* Kod ne işe yarıdığı ve nasıl çalıştığı:

İlk olarak generate, Function1, Function2 ve print1 fonksiyonları tek boyutlu diziler üzerinde çalışıyor:

Generate: bir diziyi ve boyutunu girdi olarak alır ve diziyi -MAX\_W ile MAX\_W arasında rasgele tamsayılarla doldurur.

Function1: tek boyutlu bir dizinin elemanlarını bubble sort algoritması kullanarak artan düzende sıralıyor.

Function2: bir diziyi ve onun boyutunu girdi olarak alır ve dizideki ardışık tamsayıların maksimum toplamının ortalamasını hesaplar.

Print1: bir diziyi ve boyutunu girdi olarak alır ve dizi öğelerini konsola yazdırır.

Sonra Function3, print2 ve print3 fonksiyonları iki boyutlu dizilerin üzerinde çalışıyor:

Function3: girdi olarak G\_SIZE tarafından G\_SIZE boyutunda iki tane iki boyutlu dizi alır ve ilk dizideki değerleri kopyalar, sonra 0 olan öğeleri sonsuza kadar ayarlayarak yeni bir 2B dizi oluşturur. Daha sonra, herhangi iki düğüm arasındaki en kısa yolu bulmak için yeni dizide Floyd-Warshall algoritmasını gerçekleştirir.

Print2: iki boyutlu diziyi ve boyutunu girdi olarak alır ve matrisi konsola yazdırır.

Print3: iki boyutlu diziyi, boyutunu ve bir tamsayıyı girdi olarak alır ve Floyd-Warshall algoritmasını kullanarak matristeki iki düğüm arasındaki en kısa yolu yazdırır,

(tamsayıdan az sayıların pozisyonlarını harf olarak yazdırır)

* Kodun zaman karmaşıklığı:

Generate fonksiyonu:

Function1:

Function2:

Function3:

Print1:

Print2:

Print3:

Tüm Kodun zaman Karmaşıklığı:

* Kod nasıl geliştirebileceğiz?

İlk olarak Function1 Bubble sort algoritmasını kullanıyor, zaman karmaşıklığı O(N^2).

onun yerinde Quick Sort algoritması kullanırsak, daha verimli olur.

void quickSort(int arr[], int left, int right) {

    int i = left, j = right;

    int pivot = arr[(left + right) / 2];

    while (i <= j) {

        while (arr[i] < pivot) {

            i++;

        }

        while (arr[j] > pivot) {

            j--;

        }

        if (i <= j) {

            // swap(arr[i], arr[j]);

            int tmp = arr[i];

            arr[i] = arr[j];

            arr[j] = tmp;

            i++;

            j--;

        }

    }

    if (left < j) {

        quickSort(arr, left, j);

    }

    if (i < right) {

        quickSort(arr, i, right);

    }

}

Sonra function2 yerine Kadane algoritmasi kullanabiliriz:

int function2(int a[], int size) {

    int max\_so\_far = 0, max\_ending\_here = 0, count = 0;

    for (int i = 0; i < size; i++) {

        max\_ending\_here += a[i];

        if (max\_ending\_here < 0) {

            max\_ending\_here = 0;

        }

        if (max\_so\_far < max\_ending\_here) {

            max\_so\_far = max\_ending\_here;

            count++;

        }

    }

    return max\_so\_far/count;

}

Böylece zaman karmaşıklığında büyük bir değişim olmaz, ancak daha verimli olur.

* Bu değişiklerle beraber yeni zaman karmaşıklığı:

Function1(QuickSort):

burada: a = 2, b = 2, d = 0. Ortalama durumunda => O(N\*log(N)).

En kötü durumda, pivotun dizinin en küçük veya en büyük öğesi olarak seçildiği ve bölümlemenin her zaman dengesiz alt dizilerle sonuçlandığı durumlarda, QuickSort algoritmasının zaman karmaşıklığı O(n^2) olabilir. Bu, dizi zaten sıralandığında veya neredeyse sıralandığında oluşur, bu da her bölümlemenin dizi boyutunu yalnızca 1 azaltmasına neden olur.

Function2:

Yeni Kodun zaman Karmaşıklığı:

* Çalışma Süresi:

İki kodun arasında çalışma süresinin farkı azdır.

Eski kodu çalıştırdığımda:

Calişma Suresi: 0.000265 sn

Yeni Kod:

Çalışma Süresi: 0.000221 sn