Designing and Mining a Blood-Bank Management Database System

**A Thesis Presented
to the University of the
Ahsanullah University Of Science And Technology**

**In Partial Fullfilment
of the Requirements for the Degree of
Bachelor of Science in
Computer Science and Engineering**

By ——

Deepa Das	**: 07.01.04.006**
Fareal Ahmed	**: 07.01.04.024**
Syeda Shabnam Hasan	**: 07.01.04.030**
Jannatul Ferdous	**: 07.01.04.047**

March 2011

Thesis supervisor : Ms. Rosina Surovi Khan

Acknowledgment

The enduring pages of the work are the cumulative sequence of extensive guidance and arduous work. We wish to acknowledge and express our gratitude to all those without whom our thesis could not have been a reality. We feel very delighted to get this rare opportunity to show our profound senses of reverence and indebtedness to our thesis supervisor Ms. Rosina Surovi Khan for the information she provided to us through the lecture sittings and her invaluable timely advice and guidance. We would like to extend our sincere thanks to her for giving us her precious time and for being always available to us in order to clarify our doubts regarding the thesis. This thesis is dedicated to our parents who have given us the opportunities of education from the best institutions and support throughout our lives. The last but not the least we would like to thank all those who have directly or indirectly helped and cooperated in accomplishing this thesis.

Abstract

A database is the single most useful environment in which to store data and an ideal tool to manage and manipulate that data. The benefits of a well-structured database are infinite, with increased efficiency and time-saving benefits. Our team's interest is centered around this area. At the very start, we create a database on blood-bank management system. We use Microsoft SQL Server for this purpose. We determine attributes and entities and figure out relationships among entities. Then we draw the entity-relationship diagram, convert it to a relational model (relational tables) and normalize the tables. We implement the design, create tables and insert values inside the tables using sql server. We execute sample queries on the system and verify that our system contains all required information making retrieval of the information fast and efficient. In part II of the thesis, we convert the database tables of the system to text files. Using exact and approximate string matching algorithms, we match a string in question with the strings in text files and get the index of exactly matched strings for the former and obtain approximately matched strings displaying edit distances between the two for the latter.

CONTENTS

Part-I

Chapter 1

Introduction

1.1 What is a database?

A database is a collection of organized interrelated data. Traditionally the data will be presented something like this:

firstname	surname	Dob
John	Smith	01/12/76
Sara	Jones	13/06/69
Fred	Bloggs	11/11/73

Tables in a database are used for storing specific collections of data.

1.2 Advantages of Database

- It means all of the information is together.
- The information can be portable if on a laptop.
- The information is easy to access at any time.
- It is easily retrievable.
- Many people can access the same database at the same time.
- Improved data security.
- Reduced data redundancy.
- Reduced updating errors and increased consistency.
- Greater data integrity and independence from applications programs.
- Improved data access to users through use of host and query languages.
- Reduced data entry, storage, and retrieval costs.
- Facilitated development of new applications program.

1.3 Disadvantages of Database

- Database systems are complex, difficult, and time-consuming to design.
- Initial training required for all programmers and users.
- Suitable hardware and software start-up costs.
- A longer running time for individual applications.
- Damage to database affects virtually all applications programs.
- Extensive conversion costs in moving from a file-based system to a database system.

1.4 Components of Database Design

- Entity relationship model
- Relational Model (Relational tables)
- Normalization of tables
- Implementation in SQL server
- Usage of the system (Execution of sample complex queries)

Chapter 2

Database Design

2.1 The Entity-Relationship Model

The entity-relationship (E-R) model was developed to facilitate database design by allowing specification of an enterprise schema that represents the overall logical structure of a database. The E-R data model is one of the several semantics data models; the semantic aspect of the model lies in its representation of the meaning of the data. The E-R model is very useful in mapping the meanings and interactions of real-world enterprises onto conceptual schema. The E-R data model has three basic notions: entity-sets, relationship sets and attributes.

Entity sets: An entity is a thing or object in the real world that is distinguishable from all other objects. It has a set of properties, and the values for some set of properties may uniquely identify an entity. It is also a set of entities of the same type that share the same properties.

Relationship sets: A relationship is an association among several entities. It is a set of relationships of the same type. The association between entity sets is referred to as participation. The function that an entity plays in a relationship is called that entity's role.

Attributes: An entity is represented by a set of attributes. Attributes are descriptive properties possessed by each member of an entity set. The designation of an attribute for an entity set expresses that the database stores similar information concerning each entity in the entity set; however, each entity may have its own value for each attribute. [1]

Some important features of E-R model:

Mapping Cardinality: Mapping cardinalities express the number of entities to which another entity can be associated via a relationship set. Cardinality can be---

→ One-to-one: an entity of a set can be associated with at most one entity of another.

→ One-to-many: an entity of a set is associated with any number (entities) of another set.

→ Many-to-one: an entity (1st set) is associated with at most one entity (of 2nd set). But 2nd set's entity can associate with any number of 1st entity set.

→ Many-to-many: Entities of both sets can be associated with any number of entities between them.

E-R diagram: It can express the overall logical structure of a database graphically. A diagram consists of some major components—

# Rectangles: represent entity set.
# Ellipses: represent attributes.
# Diamonds: represent relationships.
# Lines: which link attributes to entity sets and entity sets to relationship sets. [1]

