MID TERM PROJECT REPORT

Movie Guide

Submitted By :

Mukarram Ali

2021-CS-58

Rayan Rasheed

2021-CS-59

Submitted To :

Mr. Samyan Qayyum Wahla

DEPARTMENT OF COMPUTER SCIENCE

UNIVERSITY OF ENGINEERING AND TECHNOLOGY

Contents

1 Project Overview

1.1 Description

As obvious from project title “Movie guide” it will contain movies from movies industry all over the world i.e., Hollywood, Lollywood, Bollywood, Japanese many more. This movie guide contains almost 8 features with movie name and date of releasing. This guide will help the watcher to find out the right movie from guide by filtering it through date, releasing date or genre that will make it perfect guide and eventually reaching out at right movie. Moreover, this guide consists of versatile search with casting, genre and country. Moreover, this guide contain feature to sort the movies with respect to releasing date, duration or IMDB rating. And guide's user has an option to choose sorting algorithm that make this guide more reliable. In this versatile guide, progress bar along with start, end and pause button will be available to start, end or stop the movie's data scraping from the website movies counter and IMDB will be used for data scrapping and user will paste the URL in text box to scrap data from that site. Scraping the movie data according to genre make it more unique. The guide will contain sorting algorithm like insertion sort, merge sort, bubble sort, quick sort, pigeon hole sort etc.
Moreover, to search movie with name, genre or date certain searching algorithm will used such as binary search and KMP Searching algorithm. This guide will the end-level user to reach out at required movie and eventually it will help movie's producer to produce movies of that genre depending upon ranked by users.

1.2 Motivation

Motivation was to facilitates movies producer to invest on that movies genre that are voted by public and earn the most. Aim was also to facilitate public to reach at the movie of their favourite actor along with filter of year, director and genre to reach exactly at the required movie.

1.3 Target Audience

This project will help Movie Producers, Directors, Artist, General Public

1.4 Business Need

On internet there are various website for movies but there is not any guide of that type that help the watchers to find out exact movie and also help the movie producer to make a movie that will attract the viewers. Before this guide producers and directors have to go through various step to predict which genre of movies are mostly voted by viewers. This guide will help producers to know which type of movies viewers exactly likes and which movies earn the most and which movies earn least on a single click. They can also see the movies list with respect to votes from public in sorted manner to exactly analyse statistic.
It will also help public reach to the movie according to their interest in term of genre, year,

casting actor, etc. Like some people want movie of his favourite actor or director, primitively he has to go through various step to find out that actor movie after lot of browsing but this guide will help him to search that on a single click.
For Example, if a person wants to watch movie of christen Bale of crime genre. He can exactly reach at that movie. This application will work as a guide for public and movie producer to exactly know the current scenario maintained in guide.

2 Srapped Data

2.1 Attributes

The movies which are scraped have some attributes which are necessary for our data frame. The attributes are of the following:

2.1.1 Movie Name

This includes the name of the movie which depicts the main theme of the whole movie.

2.1.2 Duration

This tells us the Duration time of movie in minutes

2.1.3 Genre

This tells us the Genre of movie i.e., crime, action, romantic, etc.

