FACULTÉ DES SCIENCES ET TECHNIQUES
DE MOHAMMEDIA
UNIVERSITÉ HASSAN II DE CASABLANCA

Université Hassan II
Mohammedia Casablanca

Filière : Licence Génie Informatique

Module : Structures de Données Avancées

Rapport de Synthèse

Projet de Structures de Données

Python & C

Réalisée par : Hiba Boussairi

Encadrant F. Khoukhi

Année Universitaire : 2025 - 2026

Table des matières

Remerciements

Je tiens à exprimer ma profonde gratitude à toutes les personnes qui ont contribué à la réalisation de ce projet.

Tout d'abord, je remercie sincèrement **M. F. Khoukhi**, mon encadrant, pour sa disponibilité, ses conseils précieux et son accompagnement tout au long de ce travail. Son expertise et ses orientations m'ont permis de mener à bien ce projet.

Je remercie également l'ensemble du corps professoral de la **Faculté des Sciences et Techniques de Mohammedia** pour la qualité de l'enseignement dispensé durant cette année universitaire.

Chapitre 1

Introduction Générale

Ce projet porte sur la **visualisation interactive des structures de données et des algorithmes de tri**. L'objectif est de fournir une interface graphique permettant de comprendre visuellement le fonctionnement des algorithmes fondamentaux.

Le projet a été développé en **deux versions** : **Python** (Tkinter) et **C** (GTK 4).

Objectifs :
— Algorithmes de tri : Bubble, Insertion, Shell, Quick Sort
— Listes chaînées simples et doubles
— Arbres binaires et N-aires avec parcours DFS/BFS
— Graphes et algorithmes : Dijkstra, Bellman-Ford, Floyd-Warshall

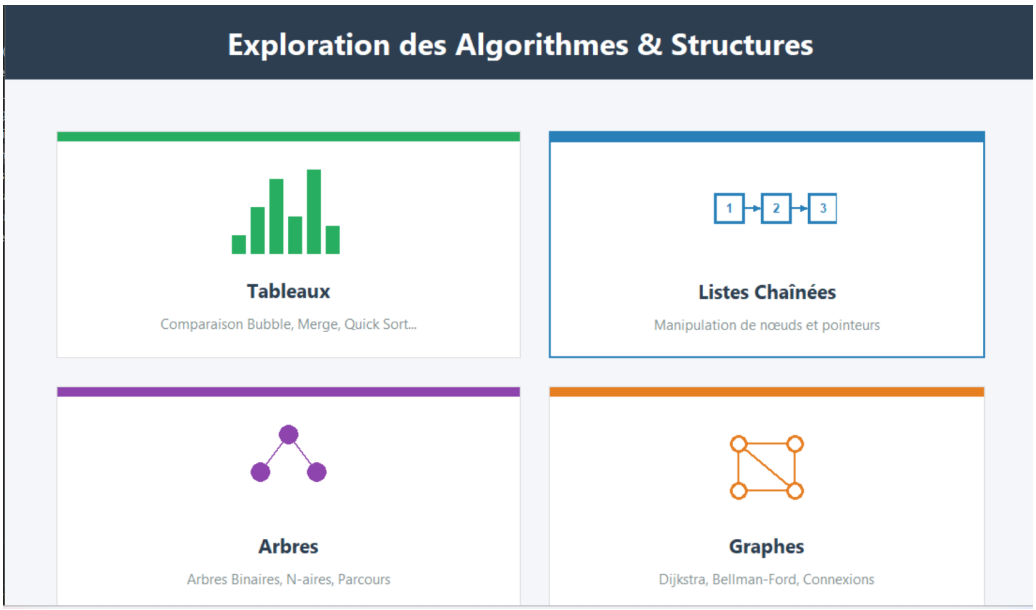

FIGURE 1.1 – Interface principale du menu de l'application

Structure du rapport : TP1 (Tableaux), TP2 (Listes), TP3 (Arbres), TP4 (Graphes), Architecture, Conclusion.