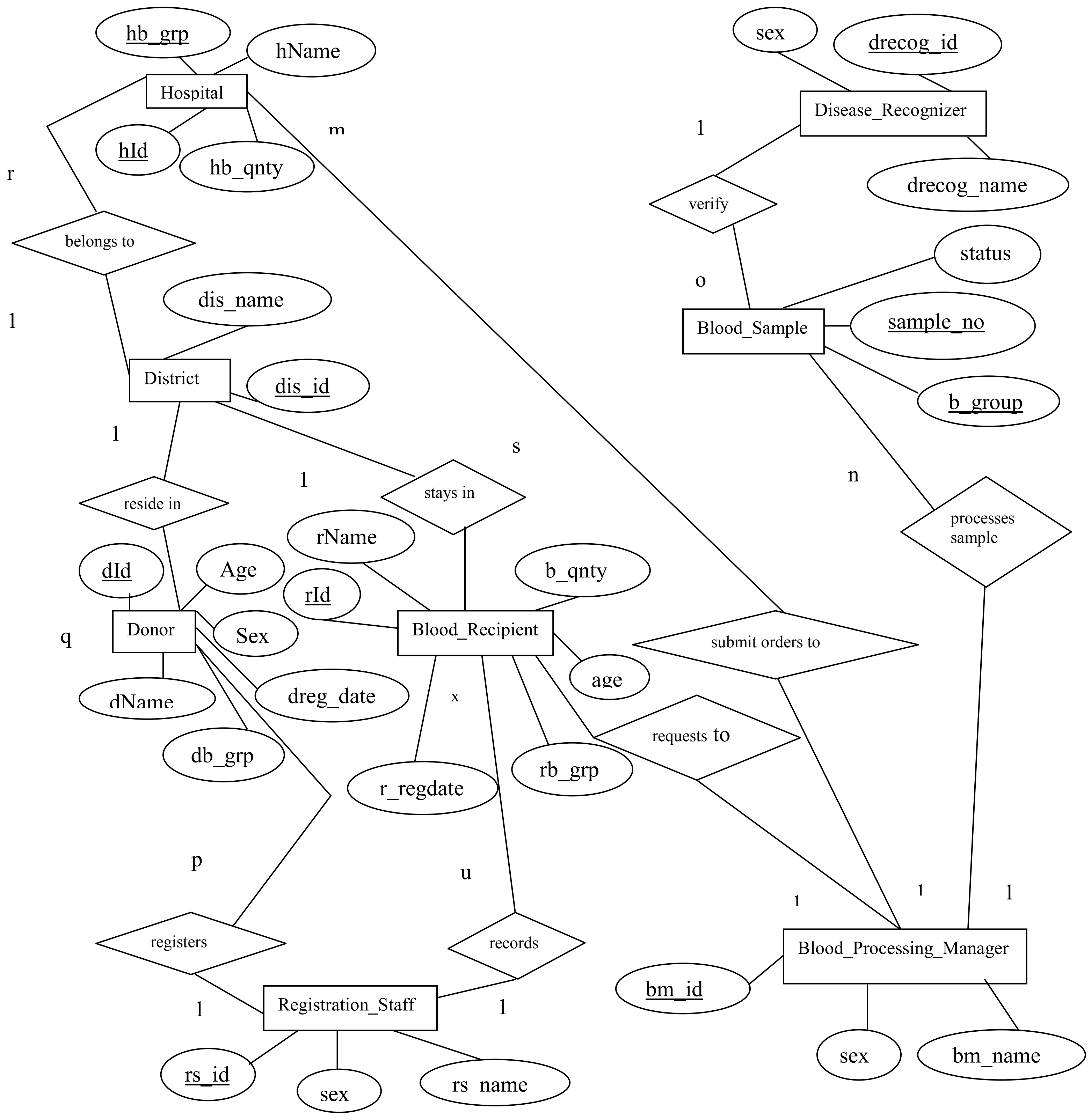


Figure 2.1.1 : ER diagram of Blood Bank Management System

Our E-R diagram represents the Blood-Bank Management system. It has eight entity sets. They are—

a) Donor: (Attributes- dName, <u>dId</u>, sex, age, dreg_date, db_grp).

b) District: (Attributes- <u>dis_id</u>, dis_name).

c) Registration_Staff: (Attributes- <u>rs_id</u>, rs_name, sex).

d) Blood_Recipient: (Attributes- <u>rId</u>, sex, age, r_regdate, rName, b_qnty, rb_grp).

e) Blood_Sample: (Attributes- <u>b_group</u>, <u>sample_no</u>, status).

f) Disease_Recognizer: (Attributes- <u>drecog_id</u>, drecog_name, sex).

g) Blood_Processing_Manager: (Attributes-<u>bm_id</u>, bm_name, sex).

h) Hospital: (Attributes- <u>hId</u>, hName, <u>hb_grp</u>, hb_qnty).

Abbreviations of all attributes are given in relational schema.
Some notes about entity sets, their attributes and cardinalities among them----

Donor- Who donates blood. When a donor will donate, an id(a serial number will be given for a specific identification (primary key)); age, sex, name, registration date (dreg_date) and blood group will be stored in the database under entity Donor.

District- Every district's/location's id is different (primary key).

Registration_Staff- Registration staffs will register the information of donors and the recipients.

Disease_Recognizer-Disease recognizer will test blood samples whether the samples are contaminated or okay.

Blood_Processing_Manager- They will take orders from the hospitals and fulfill their needed requirements of blood samples.

Blood_Sample- The quantities of blood that the Blood_bank has. Their group, sample_no, status will be stored.

Hospital- Hospitals of each district, where blood samples are needed, also included in the database.

Blood_Recipient- Who needs blood. A recipient's id, name, age, sex, the blood sample's group information will be stored in database.

Cardinality:

District & Donor- (Relationship- (stays_in), 1 to many). One donor stays in one district. In one district, many donors can stay.

Registration_Staff & Donor- (Relationship-(registers), 1 to many). A staff can ensure many donors' registration. One donor can get registered by one staff.

Registration_Staff & Blood_Recipient- (Relationship-(records), 1 to many). A staff can ensure many blood_recipients' registration. One blood_recipient can get registered by one staff.

District & Blood_Recipient - (Relationship-(resides_in), 1 to many). One recipient stays in one district. In one district, many recipients can stay.

District & Hospital- (Relationship-(belongs to), 1 to many).In a district, there are many hospitals. One hospital belongs to one district.

Blood_Processing_Manager & Hospital- (Relationship-(submit_orders_to), 1 to many). A blood processing manager can get orders from many hospitals. One hospital submits order to a blood processing manager.

Blood_Processing_Manager & Blood_Sample-(Relationship-(processes_sample), 1 to many). A manager can process many samples of blood. One blood sample can be processed by one blood processing manager.

Disease_Recognizer & Blood_sample- (Relationship-(verify), 1 to many). A disease recognizer can verify many blood samples. One blood sample is verified by one disease recognizer.

Blood_Processing_Manager & Blood_Recipient-(Relationship-(request_to), 1 to many). The samples of blood are given according to the necessity of the recipients, processed by the manager. A manager can process many samples of blood that are requested by the recipients. But one recipient can request only one blood processing manager.

2.2 Relational Schemas

Donor

Table 2.2.1

Attribute Name	Description	Type
dName	Name of the donor	varchar
Did	Id of the donor	Int
Sex	Sex of the donor	char
Age	Age of the donor	Int
dreg_date	Registration date of the donor	date
rs_id (fk)	Id of the registration staff	Int
dis_id(fk)	District id	Int
db_grp	Donor's blood group	varchar

The relationship with Registration_staff and Donor is 1 to many. That's why primary key of Registration_staff is used as a foreign key in Donor.
The relationship with District and Donor is 1 to many. That's why primary key of District is used as a foreign key in Donor.