2.1.4 Rating

This tells us the Rating of movie

2.1.5 Year

This tells us the Movie releasing year

2.1.6 Director

This tells us the Director of movie

2.1.7 Actor

This tells us the Actor of that movie

2.1.8 Gross

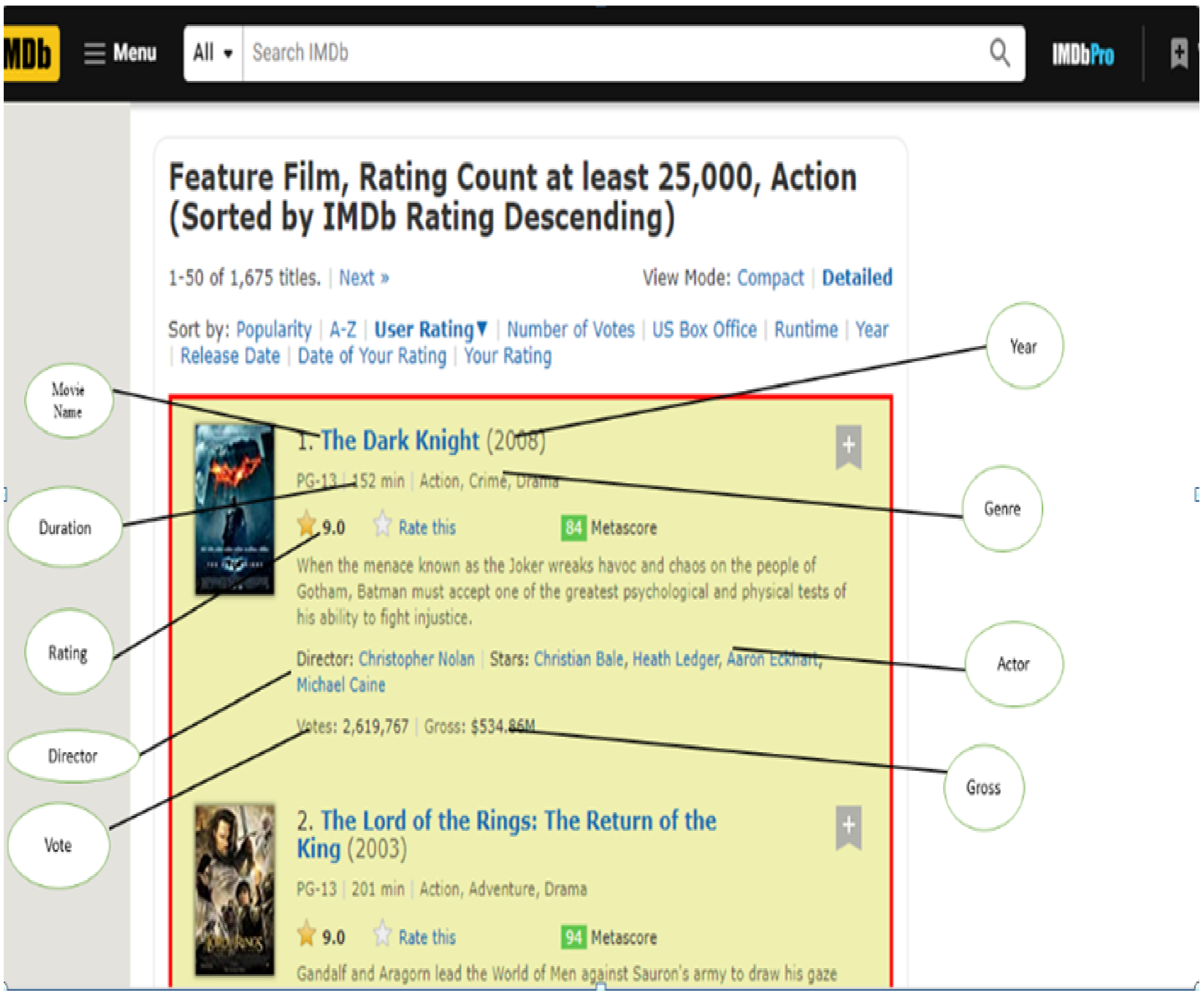
This tells us the Gross income in million

2.1.9 Votes

This tells us the Total votes by public

2.2 Sample Scrapping Source

We used IMBD as a scraping source



Feature Film, Rating Count at least 25,000, Action (Sorted by IMDb Rating Descending)

1-50 of 1,675 titles. | Next »

View Mode: Compact | Detailed

Sort by: Popularity | A-Z | User Rating▼ | Number of Votes | US Box Office | Runtime | Year | Release Date | Date of Your Rating | Your Rating

1. The Dark Knight (2008)

PG-13 | 152 min | Action, Crime, Drama

9.0 Rate this 84 Metascore

When the menace known as the Joker wreaks havoc and chaos on the people of Gotham, Batman must accept one of the greatest psychological and physical tests of his ability to fight injustice.