Chapitre 2

TP1 : Tri et Comparaison des Tableaux

2.1 Présentation du Module

Le module de tri permet de visualiser et comparer différents algorithmes de tri sur des tableaux de données. L'interface propose plusieurs fonctionnalités clés :

— Génération de données (aléatoires ou manuelles)
— Support de plusieurs types : Entiers, Réels, Caractères, Chaînes
— Visualisation avant/après tri
— Graphique de comparaison des performances

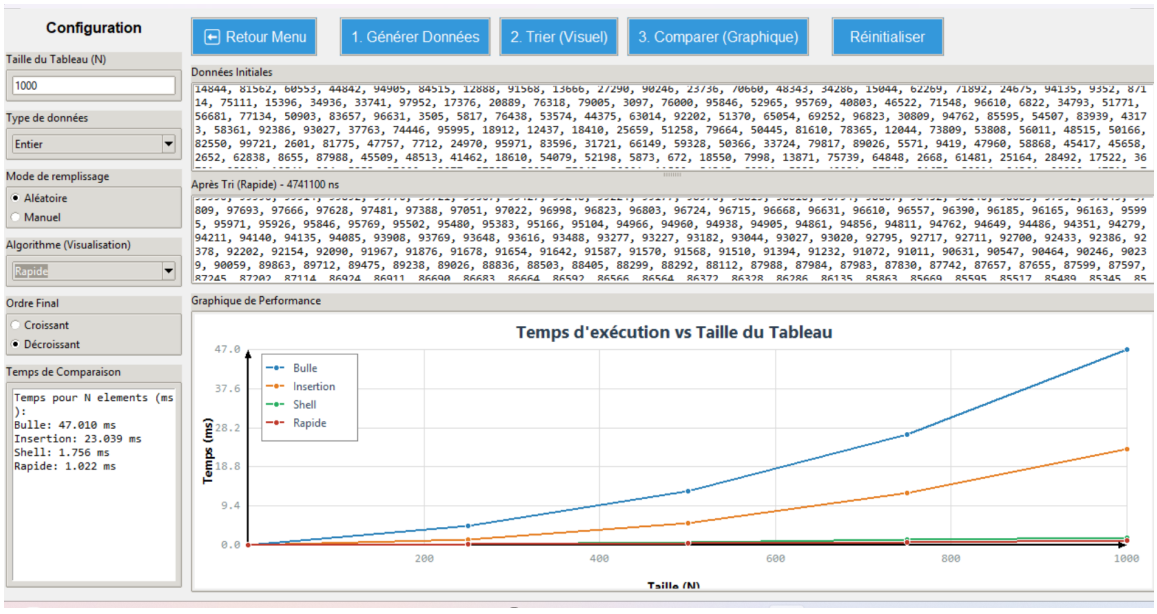

FIGURE 2.1 – Interface du module de tri avec graphique de performance

2.2 Algorithmes Implémentés

2.2.1 Tri à Bulle (Bubble Sort)

Complexité : $O(n^2)$

Le tri à bulle compare les éléments adjacents et les échange s'ils sont dans le mauvais ordre. Simple mais inefficace pour de grands ensembles.

```python
def bubble_sort(self, data, reverse):
    n = len(data)
    for i in range(n):
        swapped = False
        for j in range(0, n-i-1):
            condition = (data[j] < data[j+1]) if reverse else (data
   [j] > data[j+1])
            if condition:
                data[j], data[j+1] = data[j+1], data[j]
                swapped = True
        if not swapped: break
```

Listing 2.1 – Tri à Bulle en Python

2.2.2 Tri par Insertion (Insertion Sort)

Complexité : $O(n^2)$

Le tri par insertion construit la liste triée élément par élément en insérant chaque nouvel élément à sa bonne position.

```c
static void insertion_bench(int *arr, int n) {
    for (int k = 1; k < n; k++) {
        int key = arr[k];
        int l = k - 1;
        while (l >= 0 && arr[l] > key) {
            arr[l + 1] = arr[l];
            l--;
        }
        arr[l + 1] = key;
    }
}
```

Listing 2.2 – Tri par Insertion en C

2.2.3 Tri Shell (Shell Sort)

Complexité : O(n log² n)

Amélioration du tri par insertion utilisant des intervalles décroissants pour comparer des éléments distants.

