2019-2020 Bahar Yarıyılı BLM2512 Veri Yapıları ve Algoritmalar Dönem Projesi

Mesut Şafak Bilici
17011086

Contents

Çalıştırmadan Önce..

input-3.txt isimli dosyanın okunması, node sayısının bulunması, graph'ın oluşturulması ve BFS algoritmasının uygulanması toplamda 14 dakikayı bulmaktadır.
İstenilen ismin connection map'ini bulmak için ismi "Soyisim, İsim" şeklinde girmeniz gerekmektedir.

Yöntem

İstenilen algoritma farklı bölümlere sahiptir. Öncelikle, verilen veriyi graph'a dökmek için graph'ımızın vertex sayısını bilmemiz gerekmektedir. Verilen film-oyuncu verisi

→ film1/soyad-actor1, isim-actor1/soyad-actor2, isim-actor2
→ film2/soyad-actor2, isim-actor2/ ...

şeklinde verildiği için, stringler üzerinde bir tokenize işlemi yapmamız gerekli. Bu tokenize işlemi ise, her satırda ilk önce '\n' karakterini ayrıştırarak daha sonra ise '/' karakterini ayrıştırarak yapılmalıdır. Bu sayede verimizdeki film ve kaç aktör sayısı tokenize işlemi sayısına eşit olacaktır. Fakat bir aktör birden fazla filmde oynamış olabilir. Bu ise yapacağımız işlemi değiştirecektir. Birden fazla filmde oynayan aktörleri tek bir kişi saymamız için her satırın birleşim kümesini bulmamız gerekmektedir. Bu sayede graph'ımızın vertex sayısını bulabiliriz. Vertex sayımızı bulduktan sonra her aktörü ve filmi bir node haline getirip graph'ımızı oluşturuyoruz. Daha sonra ise Breadth First Search ile her aktör vertex'inin Kevin Bacon'a olan uzaklığını buluyoruz. Bu Breadth First Search işlemini yaparken her vertex'in parent'ını tutmak, istenilen karakterin Kevin Bacon ile arasında olan connection map'i göstermemiz için (izlediği en kısa yol) işimize yarayacaktır. Aşağıda bu yapılan işlemlerin her biri ayrı fonksiyon ve değişken başlıkları altında örnekleri verilerek daha detaylı açıklanacaktır ve bazı oyuncular için çıktılar gösterilecektir.

Algoritma: Değişkenler ve Fonksiyonlar

Bu kısımda her bir işlem için gerekli olan değişkenler ve algoritmalar açıklanacaktır. Toplamda 3 ayrı struct, 13 tane fonksiyon, 2 tane global dizi kullanılmıştır.

Global Değişkenler

Kod boyunca neredeyse her yerde kullanılacak 2 farkı dizi global değişken olarak tanımlanmıştır. Bu dizilerin isimleri ile beraber, yaptığı işler aşağıda açıklanmıştır:

- **char** storage[190000][300]: Bu değişkeni vertex sayımızı hesaplamak için ve aynı zamanda her bir film/aktör vertex'ine belirli bir $i \in \mathbb{N}$ index'ini atamak

için tanımladık. Kullanıldığı fonksiyonlar içinde yaptığı iş daha detaylı şekilde açıklanacaktır.

- **int** int moviectrl[190000]: Vertex sayısını hesaplarken gelen verinin film mi yoksa aktör mü olduğu bilgisini tutmakta belirlenmiştir. Boolean değerler tutan bir dizidir. Eğer film ise 1, aktör ise 1 tutmaktadır. Her bir index storage dizisinin indexine karşılık gelmektedir. Yani storage[k] içindeki film/aktör ismi'nin film mi yoksa aktör mü olduğu bilgisi moviectrl[k] içinde saklanmaktadır.

Fonksiyonlar

Kod boyunca toplamda 13 ayrı fonksiyon kullanılmıştır. Bunlardan 4'ü graph yapısını oluşturmak için, 3'ü Breadth First Search için, 2'si Kevin Bacon frekans dizisini oluşturmak için, 1'i en kısa yolu bulmak için, 2 tanesi önceden hesaplanmış yolu bastırmak için, son 1 tanesi ise opsiyonel olarak graph yapısını bastırmak için.

- **int** numberOfVertex(FILE*): Algoritma, elimizdeki veriyi okuyup tokenize ederek kaç aktör ve kaç film olduğunu bulmaktadır. Veride her film isminden 1 tane olmasına rağmen herhangi bir aktör 2 veya 2'den fazla filmde oynayabilir. Bunu kontrol etmezsek aynı aktöre ait 2 farklı node açmış oluruz. Algoritmada ilk önce satır satır verimizi okumaya başlıyoruz ve ilk önce '\n' karakterini elimizdeki satırdan ayrıştırıyoruz. Sonra ise ayrıştıracak bir string kalmayana kadar '/' karakterini ayrıştırıyoruz. Teker teker bu film-aktör isimlerini storage dizisine yerleştiriyoruz, fakat bunu yaparken en baştan storage dizisini kontrol ediyoruz. Eğer hiçbir elemanı elimize gelen aktör stringine denk değilse bu elemanı storage dizisinin son elemanından bir sonraki elemana atıyoruz. Elimizdeki dosya'nın sonu gelene kadar bunu yapıyoruz. Bu işlem bittikten sonra artık storage dizisi elimizdeki her filmi-aktörü bulunduruyor olacak. İşimiz kolaylaşsın diye, her bir film-aktör'ün vertex numarasını da storage içindeki index adresi belirliyoruz.