District

Table 2.2.2

Attribute Name	Description	Type
dis_id	District id	Int
dis_name	Name of the district	Varchar

Registration_Staff

Table 2.2.3

Attribute Name	Description	Type
rs_id	Id of the registration staff	Int
rs_name	Name of the registration staff	varchar
Sex	Sex of the registration staff	char

Blood_Recipient

Table 2.2.4

Attribute Name	Description	Type
Rid	Id of the recipient	int
Sex	Sex of the recipient	char
Age	Age of the recipient	int
r_regdate	Registration date of the recipient	date
Rname	Name of the recipient	varchar
b_qnty	Needed quantity of blood	int
rb_grp	Recipient’s blood group	varchar
rs_id (fk)	Id of the registration staff	int
dis_id (fk)	District id	int
bm_id (fk)	Blood processing manager’s id	int

The relationship with Registration_staff and Blood_Recipient is 1 to many. That’s why primary key of Registration_staff is used as a foreign key in Blood_Recipient.

The relationship with District and Blood_Recipient is 1 to many. That’s why primary key of District is used as a foreign key in Blood_Recipient.

The relationship with Blood_Processing_Manager and Blood_Recipient is 1 to many. That’s why primary key of Blood_Sample is used as a foreign key in Blood_Recipient.

Blood_Sample

Table 2.2.5

Attribute Name	Description	Type
b_group	Blood group of the sample	varchar
sample_no	Sample identification number	int
Status	Status of the blood sample	varchar
drecog_id (fk)	Disease Recognizer’s id	int
bm_id (fk)	Blood processing manager’s id	int

The relationship with Disease_Recognizer and Blood_Sample is 1 to many. That’s why primary key of Disease_Recognizer is used as a foreign key in Blood_Sample.

The relationship with Blood processing manager and Blood_Sample is 1 to many. That’s why primary key of Blood processing manager is used as a foreign key in Blood_Sample.

Disease_Recognizer

Table 2.2.6

Attribute Name	Description	Type
drecog_id	Disease Recognizer's id	Int
drecog_name	Disease Recognizer's name	varchar
Sex	Disease Recognizer's sex	char

Blood_Processing_Manager

Table 2.2.7

Attribute Name	Description	Type
bm_id	Blood processing manager's id	int
bm_name	Blood processing manager's name	varchar
Sex	Blood processing manager's sex	char

Hospital

Table 2.2.8

Attribute Name	Description	Type
Hid	Hospital's id	int
hb_qnty	Needed quantity of blood in a hospital	int
hb_grp	Needed blood group	varchar
HName	Hospital's Name	varchar
dis_id(fk)	District's id	int
bm_id(fk)	Blood processing manager's id	int

The relationship with District and Hospital is 1 to many. That's why primary key of District is used as a foreign key in Hospital.

The relationship with Blood processing manager and Hospital is 1 to many. That's why primary key of Blood processing manager is used as a foreign key in Hospital.

2.3 Normalization

Boyce Codd introduced a number of ‘normal forms’ (1970- 1972). They are principles that can hold for a given relation or not.

The formal definition of Normalization is: it is the sequence of steps by which a relational database model is both created and improved upon. The sequence of steps involved in the normalization process is called *normal forms*. Essentially, normal forms applied during a process of normalization allow creation of a relational database model as a step-by-step progression.

Normal Forms:

 First Normal Form (1NF): A relation is in first normal form if it contains only simple, atomic values for attributes, no sets; that is, if attributes do not have sub attributes.

Example:

Name	Place
Karim	Dhaka
Rahim	Comilla

This relation is in first normal form because attributes do not have sub attributes.

 Second Normal Form (2NF): A relation is in second normal form, if it is in 1NF and every non-primary key attribute is fully functionally dependent on the primary key of the relation.

Example:
Relation :($\underline{A}$, B, C, D)
{A} = > {B}
{A} = > {C}
{A} = > {D}

It is in 2NF because it is in 1NF and every non-primary key attribute is fully functionally dependent on the primary key of the relation. [2]

 Third Normal Form (3NF): A relation is in third normal form, if it is in 2NF and no non-primary key attribute is transitively dependent on the primary key.

Example:
Relation :($\underline{A}$, $\underline{B}$, C, D, E)

{A, B} = > {C}
{A, B} = > {E}

This relation is in third normal form because it is in 2NF and no non-primary key attribute is transitively dependent on the primary key.

 Boyce-Codd Normal Form (BCNF): A relation is in BCNF, if for every full functional dependency X = > Y holds: X is a candidate key. If part of primary key is fully functionally dependent on non primary key, BCNF violation occurs.

Example:
Relation :(A, B, C, D)
{A, B} = > {C, D}
{C} = > {A}
In 1NF, 2NF, 3NF, but not in BCNF. Because part of primary key, A is fully functionally dependent on non- primary key C. We have to split the original relation.
(A, B, D), (C, A).
Now in BCNF.

 Advantages of normalization:

i. Many unnecessary redundancies are avoided.
ii. Anomalies with input, deletion and updates can be avoided.
iii. Fully normalized, relations tend to need less space than if not normalized.

 Disadvantages of normalization:

i.Normalization splits entities and relationships into many relations, thus making them harder to understand.
ii. Queries become more complex because they have to involve more relations.
iii. Response times are longer because of a higher number of joins in the queries. [2]

Normalization of Blood Bank database:

1. Donor (dId, dName, sex, age, dreg_date, rs_id, dis_id, db_grp)

{dId} = > {dName} (functional dependency exists, because two different dNames do not correspond to the same dId).

{dId}_= > {sex} (functional dependency exists).
{dId} = > {age} (functional dependency exists).
{dId} = > {dreg}_date (functional dependency exists).
{dId} = > {rs_id} (functional dependency exists).
{dId} = > {dis_id} (functional dependency exists).
{dId} = > {db_grp} (functional dependency exists).

The relation is in 1NF because its attributes do not have sub attributes.

The relation is in second normal form, as it is in 1NF and every non-primary key attribute is fully functionally dependent on the primary key of the relation.

The relation is in third normal form, as it is in 2NF and no non-primary key attribute is transitively dependent on the primary key.

No part of primary key is fully functionally dependent on non-primary key. So, the relation is in BCNF

2. District (dis_id , dis_name)

{dis_id}= > {dis_name}

The relation is in 1NF.
The relation is in second normal form.
The relation is in third normal form.
The relation is in BCNF.

3. Registration_staff (rs_id, rs_name, sex)

{rs_id} = > {rs_name} (functional dependency exists).
{rs_id} = > {sex} (functional dependency exists).

The relation is in 1NF.
The relation is in second normal form.
The relation is in third normal form.
The relation is in BCNF.

4. Blood_recipient (rId, sex, age, r_regdate, rName, b_qnty, rb_grp, rs_id, dis_id, bm_id)

{rId} = > {sex} (functional dependency exists).
{rId} = > {age} (functional dependency exists).
{rId} = > {r_regdate} (functional dependency exists).
{rId} = > {rName} (functional dependency exists).
{rId} = > {b_qnty} (functional dependency exists).

{rId} = > {rb_grp} (functional dependency exists).
{rId} = > {rs_id} (functional dependency exists).
{rId} = > {dis_id} (functional dependency exists).
{rId} = > {bm_id} (functional dependency exists).