2.2.4 Tri Rapide (Quick Sort)

Complexité : O(n log n) en moyenne

Algorithme divide-and-conquer qui sélectionne un pivot et partitionne le tableau autour de celui-ci.

```
def quick_sort_iterative(self, data, reverse):
    size = len(data)
    stack = [(0, size - 1)]
    while stack:
        l, h = stack.pop()
        if l >= h: continue

        pivot = data[h]
        i = l - 1
        for j in range(l, h):
            condition = (data[j] > pivot) if reverse else (data[j]
   < pivot)
            if condition:
                i += 1
                data[i], data[j] = data[j], data[i]
        data[i+1], data[h] = data[h], data[i+1]
        p = i + 1

        if p - 1 > l: stack.append((l, p - 1))
        if p + 1 < h: stack.append((p + 1, h))
```

Listing 2.3 – Tri Rapide Itératif en Python

2.3 Graphique de Performance

Le module génère automatiquement un graphique comparatif montrant l'évolution du temps d'exécution en fonction de la taille du tableau pour chaque algorithme.

Chapitre 3

TP2 : Gestion des Listes Chaînées

3.1 Présentation du Module

Ce module permet de manipuler visuellement des listes chaînées avec deux types :
— **Liste Chaînée Simple** : Chaque nœud pointe vers le suivant
— **Liste Chaînée Double** : Chaque nœud pointe vers le précédent et le suivant

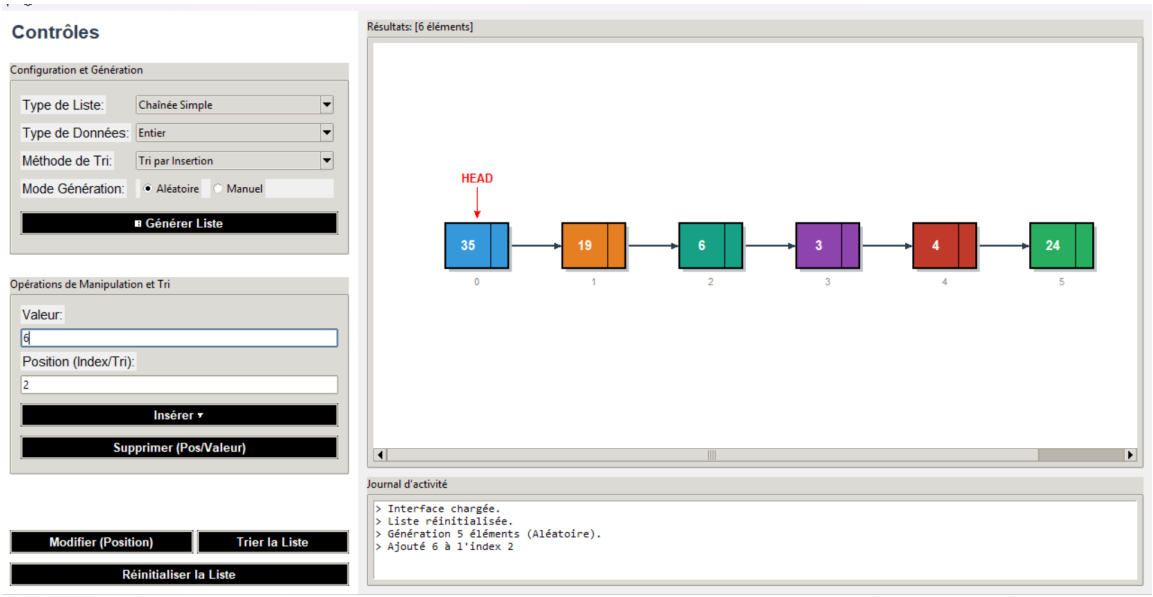

FIGURE 3.1 – Interface du module de listes chaînées

3.2 Structure de Données

```python
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None
        self.prev = None  # Pour Liste Double
```