Director: Christopher Nolan | Stars: Christian Bale, Heath Ledger, Aaron Eckhart, Michael Caine

Votes: 2,619,767 | Gross: $534.86M

2. The Lord of the Rings: The Return of the King (2003)

PG-13 | 201 min | Action, Adventure, Drama

9.0 Rate this 94 Metascore

Gandalf and Aragorn lead the World of Men against Sauron's army to draw his gaze

Figure 1: Sample Scraping Source. Screenshot taken from IMBD.com

3 Alorithms

3.1 Sorting Algorithms

[HTML]9B9B9B **Algorithm Name: Insertion Sort**

Description

Sorting algorithm with $O(n^2)$ time complexity. The elements of the array is compared sequentially with other elements and then placed at their respective position in the left.
It is stable sorting algorithm

Pseudo Code:

```
InsertionSort(A)
for i = 0 to A.Length
pick = A[i]
j = i - 1
while j >= 0 and A[j] >pick:
A[j+1] = A[j]
j = j - 1
A[j+1] = pick
```

Code in Python:

```
def Insertionsort(arr,attr,start,end):
for i in range(start, len(arr)):
pick = arr[i]
j = i - 1
while j >= 0 and arr[j] >pick:
arr[j + 1] = arr[j]
j -= 1
arr[j + 1] = pick
return arr
```

Time complexity analysis

```
for i = 0 to A.Length ——————————>n-times
pick = A[i] ——————————>n-1 times
j = i - 1 ——————————>n-1 times
while j >= 0 and A[j] >pick: ——————>summation from 0 to n(j)
A[j+1] = A[j] ——————————>summation from 0 to n(j-1)=n(n+1)/2
j = j - 1 ——————————->summation from 0 to n(j-1)=n(n-1)/2
A[j+1] = pick ——————————>n-1 times
```

Calculation: $n+n-1+n-1+n(n+1)/2+n(n-1)/2+n-1=O(n^2)$

Proof of Correctness:

Loop Invariant:

At the start of each iteration of the outer loop the elements of the sub-array A[i..j] are sorted.

Initialization:

Before the start of the loop there is only one element in the sub-array A[i..j] which is trivially sorted

Maintenance:
During the iteration of the loop the element A[j] is placed at its correct position by comparison with the previously sorted element. since the array is previously sorted, we can stop reach an element lower than A[i] at this point A[i] is placed next to A[j].

Termination:
At the end of the loop the iteration is complete from i to n, thus the array is completely sorted

Strengths:
Only requires constant amount of additional memory
efficient for small data set
simple algorithm
In pace sorting algorithm

Weaknesses:
Does Not perform well on large data set
takes n^2 time complexity

Dry Run on small input:
Original Array:

5	11	2	0	8

Dry run: Step#, state of the array

1:	5	11	2	0	8
2:	2	5	11	0	8
3:	0	2	5	11	8
4:	0	2	5	8	11

Sorted Array

0	2	5	8	11

[HTML]9B9B9B **Algorithm Name: Selection Sort**

Description:
Sorting algorithm, with O(n^2) time complexity. This algorithm iteratively selects the highest element from the array and puts it at its correct position. It is unstable sorting algorithm.

Pseudo code:
```
SelectionSort(arr):
for i = 1 to arr.Length:
min = i
for j = i to arr.Length:
if arr[j] <arr[min]
min = j
swap arr[min] and arr[i]
```

Code in Python
```
def SelectionSort(array,attr,start,end):
for i in range(start,end+1):
min=i
for j in range(i,end+1):
if((getattr(array[j],attr))<(getattr(array[min],attr))):
min=j
swap=array[min]
array[min]=array[i]
array[i]=swap
return array
```

Time Complexity Analysis
```
SelectionSort(arr):
for i = 1 to arr.Length: ——————————————>n times
min = i ——————————————>n-1 times
for j = i to arr.Length: ——————————————>n(n+1)/2
if arr[j] <arr[min] ——————————————>n(n+1)/2
min = j
swap arr[min] and arr[i]——————————————->n-1 times
```

Calculation: (n-1)+n(n+1)/2+n-1=O(n^2)

Proof of Correctness:

Loop Invariant:
A[min] is the lower than all elements of the sub-array A[i..j]

Initialization:
Before the start of the inner loop the length of sun-array A[i..j] is 0. So A[min] is

[HTML]FFFFFF lower than,all of its elements (trivially).