- **NODE*** createNode(int,char*): Gelen her elemanı linked list'in başına ekleyen fonksiyondur. Bunu yaparken hangi vertex olduğu ve hangi isimli film-aktör olduğu verilerini de atamaktadır.

- **GRAPH*** createGraph(int): vertex sayısı $\times$ NODE struct'ı boyutunda bir linked list dizisi açmaktadır. Graph yapısı burada tutulacaktır. Bu dizinin ismi adjList[]'dir.

- **GRAPH*** addEdge(int,char*): Bu fonksiyonda hangi aktör hangi film ile bağlantılı olacağını hesaplıyoruz. Bunu yaparken yine dosyayı okuyoruz, string tokenizer ile ayrıştırarak. Bu sefer elimize gelen her yeni stringi storage dizisinin elemanları ile karşılaştırıyoruz. Doğal olarak bu dizinin içinde bir yerlerde

bu film-aktör ismi olmak zorunda. storage isimli dizinin içinde elimize gelen yeni stringi bulduğumuzda bu string'in bulunduğu index'i bulmak istriyoruz aslında. Bundan sonra bu index'e sahip bir node yaratıyoruz $createNode()$ fonksiyonu ile ve bu node'u satır başında hangi film varsa onunla bağlıyoruz (edge oluşturuyoruz). Bu şu anlama gelmekterdir: $\forall\, k \in \mathbb{N}$ için adjList[k]'daki node'lar storage[k]'daki verinin bağlantılarıdır. Adımları daha açıklayıcı göstermek için pseudo-code aşağıdaki gibidir:

Algorithm 1 addEdge

1: Vertex sayısı kadar adjList[] dizisi yarat.
2: $j \leftarrow$ Her satır başındaki filmin storage içindeki index'ini bul.
3: $i \leftarrow$ Text dosyasından gelen her aktörün storage içindeki index'ini bul.
4: i değeri ile node yarat ve adjList[j]'e ekle.
5: j değeri ile node yarat ve adjList[i]'e ekle

Aynı zamanda bu işlemlerin her biri yapılırken file içindeki her satır tokenize edilirken, ilk tokenize edilen cümleyi yani film isminin karşılık geldiği k indexini moviectrl isimli integer/boolean dizisinin k indexini 1 yaptım.
Şöyle açıklarsak daha anlaşılır olacaktır, elimde bir veri var ve bu verinin storage isimli dizideki index'i k olsun. Eğer bu veri herhangi bir satırda ilk tokenize edilen cümle ise filmdir, değilse aktördür. Daha sonra ise yine aynı k indexine karşılık gelecek şekilde moviectrl[k] bölmesi filmse 1, aktörse 0 yapılır. Bu kontrol ilerde Bacon numarası için frekans hesaplaması yaparken işimize yarayacaktır.

- **void** printGraph(GRAPH*,int):

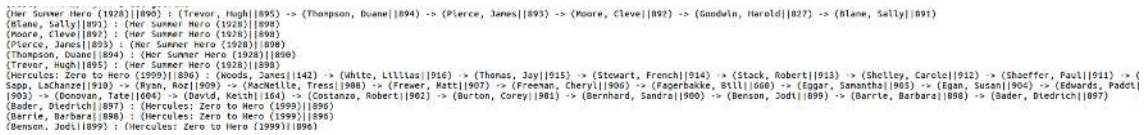

Figure 1: Graph'ın bir kısmının ekran görüntüsü.

Graph'ın bastırılması şu şekilde olmuştur:
storage[i](vertex i): graph->adjList[j](vertex m) / graph- >adjList[j]->next(vertex n)
storage[m](vertex m): graph->adjList[k](vertex i)

storage[n](vertex n): graph->adjList[p](vertex i)

- **void** eunqueue(QUEUE**,int,char*): Queue'ya eleman eklediğimiz fonksiyon. Ek olarak Queue'ya node atarken graph node'unun vertex numarasını ve film-aktör verisini de Queue noduna atıyoruz.
- **QUEUE*** dequeue(QUEUE**): Queue'dan eleman çektiğimiz fonksiyon. Ek olarak Queue boş olduğunda fonksiyon NULL döndürüyor.

- **void** BreadthFirstSearch(GRAPH**,char*,int,int*,int*): Breadth First Search bipartite graph'ta enine arama yapılmak üzere kullanılmıştır. Her node'un (filmler dahil) Kevin Bacon'a olan *en kısa* uzunluğu hesaplanacaktır. Bunu yaparken yukarıda açıklanan enqueue ve dequeue fonksiyonları ile birlikte Queue yapısı kullanılmıştır. Breadth First Search yaparken ilk önce "Bacon, Kevin" node'unun hangi index'te, diğer bir sözle, "Bacon, Kevin" isminin hang storage[] index'inde olduğu hesaplanmıştır.