Listing 3.1 – Classe Node en Python

```c
typedef struct Node {
    void *data;
    struct Node *next;
    struct Node *prev;  // Pour liste double
} Node;
```

Listing 3.2 – Structure Node en C

3.3 Opérations Disponibles

Opérations sur les Listes

- **Insertion** : Au début, à la fin, ou à une position spécifique
- **Suppression** : Par position ou par valeur
- **Modification** : Changer la valeur d'un nœud
- **Tri** : Bubble, Insertion, Shell, Quick Sort
- **Génération** : Aléatoire ou manuelle

3.4 Animation de la Construction

Le module propose une animation nœud par nœud lors de la génération de la liste, permettant de visualiser progressivement la construction de la structure.

```c
static gboolean generation_tick(gpointer user_data) {
    if (gen_state.current_count >= gen_state.target_count) {
        gen_state.timer_id = 0;
        log_msg("Generation terminee.");
        return G_SOURCE_REMOVE;
    }

    // Ajouter un noeud
    if (current_dtype == TYPE_INT) {
        int *v = malloc(sizeof(int));
        *v = rand() % 100;
        append_node(v);
    }
    gen_state.current_count++;
    gtk_widget_queue_draw(drawing_area);
    return G_SOURCE_CONTINUE;
}
```

Listing 3.3 – Animation de génération en C

Chapitre 4

TP3 : Traitement des Arbres

4.1 Présentation du Module

Le module d'arbres permet d'explorer deux types de structures arborescentes :
— **Arbres Binaires** : Maximum 2 enfants par nœud
— **Arbres N-aires** : Plusieurs enfants par nœud

