./run 54 5584 84 848 4484 …

MAIN {

strjoin

BLOC : ERROR / PARSING

- check doublons : ft_???
    - Check si deux fois (ou plus) le même paramètre est entré
- ft_split
- ft_atoi_strict :
    - Transforme tous nos arguments en int
    - Check les dernières erreurs possibles
        - INT_MIN / INT_MAX
        - Caractère autre que numérique
- if *error == 1
    - printf (“Error”);
    - **FIN PROGRAMME (**return (0);**)**

BLOC : STRUCTURE

- remplissage stack_a :
    - ft_stack_new
    - ft_stackadd_back
- malloc array copie stack_a / indexation :
    - ft_value_to_index
        - tableau temporaire
        - sorting
        - enregistrement des index
        - retourner les index

BLOC : CHECK FLAGS - ALGO

- Si flags “--adaptive” ou rien :
    - ft_compute_disorder : Voir le %age de désordre
    - Choix algorithme :
        - Simple (19%)
        - Medium (20-49%)
        - Complex (+50%)
    - Faire tourner le bon algorithme
- Sinon :
    - Faire tourner l’algorithme choisi par le flag :
        - “--simple”
        - “--medium”
        - “--complex”

RETOUR : LISTE DES OPÉRATIONS POUR TRIER LA STACK_A

FIN PROGRAMME (return (0);**)**

}

ERREURS DE NORME À CORRIGER

- ft_medium_algo.c
    - sort_chunks
- ft_complex_algo.c
    - find_target_node
- ft_push_swap.c
    - launch_algo
- main.c
    - join_split

NE PAS OUBLIER LE README
CHECKER LES EVENTUELS LEAKS

ft_update_pos(a) : Indexation des index **> Done Matt**
ft_find_target_node **> Done Matt**
ft_calculate_cost(a, b) : Calcule de chaque cost de B pour arriver à b->head **> Done Cel**
best = ft_find_best_path(a) : Savoir s'il faut ra / rra pour arriver a target_node **> Done Cel**
ft_move_b_to_a(best, a, b) : ra/rra pour que b->head soit ok dans A
ft_min_a_to_top(a) : Rotate A dans l'ordre **> Done Cel**