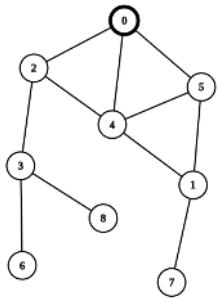

Figure 2: Örnek graf modeli.

Algoritmanın nasıl çalıştığını Figure 2'den ilerleyerek gösterelim. 0 index'li node'un "Kevin Bacon" node'u olduğunu varsayalım (başlangıç node'u Kevin Bacon olmayan senaryolar da olabilir). İlk önce visited isimli Kevin Bacon'a olan uzaklıkları tutan integer dizimizin tüm elemanlarını -1 yapıyoruz. Bu sayede asla ulaşamadığımız node'ların uzaklık değeri -1 oluyor.
Kevin Bacon'dan başlıyoruz ve Kevin Bacon'ı queue yapısına atıyoruz. Daha sonra queue'dan son elemanı çekip komşularına, yani 2 4 5 node'larını, geziyoruz. 2 4 5 node'larının ancak ve ancak film olabileceğini belirtelim. Bu node'ları gezerken sırasıyla 2 4 5 node'larını da queue'ya atıyoruz. Bu node'ların visited sayısı parentlarının visited sayısı oluyor. Bu gezme işlemini yaparken parent isimli arrayin içinde gezdiğimiz node'ların bir üst node'unu yani parentının da indexini tutuyoruz. Bu işlem yolu bastırırken işimize yarayacak.
Sonra queue'ya attığımız ilk elemanı çekiyoruz yani 2 numaralı node'u. Sonrasında komşusu olan 3'ü, 3 bir aktör, gezmiş oluyoruz. Bu işlem bu şekilde

devam ediyor. Komşu kalmadığını anlamamız için dequeue işlemi yapılırken çekecek eleman kalmadığında NULL döndürüyoruz. Artık elimizde parent ve visited dizileri var.

- **void** findMaxDistance(int,int*): Frekans dizimizin bir boyunun olması zorunlu. Histogram mantığı ile bir frekans dizisi oluşturacağımızdan visited dizisinin en büyük elemanı frekans dizimizin eleman sayısı olarak seçiliyor. Aynı zamanda bu işlem yapılmadan önce her visited değerine 2 ile bölünmüş hali atanıyor. Çünkü

  film1 - oyuncu1 - film2 - oyuncu2

  gibi bir arama yapıldığında oyuncu 1 ile oyuncu 2 arasındaki mesafe 2 çıkmaktadır çünkü aradaki filmi de sayıyoruz.

- **int*** frequencyListOfKevinBaconNumbers(GRAPH*,int,int,int*): Eğer moviectrl arrayındaki bilgi 0 ise, yani aktör ise, $i \in nvertices(\text{graph})$ olacak şekilde $frequency[visited[i]] = frequency[visited[i]]+1$ işlemi yapılıyor. 0 değeri sonsuz olarak sayılmıştır fakat bastırılırken sonsuz kelimesi yazılmıştır. Kevin Bacon'ın da sayısı sıfır olarak atanmıştır ama sonsuz frekansından bir çıkartılarak özel olarak Kevin Bacon'ın Kevin Bacon sayısı gösterilmiştir.

- **void** shortestDistance(GRAPH*,char*,int,int*,int*,char*): Kimin Kevin Bacon ile olan bağlantısını bulmak istiyorsak o kişinin storage arrayindeki index'ini buluyoruz. Bundan sonra yapılacak işlem zaten aslında BreadthFirstSearch fonksiyonu içinde yapılmıştı. Parent tutmak. Figure 2'de görüldüğü gibi 6 numaralı node'a ulaşıldığında 6'nın parent'ı 3 numaralı node, 3'ün parent'ı 2, 2'nin parent'ı ise 0. Bunu index'lerle göstermek gerekirse:

  parent[6] = 3
  parent[parent[6]] = 2
  parent[parent[parent[6]]] = 0

  olacaktır. Her bir parent'ı path isimli bir arrayin içinde tutup, path dizisi bastırılmıştır. Parent'lar film-aktör-film-aktör... şeklinde ilerleyeceği için bastırma işlemi $\forall i \in n(path)$:

  path[i] (aktör)
  path[i+2] (film)
  path[i+1] (aktör)

  şeklinde $i = i + 2$ index arttırılması ile gerçekleşmektedir.
  Ayrıca bu fonkisyon içinde projede bonus olarak bahsedilen işlem gerçekleşmektedir. Eğer önceden istenilen ismin yolu hesaplandıysa LL isimli structa ait olan linked list'ten bu ismin yolu alınır. Eğer yoksa hesaplanır ve hesaplandıktan

sonra linked list'e yeni isim koyulur. Tekrardan o isim hesaplanmak istediğinde bu sefer linked list'teki önceden hesaplanmış yol basılır.