The relation is in 1NF.
The relation is in second normal form.
The relation is in third normal form.
The relation is in BCNF.

5. Blood_Sample (b_group, sample_no, status, drecog_id, bm_id)

{b_group,sample_no} = > {status} (functional dependency exists).
{b_group,sample_no} = > {drecog_id} (functional dependency exists).
{b_group,sample_no} = > {bm_id} (functional dependency exists).

The relation is in 1NF.
The relation is in second normal form.
The relation is in third normal form.
The relation is in BCNF.

6. Disease_recognizer (drecog_id, drecog_name, sex)

{drecog_id} = > {drecog_name}.
{drecog_id} = > {sex} (functional dependency exists).

The relation is in 1NF.
The relation is in second normal form.
The relation is in third normal form.
The relation is in BCNF.

7. Blood_processing_manager (bm_id, bm_name, sex)

{bm_id} = >{bm_name}
{bm_id} = > {sex} (functional dependency exists)

The relation is in 1NF.
The relation is in second normal form.
The relation is in third normal form.
The relation is in BCNF.

8. Hospital (hId, hb_qnty, hb_grp ,dis_id, bm_id, hName)

{hId}= > {hName, dis_id, bm_id}
{hId, hb_grp} = > hb_qnty (functional dependency exists)

The relation is in 1NF.
The relation is not in second normal form, as it is in 1NF but not every non-primary key attribute is fully functionally dependent on the primary key of the relation. So, we have to split the relation.

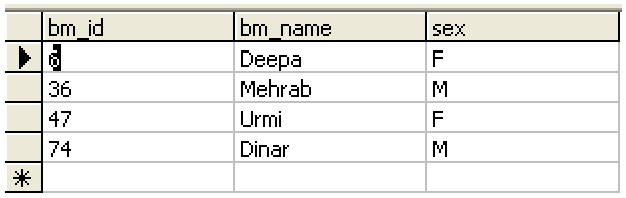
Hospital_1(hId, hName,dis_id,bm_id).
Hospital_2(hId, hb_grp, hb_qnty)

Now it is in 2NF.
The relation is in third normal form.
The relation is in BCNF.

2.4 Tables with sample values after Normalization

Blood-Processing-Manager

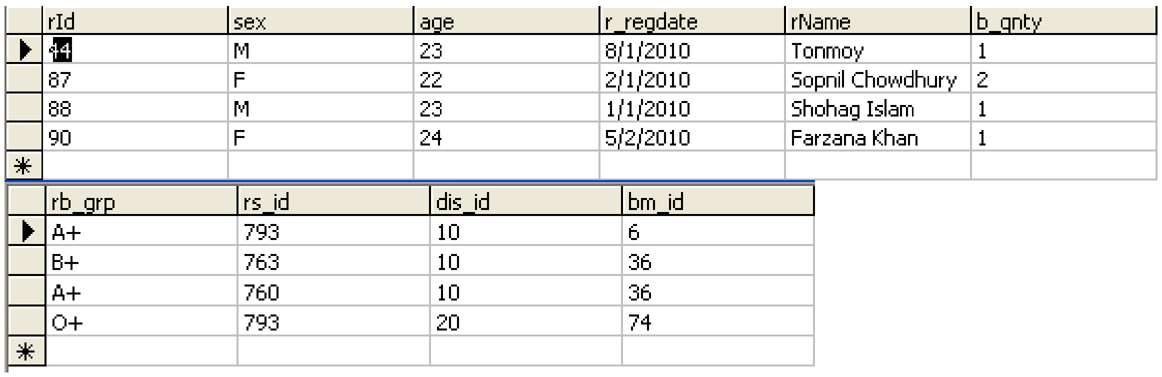
Table 2.4.1



bm_id	bm_name	sex
6	Deepa	F
36	Mehrab	M
47	Urmi	F
74	Dinar	M

Blood_Recipient

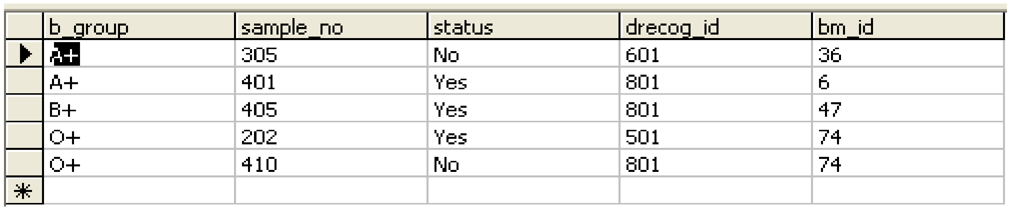
Table 2.4.2



rId	sex	age	r_regdate	rName	b_qnty	rb_grp	rs_id	dis_id	bm_id
44	M	23	8/1/2010	Tonmoy	1	A+	793	10	6
87	F	22	2/1/2010	Sopnil Chowdhury	2	B+	763	10	36
88	M	23	1/1/2010	Shohag Islam	1	A+	760	10	36
90	F	24	5/2/2010	Farzana Khan	1	O+	793	20	74

Blood_Sample

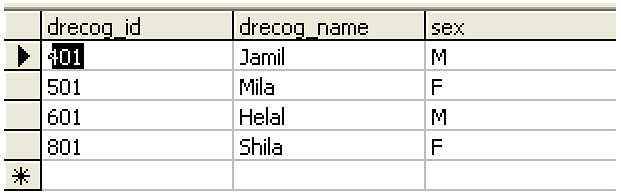
Table 2.4.3



b_group	sample_no	status	drecog_id	bm_id
A+	305	No	601	36
A+	401	Yes	801	6
B+	405	Yes	801	47
O+	202	Yes	501	74
O+	410	No	801	74

Disease-Recognizer

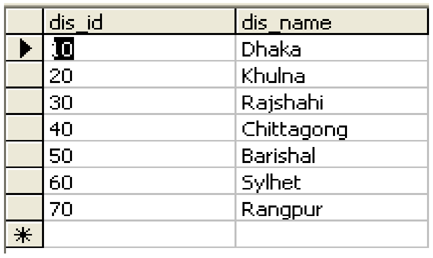
Table 2.4.4



drecog_id	drecog_name	sex
401	Jamil	M
501	Mila	F
601	Helal	M
801	Shila	F

District

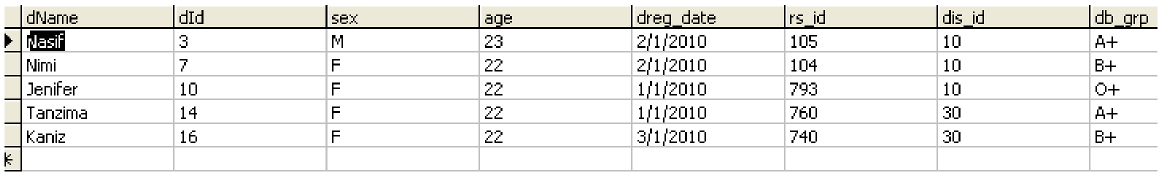
Table 2.4.5



dis_id	dis_name
10	Dhaka
20	Khulna
30	Rajshahi
40	Chittagong
50	Barishal
60	Sylhet
70	Rangpur