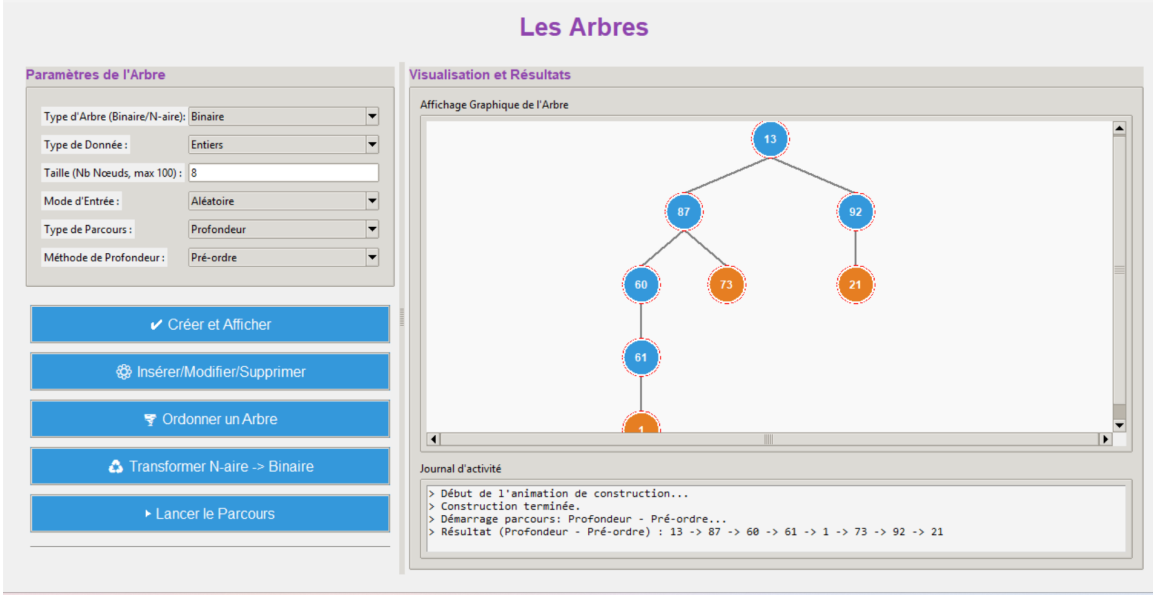

FIGURE 4.1 – Interface du module des arbres avec parcours animé

4.2 Structure de Données

```python
class TreeNode:
    def __init__(self, value):
        self.value = value
        self.children = []  # Liste de TreeNode
        self.x = 0  # Position pour affichage
        self.y = 0
```

Listing 4.1 – Classe TreeNode en Python

```c
#define MAX_CHILDREN 10

typedef struct TNode {
    void *data;
    struct TNode *children[MAX_CHILDREN];
    int child_count;
    double x, y;       // Position pour layout
    int index;         // Pour animation
    int anim_state;    // 0=Normal, 1=Visite, 2=Actif
} TNode;
```

Listing 4.2 – Structure TNode en C

4.3 Parcours Disponibles

4.3.1 Parcours en Profondeur (DFS)

Types de Parcours en Profondeur

- **Pré-ordre** : Racine → Gauche → Droite
- **Ordre (In-ordre)** : Gauche → Racine → Droite
- **Post-ordre** : Gauche → Droite → Racine

```python
def dfs(n):
    if method == "Pre-ordre": path.append(n)

    if method == "Ordre":
        if n.children:
            dfs(n.children[0])
            path.append(n)
            for c in n.children[1:]: dfs(c)
        else:
            path.append(n)

    if method == "Poste fixe":
        for c in n.children: dfs(c)
        path.append(n)

    if method == "Pre-ordre":
        for c in n.children: dfs(c)
```

Listing 4.3 – Parcours DFS avec animation

4.3.2 Parcours en Largeur (BFS)

```c
static void collect_bfs(TNode *root_node, TraversalAnim *anim) {
    if (!root_node) return;
    TNode *q[500];
```

```
    int f = 0, b = 0;
    q[b++] = root_node;

    while (f < b && anim->count < 500) {
        TNode *curr = q[f++];
        anim->path[anim->count++] = curr;
        for (int i = 0; i < curr->child_count; i++)
            q[b++] = curr->children[i];
    }
}
```

Listing 4.4 – Parcours BFS en C

4.4 Transformation BST

Le module permet de réorganiser un arbre en **Arbre Binaire de Recherche (BST)**où chaque nœud gauche est inférieur à la racine et chaque nœud droit est supérieur.

```
def build_bst(vals):
    if not vals: return None

    mid = len(vals) // 2
    node = TreeNode(vals[mid])

    left_child = build_bst(vals[:mid])
    right_child = build_bst(vals[mid+1:])

    if left_child: node.children.append(left_child)
    if right_child: node.children.append(right_child)

    return node
```

Listing 4.5 – Construction d'un BST équilibré

Chapitre 5

TP4 : Exploration des Graphes

5.1 Présentation du Module

Le module de graphes permet de créer et manipuler des graphes avec :
— **Graphes Orientés (GO)** : Arêtes avec direction
— **Graphes Non Orientés (GNO)** : Arêtes bidirectionnelles