İstenilen Çıktılar

Adile Naşit'in ismi verilen input-3 datasında olmadığı için girilen isim verilen data içinde değildir uyarısı vermektedir.

```
(base) safak@progcu:~/Desktop/Kevin Bacon/src$ ./b17011086
Enter the file name: input-3.txt

It can take minutes, calculating...184647
-----------------------------------------
Kevin Bacon has 0 Kevin Bacon Number.
717 actors/actress: Infinite Kevin Bacon Number.
1373 actors/actress: 1 Kevin Bacon Number.
93798 actors/actress: 2 Kevin Bacon Number.
72979 actors/actress: 3 Kevin Bacon Number.
1636 actors/actress: 4 Kevin Bacon Number.
14 actors/actress: 5 Kevin Bacon Number.
-----------------------------------------

-------------------------------------------------------------------------------------------------------
Actor name and surname must be given as: Surname, Name.
Enter the actor's/actress's name to find connection map between Kevin Bacon: Streep, Meryl
-------------------------------------------------------------------------------------------------------
Streep, Meryl's Kevin Bacon number is 1:

Streep, Meryl - Bacon, Kevin -> River Wild, The (1994)

-------------------------------------------------------------------------------------------------------
To quit the program, type quit to actor's/actress's name.
Actor name and surname must be given as: Surname, Name.
Enter the actor's/actress's name to find connection map between Kevin Bacon: Cage, Nicolas
-------------------------------------------------------------------------------------------------------
Cage, Nicolas's Kevin Bacon number is 2:

Cage, Nicolas - Dillon, Matt -> Rumble Fish (1983)
Dillon, Matt - Bacon, Kevin -> Wild Things (1998)

-------------------------------------------------------------------------------------------------------
To quit the program, type quit to actor's/actress's name.
Actor name and surname must be given as: Surname, Name.
Enter the actor's/actress's name to find connection map between Kevin Bacon: Samaha, Elie
-------------------------------------------------------------------------------------------------------
Samaha, Elie's Kevin Bacon number is 3:

Samaha, Elie - Carrere, Tia -> 20 Dates (1998)
Carrere, Tia - Pickens Jr., James -> Hostile Intentions (1994)
Pickens Jr., James - Bacon, Kevin -> Sleepers (1996)

-------------------------------------------------------------------------------------------------------
To quit the program, type quit to actor's/actress's name.
Actor name and surname must be given as: Surname, Name.
Enter the actor's/actress's name to find connection map between Kevin Bacon: Fanning, Dakota
-------------------------------------------------------------------------------------------------------
Fanning, Dakota's Kevin Bacon number is 2:

Fanning, Dakota - Dern, Laura -> I Am Sam (2001)
Dern, Laura - Bacon, Kevin -> Novocaine (2001)

-------------------------------------------------------------------------------------------------------
To quit the program, type quit to actor's/actress's name.
Actor name and surname must be given as: Surname, Name.
Enter the actor's/actress's name to find connection map between Kevin Bacon: Naşit, Adile
-------------------------------------------------------------------------------------------------------
The name Naşit, Adile cannot be found in movie data!
-------------------------------------------------------------------------------------------------------
To quit the program, type quit to actor's/actress's name.
Actor name and surname must be given as: Surname, Name.
Enter the actor's/actress's name to find connection map between Kevin Bacon:
```

Figure 3: İstenilen çıktılar.

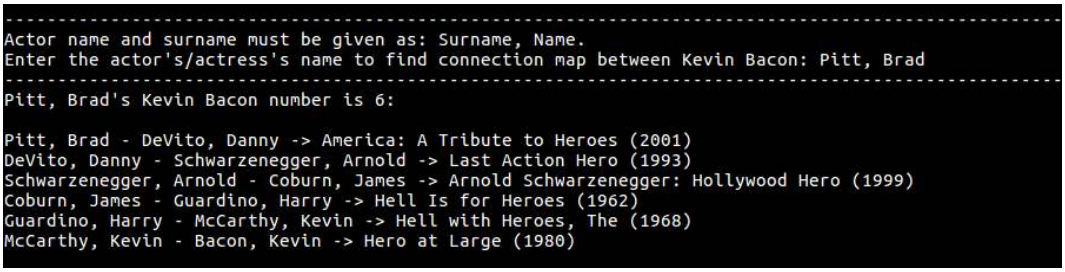

Figure 4: input-2.txt'den Pitt, Brad.

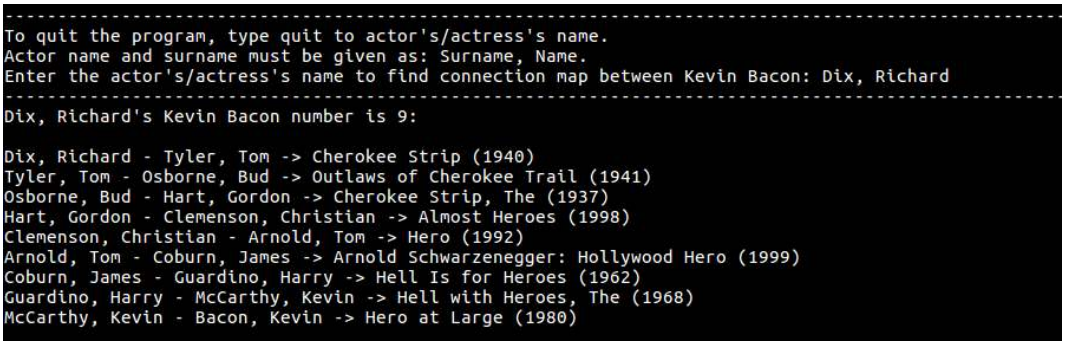

Figure 5: input-2.txt Dix, Richard.