Donor

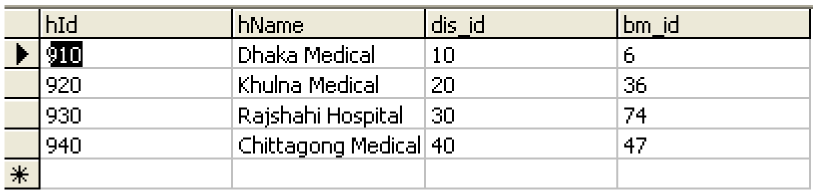
Table 2.4.6



dName	dId	sex	age	dreg_date	rs_id	dis_id	db_grp
Nasif	3	M	23	2/1/2010	105	10	A+
Nimi	7	F	22	2/1/2010	104	10	B+
Jenifer	10	F	22	1/1/2010	793	10	O+
Tanzima	14	F	22	1/1/2010	760	30	A+
Kaniz	16	F	22	3/1/2010	740	30	B+

Hospital_1

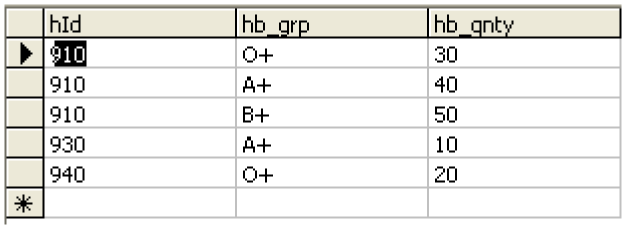
Table 2.4.7



hId	hName	dis_id	bm_id
910	Dhaka Medical	10	6
920	Khulna Medical	20	36
930	Rajshahi Hospital	30	74
940	Chittagong Medical	40	47

Hospital_2

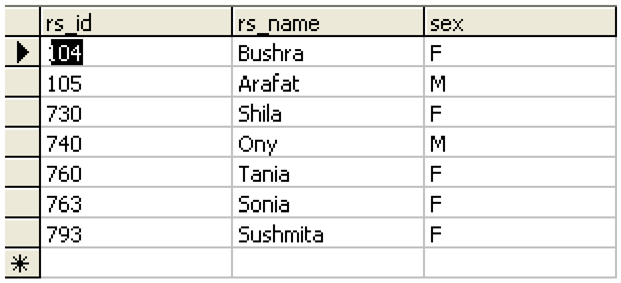
Table 2.4.8



hId	hb_grp	hb_qnty
910	O+	30
910	A+	40
910	B+	50
930	A+	10
940	O+	20

Registration_Staff

Table 2.4.9



rs_id	rs_name	sex
104	Bushra	F
105	Arafat	M
730	Shila	F
740	Ony	M
760	Tania	F
763	Sonia	F
793	Sushmita	F

Implementation in SQL Server-

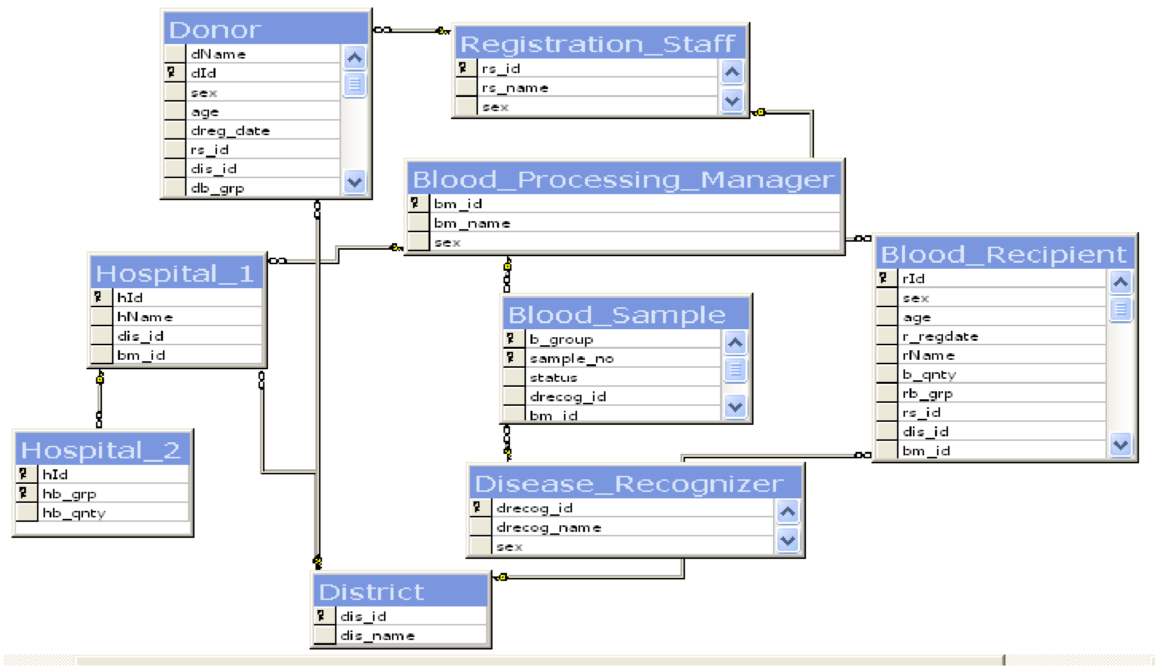
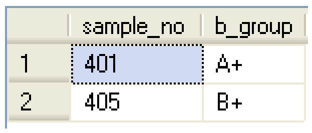


Figure 2.4.1 : Blood-Bank Management Database System

2.5 Queries

1. Show the uncontaminated blood samples verified by Dr. Shila.

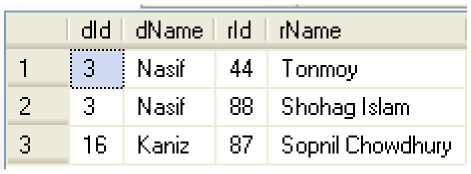
```
Select sample_no,b_group FROM Blood_Sample,Disease_Recognizer
WHERE Blood_Sample.drecog_id=Disease_Recognizer.drecog_id AND
drecog_name='Shila' AND status='Yes'
```



	sample_no	b_group
1	401	A+
2	405	B+

2. Show the donors having the blood groups that are required by recipients living in the same district. Show the recipient details also.

