous poussez un cairn, la moitié de ses cailloux (arrondie à l'inférieur) tombe. Combien de cailloux restera-t-il au minimum après que vous ayez procédé à **k** chamboulements de cairns (différents ou non) ?

On va considérer par exemple $k = 5$ et la liste d'entiers : `[17, 321, 28, 50]`, listant le nombre de cailloux dans chaque cairn. Ici cela signifie qu'il y a 4 cairns, le premier ayant 17 cailloux, le deuxième 321, le troisième 28 et enfin le dernier 50.

Comment faire ? Au mieux pour le premier chamboulement vous pouvez renverser 160 cailloux en chamboulant le deuxième cairn : la liste devient `[17, 161, 28, 50]`. Puis 80 cailloux toujours dans le deuxième cairn → `[17, 81, 28, 50]`. Puis 40 (toujours le deuxième) → `[17, 41, 28, 50]`. Pour le quatrième chamboulement, il est plus avantageux de renverser le dernier cairns : 25 cailloux en moins → `[17, 41, 28, 25]`. Et enfin pour le dernier, on re-chamboule le deuxième pour enlever 20 cailloux : → `[17, 21, 28, 25]`. Au final, il reste $17 + 21 + 28 + 25 = 91$ cailloux au minimum, c'est ce qu'on souhaite calculer automatiquement !

Écrire une fonction qui étant donnée une liste listant le nombre de cailloux dans chaque cairn, ainsi qu'un nombre k de chamboulements, renvoie le nombre minimal de cailloux restants après k chamboulements de moitiés de cairns.

IMPORTANT aussi : Quelle est la complexité de votre algorithme ? A justifier !

Je vous ai écrit un petit programme pour tester si votre fonction marche correctement :

```c
#include <assert.h>
#include <time.h>
#include <limits.h>

/**
 * @brief Permet de vérifier le bon fonctionnement et la rapidité de
 * votre fonction qui compte le nombre de cailloux minimaux
 * restants parmi les cairns après un certain nombre de chamboulements. \n
 * Une liste de la bonne taille est tirée aléatoirement.
 * @param votre_fonction la fonction que vous avez écrit
 * (elle doit prendre une Liste et un size_t en entrées et un size_t en sort
 * @param taille la taille de la liste tirée aléatoirement pour le test.
 */
void test_cairns(size_t (*votre_fonction) (Liste, size_t), size_t taille ){

    // Je commence par construire une liste aléatoire de la bonne taille :
    Liste l = liste_vide();

    int x = 1;
    for(size_t i = 0; i < taille ; i++){
```

```
        size_t j = rand() % (i+1);
        ajouter_en_fin(l,x);
        modifier(l,i,element(l,j));
        modifier(l,j,x);
        x += 1 + rand() % 4;
    }

    size_t sol = 0;
    for(size_t i = 0; i < taille ; i++){
        modifier(l,i,element(l,i) + x + 1);
        sol += element(l,i);
    }



    size_t k = 0;
    for(size_t i = 0; i < taille ; i++){
        int cailloux = element(l,i);
        while( rand() % 3  < 2 && cailloux < INT_MAX/8){
            k++;
            cailloux = cailloux*2 - (rand() % 2);
        }
        modifier(l,i,cailloux);
    }


    // J'affiche la tête de l'entrée si la taille est inférieure à 30
    if (taille < 30) {
        printf("\nListe testée :");
        for(size_t i = 0; i < taille; i++){
            printf(" %d",element(l,i));
        }
        printf("\nNombre de chamboulements : %ld \n",k);
    }


    struct timespec start, end;
    clock_gettime(CLOCK_MONOTONIC, &start);

    // Je teste votre fonction ici :
    assert( votre_fonction(l,k) == sol );

    clock_gettime(CLOCK_MONOTONIC, &end);
    long int elapsed_time_ns = (end.tv_sec - start.tv_sec) * 1000000000 + (e
    printf("Temps d'exécution (pour une taille %ld et un nombre de chamboule

    liberer_liste(l);
}
```

Il suffira juste d'écrire l'instruction `test_cairns(votre_fonction,1000);` par exemple.

Jusqu'à quelle taille arrivez-vous à résoudre le problème ? Normalement vous devriez arriver jusqu'à 1 million, sans problème !