Veri Yapıları 1. Ödev

Kullanılan Kütüphaneler:

```
#include <iostream>
```

Ekrana yazı yazdırmak için,

```
#include <fstream>
```

Dosya okuma işlemleri için

```
#include <vector>
```

Okunan verileri parse edebilmek için (resim 2.1)

Kullanılan Sınıflar:
DoublyLinkedList ve Node

İlk olarak txt dosyası okutularak satır sayısı alınır. Bunun için mainde kalabalık olmaması için metod oluşturulmuştur. Satır sayısı alındıktan sonra txt dosyası tekrardan açılarak her satır için bir DoublyLinkedList oluşturulur. Oluşturulan listeler DoublyLinkedList tanımlı dinamik bir dizide tutulur.

Dairesel Bağlı Liste nasıl oluşturuluyor?
txt dosyasındaki satır bir bütün olarak sınıfa düşer. Ardından <vector> kütüphanesinden faydalanılarak string parse edilir.(resim 2.1)

```
vector<string> DoublyLinkedList::split(string target, string delim)
{
    vector<string> v;
    if (!target.empty())
    {
        string::size_type start = 0;
        do
        {
            size_t x = target.find(delim, start);
            if (x == string::npos)
                break;

            v.push_back(target.substr(start, x - start));
            start = x + delim.size();
        } while (true);

        v.push_back(target.substr(start));
    }
    return v;
```

2.1

Bu sayede hem dizi boyutuna ulaşmamız kolaylaşıyor, hemde parse etmemiz kolaylaşıyor.
Ekleme işlemi aşağıdaki kod bloğunda olmaktadır.

```
mid = new Node(temp, NULL, NULL); //ilk node ekleniyor
vector<string> right;
vector<string> left;
halfTotal = splitted.size() / 2;
for (int i = 1; i <= splitted.size() / 2; i++)
{
    left.push_back(splitted[i]);
    right.push_back(splitted[splitted.size() - i]);
}
reverse(right.begin(), right.end());
string *leftarr = new string[left.size()];
string *rightarr = new string[right.size()];
for (int i = 0; i < left.size(); i++)
    leftarr[i] = left[i];
for (int i = 0; i < right.size(); i++)
    rightarr[i] = right[i];
addleft(leftarr, left.size(), mid);
addright(rightarr, right.size(), mid);
```

Hemen ardından ödevde istenilen formata uymak için vector listesi arraye çevriliyor.
Mid Node önce oluşturulup önce sol, sonra sağ taraf eklenir ve dairesel liste elde edilir.
Dairesel liste oluşturulurken önce sol tarafın en sondaki elemanı mid ile döngüye bağlanır.
Ardından sol tarafın son elemanı sağ tarafa ekleme metodunda yakalanarak, sağ taraf ekleme işlemi bittiğinde dairesel liste doğru formata getirilir.

--Çaprazlama nasıl yapılır?
Çaprazlanacak listeler bulunduktan sonra swapper adındaki metod kullanılır.

```
void swapper(DoublyLinkedList *min, DoublyLinkedList *max)
{
    Node *minendLeft = min->mid->prev;
    Node *minstartLeft = min->mid;

    Node *minstartRight = min->mid->next;
    Node *minendRight = min->mid;
    for (int i = 0; i < min->halfTotal; i++)
    {
        minstartLeft = minstartLeft->prev;
        minendRight = minendRight->next;
    }

    //-----------------------------------
    Node *maxendLeft = max->mid->prev;
    Node *maxstartLeft = max->mid;

    Node *maxstartRight = max->mid->next;
    Node *maxendRight = max->mid;
    for (int i = 0; i < max->halfTotal; i++)
    {
        maxstartLeft = maxstartLeft->prev;
        maxendRight = maxendRight->next;
    }
```

txt dosyası okunurken sağ ve sol tarafta ne kadar üye olacağını biliyorduk. Bu veri DoublyLinked sınıfından halftotal değişkeni içerisinde tutuluyor. Bu sayede ilk olarak hangi kısımların çaprazlanacağını elde ediyoruz. Ardından ise sol ve sağ tarafın uçları reverse işlemini gerçekleştirebilmek için NULL'a çekiliyor.

```
minstartLeft->prev = NULL;
minendLeft->next = NULL;
minendRight->next = NULL;
minstartRight->prev = NULL;

maxstartLeft->prev = NULL;
maxendLeft->next = NULL;
maxendRight->next = NULL;
maxstartRight->prev = NULL;
```

Hemen ardından çaprazlanacak kollar reverse ediliyor.

```
void reverse(Node **head_ref)
{
    Node *temp = NULL;
    Node *current = *head_ref;
    while (current != NULL)
    {
        temp = current->prev;
        current->prev = current->next;
        current->next = temp;
        current = current->prev;
    }

    if (temp != NULL)
        *head_ref = temp->prev;
}
```

Reverse işlemi bittikten sonra kollar yer değiştirilerek dairesel liste yapısı tekrardan oluşturuluyor. Ekrana yazdırma işlemlerinden sonra heap temizlenerek programdan çıkış yapılıyor.

```
//free memory
for (int i = 0; i < line_count; i++)
{
    Node *next = dyn_arr[i]->mid;

    for (int j = 0; j <= dyn_arr[i]->halfTotal * 2; j++)
    {
        Node *node = next;
        next = node->next;
        node->~Node();
    }
    free(dyn_arr[i]);
    free(next);
}
```

Ekran çıktısı:

```
Microsoft Windows [Version 10.0.19041.630]
(c) 2020 Microsoft Corporation. Tüm hakları saklıdır.

C:\Users\sbuys\Desktop\veri_odev_satis_1_hotfix1\veri_odev_1_hotfix2>mingw32-make
g++ -I "./include" -c ./src/Node.cpp -o ./lib/Node.o
g++ -I "./include" -c ./src/DoublyLinkedList.cpp -o ./lib/DoublyLinkedList.o
g++ -I "./include" -c ./src/main.cpp -o ./lib/main.o
g++ ./lib/Node.o ./lib/DoublyLinkedList.o ./lib/main.o -o ./bin/cikti -std=c++0x
./bin/cikti.exe
En Buyuk Liste Orta Dugum Adres: 0x102aab0
En Buyuk Liste Degerler
1 6 19 45 54 13 5 52 102

En Kucuk Liste Orta Dugum Adres: 0x102aab0
En Kucuk Liste Degerler
2 90 80 17 20 11 73
C:\Users\sbuys\Desktop\veri_odev_satis_1_hotfix1\veri_odev_1_hotfix2>
```