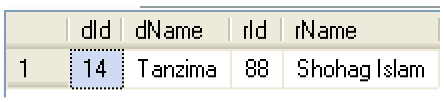
```sql
SELECT dId,dName,rId,rName FROM Donor,Blood_Recipient WHERE
db_grp=rb_grp AND Donor.dis_id=Blood_recipient.dis_id
```



	dId	dName	rId	rName
1	3	Nasif	44	Tonmoy
2	3	Nasif	88	Shohag Islam
3	16	Kaniz	87	Sopnil Chowdhury

3. Show the donor and recipients details having same blood group registered by staff Tania on the same date.

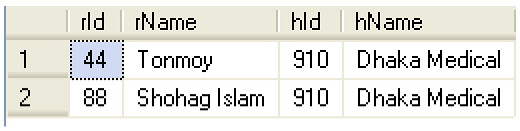
```sql
select dId,dName,rId,rName from Donor,Blood_Recipient,Registration_Staff
where db_grp=rb_grp AND dreg_date=r_regdate AND
Donor.rs_id=Blood_Recipient.rs_id AND rs_name='Tania'
```



	dId	dName	rId	rName
1	14	Tanzima	88	Shohag Islam

4. Show detailed information of the recipients and hospitals of Dhaka city who need A+ blood group.

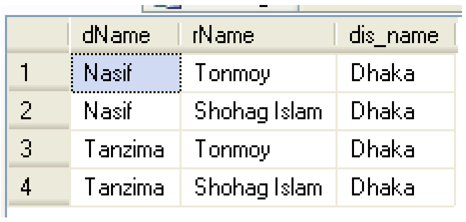
```sql
select rId,rName,Hospital_1.hId,hName from Hospital_1,Hospital_2,District,Blood_Recipient
where Hospital_1.hId=Hospital_2.hId and hb_grp='A+' AND Hospital_2.dis_id=District.dis_id .
dis_name='Dhaka' and rb_grp=hb_grp and Blood_Recipient.dis_id=District.dis_id
```



	rId	rName	hId	hName
1	44	Tonmoy	910	Dhaka Medical
2	88	Shohag Islam	910	Dhaka Medical

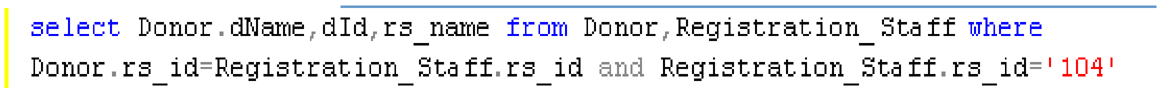
5. Find out the recipient name who took A+ type blood from the donor(also show donor's name) and both's district ids must be '10'.

```sql
select dName,rName,dis_name from
Donor,Blood_Recipient,District where Donor.db_grp='A+'
and
Donor.db_grp=Blood_Recipient.rb_grp
and District.dis_id='10'
```

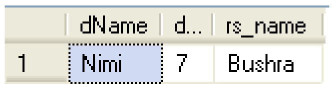


	dName	rName	dis_name
1	Nasif	Tonmoy	Dhaka
2	Nasif	Shohag Islam	Dhaka
3	Tanzima	Tonmoy	Dhaka
4	Tanzima	Shohag Islam	Dhaka

6. Find out donor name, id who is registered by registration staff_id '104' and show the registration staff's name also.

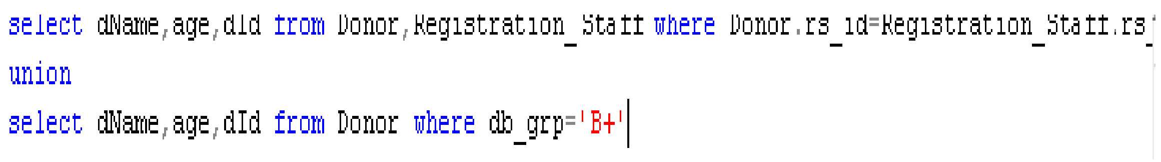


```
select Donor.dName,dId,rs_name from Donor,Registration_Staff where
Donor.rs_id=Registration_Staff.rs_id and Registration_Staff.rs_id='104'
```

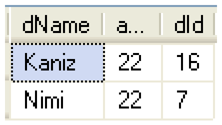


	dName	d...	rs_name
1	Nimi	7	Bushra

7. List the name, age and id of donor who is registered by registration staff 'Bushra' or who have B+ blood group

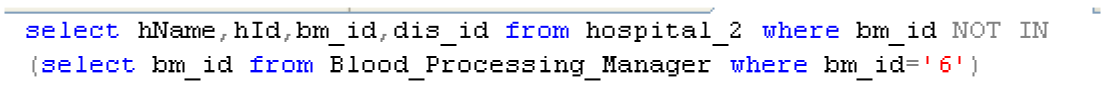


```
select dName,age,dId from Donor,Registration_Staff where Donor.rs_id=Registration_Staff.rs_
union
select dName,age,dId from Donor where db_grp='B+'
```

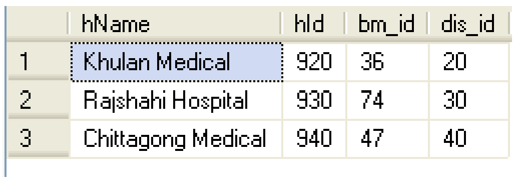


dName	a...	dId
Kaniz	22	16
Nimi	22	7

8. Find out all information about hospital_2 which has not been processed by the blood processing manager having id '6'.



```
select hName,hId,bm_id,dis_id from hospital_2 where bm_id NOT IN
(select bm_id from Blood_Processing_Manager where bm_id='6')
```



	hName	hId	bm_id	dis_id
1	Khulan Medical	920	36	20
2	Rajshahi Hospital	930	74	30
3	Chittagong Medical	940	47	40

Part-II

Chapter 3

Text File Exportation

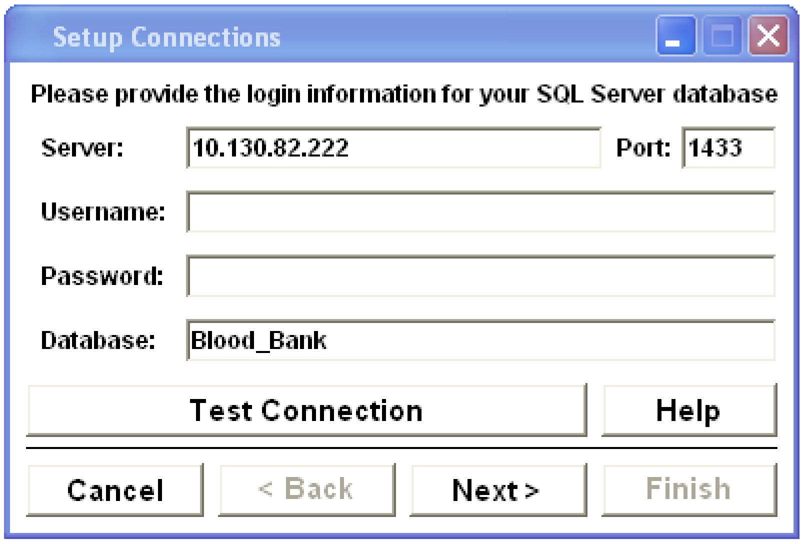
We will be exporting text files from the database tables in order to mine data. Mining data in our thesis means matching a string in question with that of text files and if found, will return the index of exactly matched string or the approximately matched string along with the edit distance depending on the algorithms.