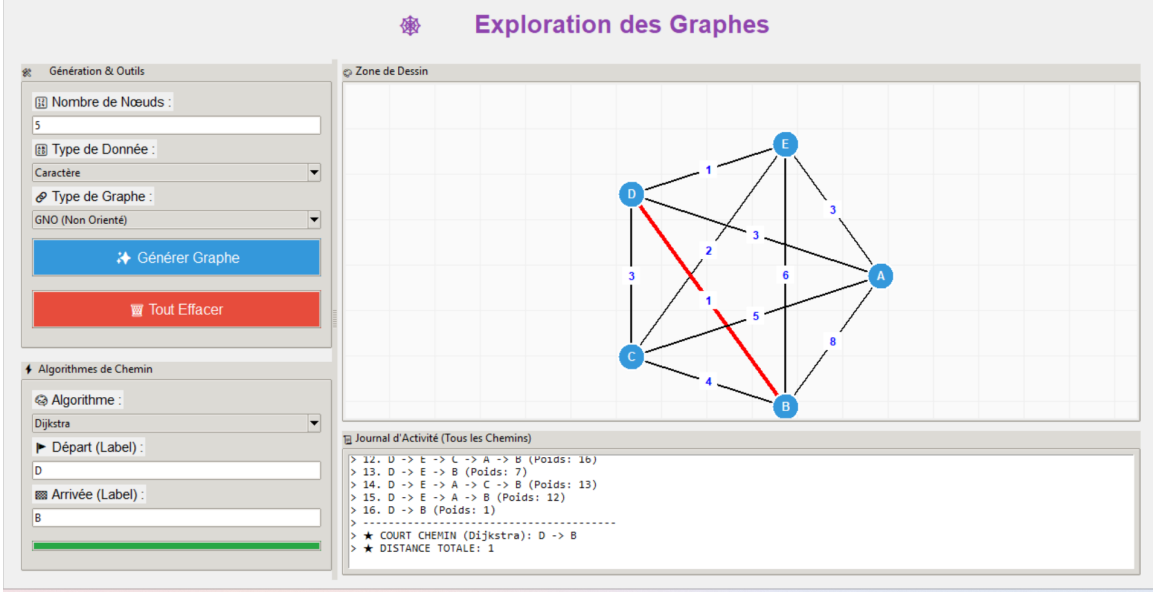

FIGURE 5.1 – Interface du module de graphes avec visualisation du chemin

5.2 Structure de Données

```c
typedef struct Node {
    double x, y;
    char label[16];
    int id;
} Node;

typedef struct Edge {
    int u, v;       // Indices des noeuds
```

```c
    int weight;     // Poids de l'arete
} Edge;
```

Listing 5.1 – Structures pour les graphes en C

5.3 Algorithmes de Plus Court Chemin

5.3.1 Algorithme de Dijkstra

Dijkstra - Complexité : O(V² ou V log V)

Trouve le plus court chemin depuis un nœud source vers tous les autres nœuds. Fonctionne uniquement avec des poids positifs.

```c
static void run_dijkstra(int start, int end) {
    int dist[MAX_NODES], prev[MAX_NODES], visited[MAX_NODES];

    for (int i = 0; i < node_count; i++) {
        dist[i] = INF;
        prev[i] = -1;
        visited[i] = 0;
    }
    dist[start] = 0;

    for (int i = 0; i < node_count; i++) {
        int u = -1, min_d = INF;
        for (int j = 0; j < node_count; j++) {
            if (!visited[j] && dist[j] < min_d) {
                min_d = dist[j];
                u = j;
            }
        }
        if (u == -1 || dist[u] == INF) break;
        visited[u] = 1;
        // Relaxation des voisins
        for (int k = 0; k < edge_count; k++) {
            if (edges[k].u == u) {
                int v = edges[k].v;
                int alt = dist[u] + edges[k].weight;
                if (alt < dist[v]) {
                    dist[v] = alt;
                    prev[v] = u;
                }
            }
        }
    }
}
```

Listing 5.2 – Algorithme de Dijkstra en C

5.3.2 Algorithme de Bellman-Ford

Bellman-Ford - Complexité : O(VE)

Trouve le plus court chemin même avec des poids négatifs. Détecte les cycles négatifs.