Maintenance:
During the execution of the loop there are two cases. either A[min] =<A[j] nothing happensin this case, the second case is if A[min] >A[j], in this case the elements are swapped so the loop invariant

Termination:
After the termination of the loop A[min] is the smallest element in the sub-array A[i..n] since j <= n. We place this element at i. In this way when the outer loop goes from 0 to n, the array gets completely sorted

Strengths:
Is simple to use
Is easy to understand
Is in place
No additional temporary storage required

Weaknesses:
Not useful for larger number of items in array
Takes O(n^2) time complexity
Is unstable

Dry Run on small input:
Original array:

9	5	0	7	15

Dry run: Step#, State of array

1:	0	9	5	7	15
2:	0	5	9	7	15
3:	0	5	7	9	15

Sorted Array:

0	5	7	9	15

[HTML]9B9B9B **Algorithm Name: Merge Sort**

Description:
Sorting algorithm with O (n*lg n) time Complexity. It is done on the principal of Divide and Conquer. In this technique the original swapped so the loop invariant

Pseudo code:

```
MergeSort(A, start, end)
if start >= end
return
else
mid = (end - start)/2
MergeSort(A, start, mid)
MergeSort(A, mid+1, start)
Merge(A, start, mid, end)
Merge(A, start, mid, end)
let L = A[start .. mid] and R = A[mid+1..end]
i = j = k = 0
while i <L.Length and j <R.Length
if L[i] <R[j]
A[k] = L[i]
i = i + 1
else
A[k] = R[j]
j = j + 1
k = k + 1
while i <L.Length
A[k] = L[i]
i = i + 1
k = k + 1
while J <R.Length
A[k] = L[j]
j = j + 1
k = k + 1
```

Code in Python:

```
def MergeSort(Arr,attr,p,r):
if(p<r):
q=m.floor((p+r)/2)
MergeSort(Arr,attr,p, q)
MergeSort(Arr,attr,q+1,r)
Merge(Arr,attr,p,q,r)
return Arr
else:
return Arr
```

```
def Merge(Arr,attr,p,q,r):
n1=q-p+1
n2=r-q
L=[]
R=[]
for i in range(n1):
L.append(Arr[p+i])
for j in range(n2):
R.append(Arr[q+j+1])
attribute=getattr(Arr[0], attr)
if(type(attribute)!=str):
a=copy.deepcopy(Arr[4])
setattr(a, attr, sys.maxsize)
L.append(a)
R.append(a)
else:
a=copy.deepcopy(Arr[4])
setattr(a, attr, "zzzzzzzzzzz")
L.append(a)
R.append(a)
i=0
j=0
for k in range(p,r+1):
if(getattr(L[i],attr)<getattr(R[j],attr)):
Arr[k]=L[i]
i +=1
else:
Arr[k]=R[j]
j +=1
```

Calculation:

The recurrence equation comes out to be: T(n) = 2T(n/2) + O(n) Solving this we get O(n*lgn)

Time Complexity Analysis:

```
Merge(A, start, mid, end) ——————>1 Time
let L = A[start .. mid] and R = A[mid+1..end] ——————>1 time
i = j = k = 0 ——————>1 times
while i <L.Length and j <R.Length ——————>n times
if L[i] <R[j] ——————->n-1 times
A[k] = L[i] ——————->1 times
i = i + 1 ——————->1 times
else ——————->n-1 times
A[k] = R[j]
j = j + 1
```

3.2 Searching Algorithms

3.2.1 KMP Search

4 User Interface

4.1 Initial GUI Design

4.2 GUI Components

4.2.1 Main Screen

4.2.2 Scrapping Screen

4.3 Implemented GUI

4.3.1 Main Window Page

4.3.2 Scrapping Page

4.3.3 Advanced Search Page

5 Integration

5.1 Problem and their solution

5.2 Overview of Whole Working

5.3 Collaboration

5.4 Bonus Tasks