3.1 Exporting Text Files From Database Tables

MS SQL Server Export Table to Text File Software is used to export text files from database tables.

Subsequent steps are:

1. At first, we connected our database with the local server.



Please provide the login information for your SQL Server database

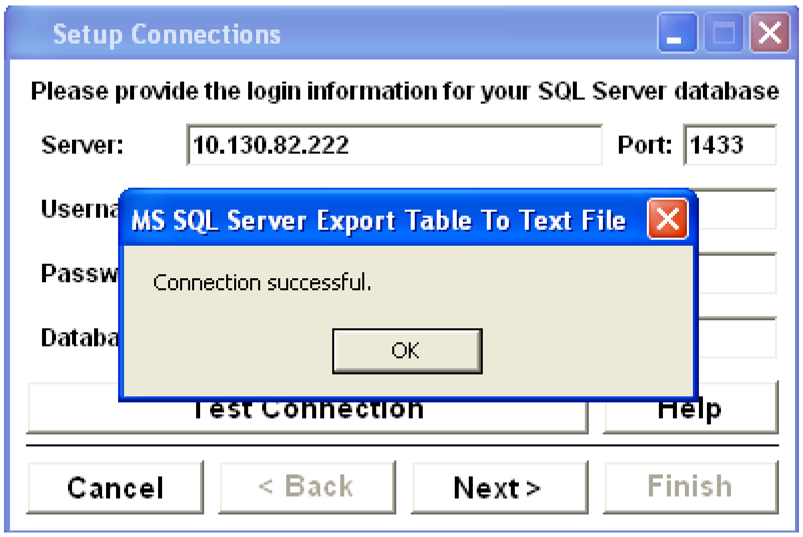
Server: 10.130.82.222 Port: 1433

Username:

Password:

Database: Blood_Bank

Figure 3.1.1


Setup Connections

Please provide the login information for your SQL Server database

Server: 10.130.82.222 Port: 1433

Userna

Passw

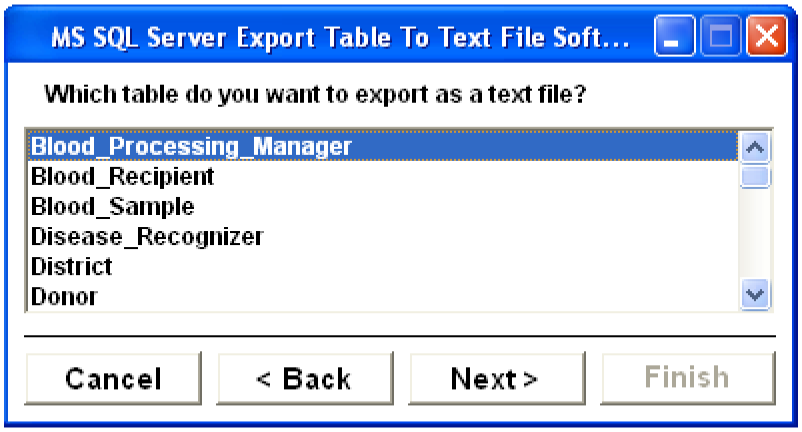
Databa

MS SQL Server Export Table To Text File

Connection successful.

Figure 3.1.2

2. Then we opened tables one by one.

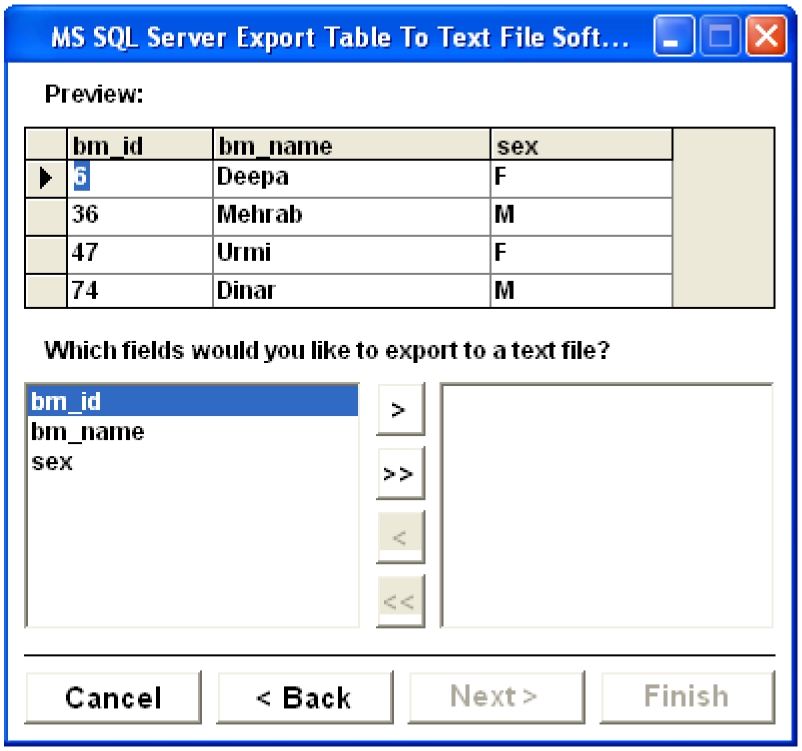

MS SQL Server Export Table To Text File Soft...

Which table do you want to export as a text file?

Blood_Processing_Manager
Blood_Recipient
Blood_Sample
Disease_Recognizer
District
Donor

Figure 3.1.3

3. We selected the table names with their respective fields to export as text files.


MS SQL Server Export Table To Text File Soft...