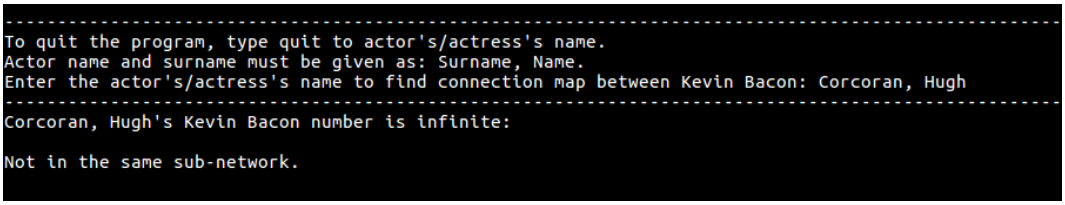

Figure 6: input-2.txt'den Corcoran, Hugh.

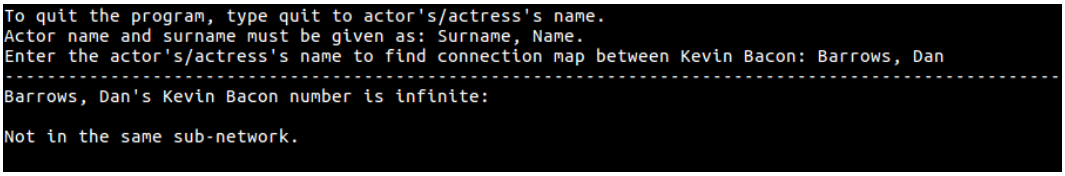

Figure 7: input-2.txt'den Barrows, Dan.

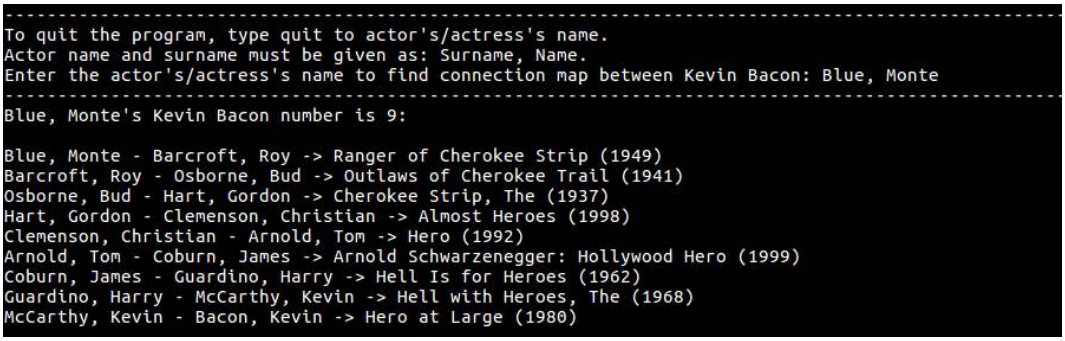

Figure 8: input-2.txt'den Blue, Monte.

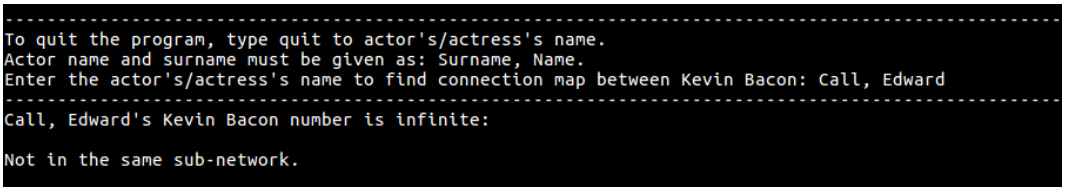

Figure 9: input-2.txt'den Call, Edward.

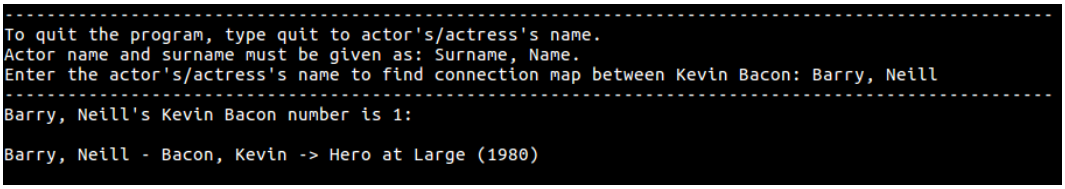

Figure 10: input-2.txt'den Barry, Neill.