5.3.3 Algorithme de Floyd-Warshall

Floyd-Warshall - Complexité : $O(V^3)$

Calcule les plus courts chemins entre toutes les paires de nœuds. Produit une matrice de distances.

```
static void run_floyd() {
    int mat[MAX_NODES][MAX_NODES];
    for (int i = 0; i < node_count; i++)
        for (int j = 0; j < node_count; j++)
            mat[i][j] = (i == j) ? 0 : INF;

    for (int i = 0; i < edge_count; i++)
        mat[edges[i].u][edges[i].v] = edges[i].weight;

    for (int k = 0; k < node_count; k++)
        for (int i = 0; i < node_count; i++)
            for (int j = 0; j < node_count; j++)
                if (mat[i][k] + mat[k][j] < mat[i][j])
                    mat[i][j] = mat[i][k] + mat[k][j];
}
```

Listing 5.3 – Floyd-Warshall en C

5.4 Recherche de Tous les Chemins

Le module affiche également tous les chemins possibles entre deux nœuds avec leurs poids respectifs dans le journal d'activité.

Chapitre 6

Architecture et Technologies

6.1 Version Python

6.1.1 Technologies Utilisées

Composant	Technologie	Description
Langage	Python 3	Langage de programmation principal
GUI	Tkinter	Bibliothèque graphique native
Graphiques	Canvas	Dessin 2D personnalisé
Style	ttk + custom	Thèmes personnalisés

TABLE 6.1 – Technologies de la version Python

6.1.2 Structure des Fichiers Python

```
Version py/
|-- main.py          # Point d'entree - Menu principal
|-- table.py         # Module de tri des tableaux
|-- list.py          # Module des listes chainees
|-- tree.py          # Module des arbres
|-- graph.py         # Module des graphes
|-- styles.py        # Styles et themes
```

Listing 6.1 – Structure du projet Python

6.2 Version C/GTK

6.2.1 Technologies Utilisées

Composant	Technologie	Description
Langage	C	Performance native
GUI	GTK 4	Bibliothèque graphique moderne
Graphiques	Cairo	Dessin vectoriel 2D
Style	CSS GTK	Styles intégrés

TABLE 6.2 – Technologies de la version C

6.2.2 Structure des Fichiers C

```
Version_C/
|-- main.c          # Point d'entree - Initialisation GTK
|-- app.h           # Declarations et styles CSS
|-- table.c         # Module de tri (sorting_view)
|-- list.c          # Module des listes chainees
|-- tree.c          # Module des arbres
|-- graph.c         # Module des graphes
|-- style.c         # Menu principal (menu_view)
```

Listing 6.2 – Structure du projet C

6.3 Comparaison des Deux Versions

Critère	Python	C/GTK
Performance	Modérée	Excellente
Facilité de développement	Élevée	Modérée
Portabilité	Excellente	Bonne (recompilation)
Animations	Fluides	Très fluides
Taille de l'exécutable	Nécessite Python	Standalone

TABLE 6.3 – Comparaison Python vs C/GTK

Chapitre 7

Conclusion et Perspectives

Bilan : Ce projet a permis de développer une application complète de visualisation des structures de données avec **4 modules fonctionnels** (Tri, Listes, Arbres, Graphes) en **double implémentation** Python et C, offrant des animations interactives et un support multi-types.

Difficultés rencontrées :
— Layouts dynamiques pour arbres et graphes
— Synchronisation des animations
— Migration GTK 3 → GTK 4
— Gestion mémoire en C

Perspectives :
— Ajout d'algorithmes : Merge Sort, Heap Sort, A*
— Export PDF/image des résultats
— Mode tutoriel pas-à-pas
— Version web (WebAssembly / Pyodide)

Apports pédagogiques : Compréhension visuelle des algorithmes, comparaison Python/C, maîtrise des GUI (Tkinter, GTK 4), programmation orientée objet et procédurale.

Fin du Rapport