Profiling

חלק זה מטרתו להשוות 3 גרסאות של חישוב שטח ה-Convex Hull , באמצעות מבני נתונים שונים ← list, vector , deque.

עיקר החלק הזה הינו שאנו עושים שימוש ב-**Profiling** , שזהו תהליך שבו אנחנו מודדים ביצועים בפועל של התוכנית שלנו, זאת על מנת לבדוק כמה זמן לוקח לכל פונקציה לרוץ, איפה מתבזבז הזמן, וכמה פעמים נקראות פונקציות מסוימות.

מטרתנו לגלות איזו מ-3 הגרסאות שלנו רצה בזמן הכי מהיר עבור קלט נתון.

מימוש שטח ה-Convex Hull דרך vector מימשנו בחלק 1, אז לכן לא נרחיב עליו.

במימוש של השטח דרך ה**deque** הינו דומה מאוד כמו שאצל **vector** , יש כמה הבדלים קטנים :

אם זה באופן ההוספה וההסרה, שמבוצע דרך push_back, pop_back :

```
//Lower shell:
for(int i=0;i<n;i++){
    while(hull.size()>=2 && cross(hull[hull.size()-2],hull[hull.size()-1],pts[i])<=0){
        hull.pop_back();
    }
    hull.push_back(pts[i]);
}

//Upper shell:
size_t lower_size = hull.size() ; //t is the size of the lower hull
for (int i = n-2; i >= 0; --i) {
    while (hull.size()> lower_size && cross(hull[hull.size()-2],hull[hull.size()-1], pts[i]) <= 0){
       hull.pop_back();
    }
    hull.push_back(pts[i]);
}

hull.pop_back(); //The first snd the last point returns itself=>remove one of them.
```

ובאופן השמירה הדינאמית של אורך המעטפת:

בdeque לא צריך אינדקס שלשהו , לעומת vector שמשתמשים במשתנה k כדי לנהל את האינדקס .

במימוש של השטח דרך ה**list** , ישנו שימוש ברשימה מקושרת דו-כיוונית, היא מאפשרת גישה קלה לאיבר האחרון ולקודמו, זאת דרך prev :

```
while(hull.size()>=2)
{
    auto last = prev(hull.end());
    auto second_last = prev(last);
```

אין אפשרות כמו בdeque ובvector לקבל גישה באינדקס.

כעת, נרצה לבצע הרצה מקיפה של 3 הגרסאות עבור השטח של הקמור, זאת כדי להבין מי יותר יעיל מבחינת זמן ריצה.

נכין דוגמא של רשימת נקודות מוגדרת מראש:

```
vector<pair<float,float>> points = {
    {0, 0}, {0, 1}, {1, 2}, {2, 2}, {3, 1},
    {3, 0}, {2, -1}, {1, -1}, {0.5, 0.5}, {1.5, 1.5}
};
```

נריץ את 3 הפונקציות שלנו , אשר כל פונקציה מחשבת שטח לפי סוג מבנה הנתונים הספציפי שהגדרנו :

ונבצע מדידת זמן הריצה של כל אחד מהפונקציות , נבצע הדפסה של השטח שחושב + הזמן שלקח לחשב אותו:

```
// Vector version
auto start1 = high_resolution_clock::now();
float area1 = CHArea_vec(points);
auto end1 = high_resolution_clock::now();
auto dur1 = duration_cast<nanoseconds>(end1 - start1).count();

// Deque version
auto start2 = high_resolution_clock::now();
float area2 = CHArea_deque(points);
auto end2 = high_resolution_clock::now();
auto dur2 = duration_cast<nanoseconds>(end2 - start2).count();

// List version
auto start3 = high_resolution_clock::now();
float area3 = CHArea_list(points);
auto end3 = high_resolution_clock::now();
auto dur3 = duration_cast<nanoseconds>(end3 - start3).count();
```

כעת נריץ ונקבל כך:

```
roy3177@LAPTOP-QCUJB7RP:~/Convex Hull server/part2-Profiling$ ./main
Points:
(0,0) (0,1) (1,2) (2,2) (3,1) (3,0) (2,-1) (1,-1) (0.5,0.5) (1.5,1.5)

Area (vector): 7 | Time: 5807 ns
Area (deque):  7 | Time: 4320 ns
Area (list):   7 | Time: 4487 ns
roy3177@LAPTOP-QCUJB7RP:~/Convex Hull server/part2-Profiling$ make clean
rm -f main *.o *.gcno *.gcda *.gcov
roy3177@LAPTOP-QCUJB7RP:~/Convex Hull server/part2-Profiling$ make
g++ -std=c++17 -Wall -Wextra main.cpp CH_vec.cpp CH_deque.cpp CH_list.cpp -o main
./roy3177@LAPTOP-QCUJB7RP:~/Convex Hull server/part2-Profiling$ ./main
Points:
(0,0) (0,1) (1,2) (2,2) (3,1) (3,0) (2,-1) (1,-1) (0.5,0.5) (1.5,1.5)

Area (vector): 7 | Time: 12719 ns
Area (deque):  7 | Time: 10135 ns
Area (list):   7 | Time: 11300 ns
```

ביצענו 2 הרצות, ואפשר לראות את תוצאותיה של ה-profiling בזמן הריצה של 3 הגרסאות של פונקציית השטח שך הקמור על קבוצת נקודות זהה.

אפשר לראות כי ב-2 ההרצות תמיד לvector לוקח הכי הרבה זמן , וכי ה-deque בזמן הכי קצר שיש.