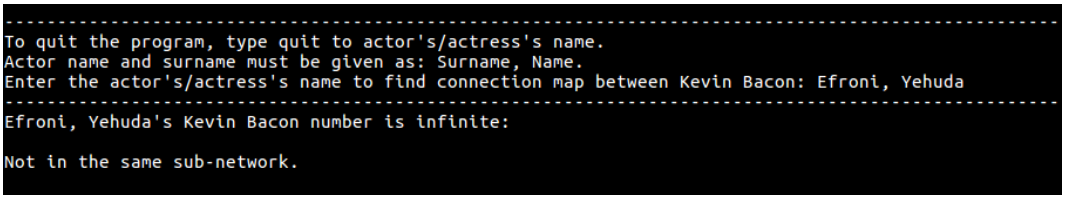

Figure 11: input-2.txt'den Efroni, Yehuda.

```c
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<uchar.h>
char storage[190000][200];
int moviectrl[190000];
struct node{
        int vertex;
        char movie_actor[200];
        int visited;
        struct node* next;
};

struct graph{
        int nVertex;
        struct node** adjList;
};
struct queue{
        int vertex;
        char movie_actor[200];
        struct queue* next;
};
struct linkedlist{ // özel olarak önceden hesaplanm¿¿ yollar¿ tutuyor.
  struct linkedlist* next; // istenildi¿i zaman önceden hesaplanm¿¿ ki¿i,
  char name[100];// ej bir hesaplama yapmadan buras¿ üzerinden görüyor
  char path[1000];
};

typedef struct linkedlist LL;
typedef struct queue QUEUE;
typedef struct node NODE;
typedef struct graph GRAPH;
int numberOfVertex(FILE*);
NODE* createNode(int,char*);
GRAPH* createGraph(int);
GRAPH* addEdge(int,char*);
void printGraph(GRAPH*,int);
void enqueue(QUEUE**,int,char*);
QUEUE* dequeue(QUEUE**);
void BreadthFirstSearch(GRAPH**,char*,int,int*,int*);
int findMaxDistance(int,int*);
int* frequencyListOfKevinBaconNumbers(GRAPH*,int,int,int*);
void update(LL**, char*);
void insert(LL**,char*);
void shortestDistance(GRAPH*,char*,int,int*,int*,char*,LL**);
int main(int argc, char** argv){
        char filename[100]; // file ismi
        printf("Enter the file name: ");
        scanf("%s",filename);
        FILE* fp = fopen(filename,"r"); // kullan¿lacak file
        if(fp == NULL){
          fprintf(stderr,"File Error!");
          exit(0);
        }
        printf("\n");

        LL* head = (LL*)malloc(sizeof(LL)); // hesaplanan verileri tutan tablo
        int i;
        printf("It can take minutes, calculating...");
        int n = numberOfVertex(fp); // vertex say¿s¿
        printf("%d\n",n);
        int* parent = (int*)malloc(n*sizeof(int)); // gezilen kom¿ular¿n parent inde
ksi
        for(i =0; i<n; i++){
                parent[i] = -1;
        }
        GRAPH* _graph = addEdge(n,filename); // grafimiz
        int* visited = (int*)calloc(n,sizeof(int)); // distance arrayimiz
        BreadthFirstSearch(&_graph,"Bacon, Kevin",n,visited,parent);
        int max = findMaxDistance(n,visited); // bacon number frequency dizisi için
boyutumuz
        int* frequency = (int*)calloc(max,sizeof(int)); // bacon number frequency li
st
```

```
        frequency = frequencyListOfKevinBaconNumbers(_graph,max,n,visited);
        printf("\n");
        char destination_actor[100]; // kimi bulmak istiyoruz
        printf("-------------------------------------------------------------------------------
--------------------------------\n");
        printf("Actor name and surname must be given as: Surname, Name.\n");
        printf("Enter the actor's/actress's name to find connection map between Kevi
n Bacon: ");
        scanf(" %[^\n]s",destination_actor);
        printf("-------------------------------------------------------------------------------
--------------------------------");
        printf("\n");
        while(strcmp(destination_actor,"quit") != 0){
          shortestDistance(_graph,"Bacon, Kevin",n,visited,parent,destination_actor,
&head);
                printf("\n");
                printf("-------------------------------------------------------------------
-----------------------------------------\n");
                printf("To quit the program, type quit to actor's/actress's name.\n"
);
                printf("Actor name and surname must be given as: Surname, Name.\n");
                printf("Enter the actor's/actress's name to find connection map betw
een Kevin Bacon: ");
        scanf(" %[^\n]s",destination_actor);
                printf("-------------------------------------------------------------------
-----------------------------------------");
        printf("\n");
        }
        return 0;
}
void update(LL** current, char path [10000]){// veri taban¿nda yoksa yolu ekliyor
  strcat((*current)->path,path); // yollar¿ birle¿tirmek için
}
void insert(LL** head, char name[100]){ // veritaban¿nda yoksa en ba¿a o bilginin
  LL* newNode = (LL*)malloc(sizeof(LL));// node'unu olu¿turuyor.
  strcpy(newNode->name,name);
  if(*head == NULL){
    *head = newNode;
    newNode->next = NULL;
  }
  else{
    newNode->next = *head;
    *head = newNode;
  }
}

NODE* createNode(int vertex, char* movie_actor){// graf için node
        NODE* newNode = (NODE*)malloc(sizeof(NODE));
        newNode->vertex = vertex;
        strcpy(newNode->movie_actor,movie_actor);
        newNode->next = NULL;
        newNode->visited = 0;
        return newNode;
}

GRAPH* createGraph(int nVertices){// adjList initialization
        int i;
        GRAPH* _graph = (GRAPH*)malloc(sizeof(GRAPH));
        _graph->nVertex = nVertices;
        _graph->adjList = (NODE**)malloc(nVertices*sizeof(NODE*));
        for(i=0; i<nVertices; i++){
                _graph->adjList[i] = NULL;
        }
        return _graph;
}
GRAPH* addEdge(int nVertices,char filename[]){// bipartite graf¿m¿z¿ olu¿turuyor
        GRAPH* _graph = createGraph(nVertices); // graf¿m¿z olu¿uyor
        FILE * fp = fopen(filename,"r"); // file
        NODE* newNode;
        char* ptr1; // new line tokenize edilen string
        char* ptr2; // kalan tokenize i¿lemleri için
        char movie_buffer[100]; // filmi tutmak için
        char buffer[10000]; // fgets ile okurken sat¿rlar¿ tutmak için
```

```
        int i=0;
        for(i=0; i<nVertices; i++){
                moviectrl[i] = -1;
        }
        i = 0;
        int j=0;
        while(fgets(buffer,10000,fp)){
                j=0;
                ptr1=strtok(buffer,"\n");
                ptr2=strtok(ptr1,"/");
                strcpy(movie_buffer,ptr2);
                while(strcmp(storage[j],movie_buffer)!=0 && j<nVertices){
                        j++;
                }
                moviectrl[j] = 1; // film oldu¿u için 1
                while(ptr2!=NULL){
                        ptr2= strtok(NULL,"/");
                        if(ptr2!=NULL){
                                while(strcmp(storage[i],ptr2)!=0 && i<nVertices){
                                        i++;
                                }
                                moviectrl[i] = 0;
                                newNode = createNode(i,ptr2);
                                newNode->next = _graph->adjList[j];
                                _graph->adjList[j] = newNode;
                                newNode = createNode(j,movie_buffer);
                                newNode->next = _graph->adjList[i];
                                _graph->adjList[i] = newNode;
                        }
                        i=0;
                }
        }
        return _graph; // edgeleri ba¿lanm¿¿ graf
}

int numberOfVertex(FILE* fp){// adjList'in boyutunu buluyor
        int nVertex=0;
        char* ptr1; // new line tokenize edilen string
        char* ptr2; // kalan tokenize i¿lemlerini tutan string
        char buffer[10000]; // line line okurken buffer
        int i=0;
        int j=0;
        while(fgets(buffer,10000,fp)){
                ptr1=strtok(buffer,"\n");
                ptr2=strtok(ptr1,"/");
                while(ptr2!=NULL){
                        while(i<nVertex && strcmp(storage[i],ptr2)!=0){
                                i++;
                        }
                        if(i==nVertex){ // e¿er yoksa storage içinde bi yeni bir akt
ördür.
                                strcpy(storage[j],ptr2);
                                j++;
                                nVertex++;
                        }
                        ptr2=strtok(NULL,"/");
                        i=0;
                }
        }
        return nVertex;
}

void printGraph(GRAPH* _graph, int n){
  GRAPH* current = _graph;
        int i;
  for(i=0; i<n; i++){
    printf("(%s||%d) : ",storage[i],i);
    while(current->adjList[i]->next!=NULL){
      printf("(%s||%d) -> ",current->adjList[i]->movie_actor,current->adjList[i]->ve
rtex);
      current->adjList[i] = current->adjList[i]->next;

    }
```

```
    printf("(%s||%d)\n",current->adjList[i]->movie_actor,current->adjList[i]->vertex
);
  }
}


void enqueue(QUEUE** q, int vertex, char movie_actor[100]){ // queue'ya eleman ekler
        QUEUE* newNode = (QUEUE*)malloc(sizeof(QUEUE));
        newNode->vertex = vertex;
        strcpy(newNode->movie_actor,movie_actor);
        if((*q)==NULL){
                newNode->next = NULL;
                (*q) = newNode;
        }
        else{
                newNode->next = *q;
                *q = newNode;
        }
}

QUEUE* dequeue(QUEUE** q){ // queue'dan eleman çeker
        if(*q == NULL){
                return NULL;
        }
        else{
                int counter = 0;
                QUEUE* current = *q;
                QUEUE* before = current;
                while(current->next != NULL){
                        counter++;
                        before = current;
                        current = current->next;
                }
                if(counter != 0){
                        before->next = NULL;
                        return current;
                }
                else if(counter == 0){
                        *q = NULL;
                        return current;
                }
        }
}

void BreadthFirstSearch(GRAPH** _graph, char actor[100], int n,int* visited,int* par
ent){
        for(int k=0; k<n; k++){
                visited[k] = -1;
        }
        QUEUE* q = (QUEUE*)malloc(sizeof(QUEUE)); // queue
        QUEUE* returned = (QUEUE*)malloc(sizeof(QUEUE)); // queue'dan çekilen eleman
        GRAPH* current = *_graph; // graph¿n headini kaybetmemek için
        q = NULL;
        int i=0,j=0;
        char actors_movies[100];
        strcpy(actors_movies,actor);
        while(strcmp(storage[i],actors_movies) != 0){//kevin bacon nerede?
          i++;
        }
        enqueue(&q,i,actors_movies);
        visited[i] = 0;
        returned = dequeue(&q);
        while(returned != NULL){
                while(strcmp(storage[j],returned->movie_actor) != 0){
                        j++;
                }
                GRAPH* tmp = current;
                while(tmp->adjList[j] != NULL){ // tüm kom¿ular¿n¿ gez
                        if(visited[tmp->adjList[j]->vertex] == -1){
                                enqueue(&q,tmp->adjList[j]->vertex,tmp->adjList[j]->
movie_actor);
                                visited[tmp->adjList[j]->vertex] =visited[returned->
vertex]  + 1; // distance hesapla
```

```
                                parent[tmp->adjList[j]->vertex] = returned->vertex;;
 // parent¿n¿ tut
                        }
                        tmp->adjList[j] = tmp->adjList[j]->next;
                }
                j=0;
                returned = dequeue(&q);
        }
}

int* frequencyListOfKevinBaconNumbers(GRAPH* _graph,int max,int n,int* visited){
        int i,j;
        GRAPH* current = _graph;
        int* frequency = (int*)calloc(max+1,sizeof(int)); // frekans dizisi
        j = 0;
        for(i=0; i<n; i++){
                if(moviectrl[i] == 0){
                 frequency[visited[i]] = frequency[visited[i]] + 1;
                }
        }
        printf("-----------------------------------------------\n");
        printf("Kevin Bacon has 0 Kevin Bacon Number.\n");
        for(i=0; i<max+1; i++){
                if(i!=0){
                        printf("%d actors/actress: %d Kevin Bacon Number.\n",frequen
cy[i],i);
                }
                else{
                        printf("%d actors/actress: Infinite Kevin Bacon Number.\n",f
requency[i]-1);
                }
        }
        printf("-----------------------------------------------\n");
        return frequency;
}

int findMaxDistance(int n,int* visited){ // frekans dizisinin boyunu hesaplar
        int i;
        int max;
        int start = 0;
        for(i=0; i<n; i++){
                if(moviectrl[i] == 0 && start == 0){
                        visited[i] = visited[i]/2;
                        max = visited[i];
                        start++;
                }
                if(moviectrl[i] == 0 && start !=0){
                        visited[i] = visited[i]/2;
                        if(max< visited[i]){
                        max = visited[i];
                        }
                }
        }
        return max;
}

void shortestDistance(GRAPH* _graph,char from[100], int n,int* visited, int* parent,
char destination_actor[100], LL** head){
        // connection map'i bast¿r¿r
        int i;
        //      GRAPH* current = _graph;
        LL* current = *head;
        int path[10000]; // yolumuzu tutar
        int pathIndex=0; // ilk eleman
        char buffer[200];
        i = 0;
        while(current != NULL){ // önceden hesaplanm¿¿ m¿ hesaplanmam¿¿ m¿?
          if(current != NULL && strcmp(current->name,destination_actor) == 0){
            printf("Mined from linked list database...");
            printf("%s",current->path);
            return;
          }
          current = current->next;
```

```
        }
        // yukardaki whilde içinde return ile fonksiyondan ç¿kmazsa a¿a¿¿daki
        // linked list tablosuna yeni verileri ekleme i¿lemleri
        // update fonksiyonu ile yap¿l¿yor
        insert(head,destination_actor);
        while(i<n && strcmp(storage[i],destination_actor) != 0){
                i++;
        }
        if(i == n){
                printf("The name %s cannot be found in movie data!\n",destination_ac
tor);
                update(head,"The name cannot be found in movie data!\n");
        }
        else if(strcmp(destination_actor,"Bacon, Kevin") == 0){
                printf("Kevin Bacon's Kevin Bacon Number is 0.");
                update(head,"Kevin Bacon's Kevin Bacon Number is 0.\n");
        }
        else{
                int distance = i;
                path[pathIndex] = i;
                pathIndex++;
                while(parent[i] != -1){
                        path[pathIndex] = parent[i];
                        pathIndex++;
                        i = parent[i];
                }
                if(visited[distance] != 0){
                        char bf[20];
                        printf("%s's Kevin Bacon number is %d:\n\n",destination_acto
r,visited[distance]);
                        sprintf(bf,"%d",visited[distance]); // distane'¿ stringe çev
irir
                        update(head,"Kevin Bacon number is : ");
                        update(head,bf);
                        update(head,"\n");
                }
                else{
                        update(head,"Kevin Bacon number is infinite:");
                        update(head,"\n");
                        printf("%s's Kevin Bacon number is infinite:\n\n",destinatio
n_actor);

                }
                char cat[1000];
                for(i=0; i<pathIndex-2; i=i+2){
                  update(head,storage[path[i]]);
                  update(head," - ");
                  update(head,storage[path[i+2]]);
                  update(head," -> ");
                  update(head,storage[path[i+1]]);
                  update(head,"\n");

                        printf("%s - %s -> %s\n",storage[path[i]],storage[path[i+2]]
,storage[path[i+1]]);
                }
                //printf("%s",(*head)->path);
                if(i==0){
                  update(head,"Not in the same sub-network.\n");
                                printf("Not in the same sub-network.\n");
                }
        }
}
```