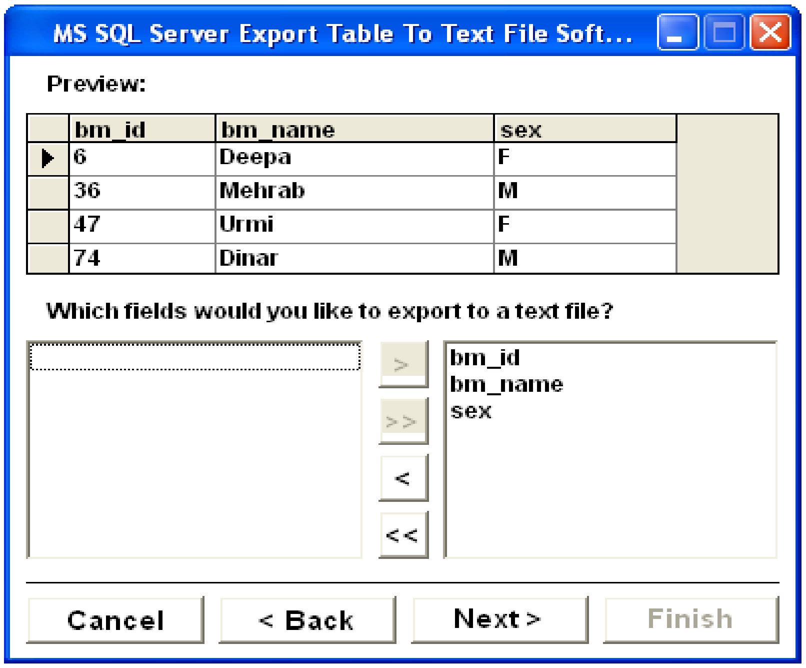
Preview:

bm_id	bm_name	sex
6	Deepa	F
36	Mehrab	M
47	Urmi	F
74	Dinar	M

Which fields would you like to export to a text file?

bm_id
bm_name
sex

Figure 3.1.4


MS SQL Server Export Table To Text File Soft...

Preview:

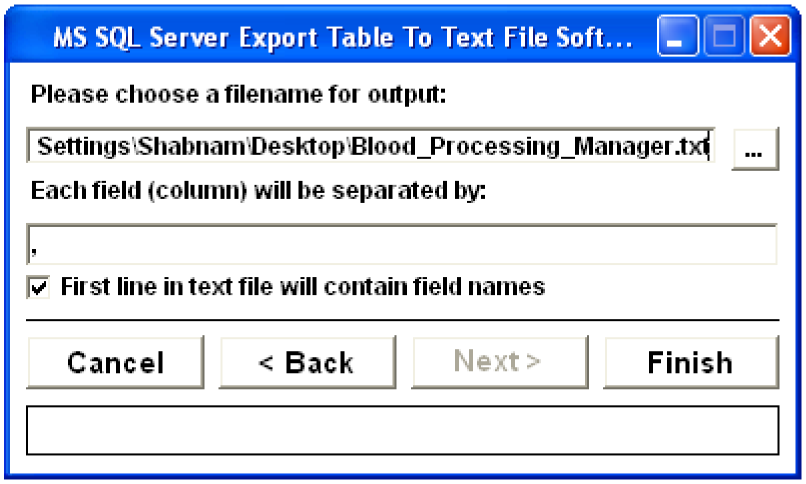
bm_id	bm_name	sex
6	Deepa	F
36	Mehrab	M
47	Urmi	F
74	Dinar	M

Which fields would you like to export to a text file?

bm_id
bm_name
sex

Figure 3.1.5

4. We selected a particular delimiter to separate each field of the tables.



Please choose a filename for output:

Settings\Shabnam\Desktop\Blood_Processing_Manager.txt

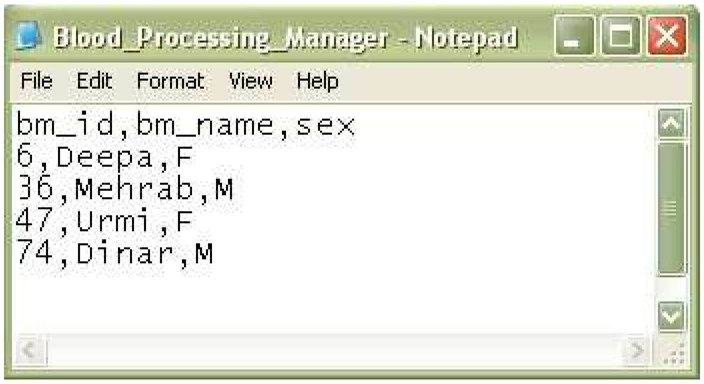
Each field (column) will be separated by:

,

☑ First line in text file will contain field names

Figure 3.1.6

5. At last, we got our required text files after clicking finish.


Blood_Processing_Manager - Notepad
File Edit Format View Help

```
bm_id,bm_name,sex
6,Deepa,F
36,Mehrab,M
47,Urmi,F
74,Dinar,M
```

Figure 3.1.7

3.2 Elementary program in C to read strings of text files

The C code which will write the output of the "MS SQL Server Export Table to Text File" software into another file in an arranged formatted way is as follows.

3.2.1 Procedure –

a) First open the source file
b) Create a destination file
c) Count total new lines of source code
d) Then repeat the file pointer to the starting position of the source file.
e) Because, the 1st line or row contains the name of the columns so if we will find out each one's length with spaces.
f) These lengths will be stored in an array.
g) Now when we start to print the values under each column then each value of a row will maintain its place because, total length of each column will be compared with it and will be printed in arranged way.
h) Before printing the last line the loop will be stopped. A new loop will be started to print the last line following previous procedure.
i) This is done only for avoiding the infinite looping.
j) After finishing the job the files will be closed.

3.2.2 The code –

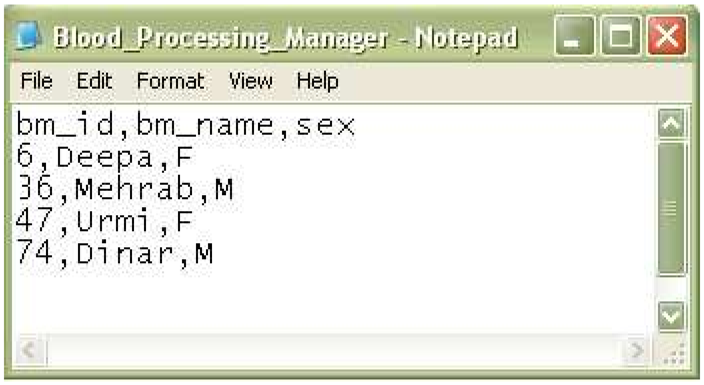
```
#include<stdio.h>
#include<conio.h>

void main()
{
  FILE *fr,*fw;
  char ch;
  int i=0,k=0,word=0,c=0,count=0,num=0,j=0,ar[20],line=0;

  fr=fopen("BR.TXT","r");          // source file will open if not then show the message
  if(fr==NULL)                     // "can't open source file"
  { puts("Can't open source file");
    exit();
  }
  fw=fopen("BRN.TXT","w");    //destination file where output will be written
  if(fw==NULL)
  {
    puts("Can't open source file");
    fclose(fr);
    exit();
  }


  while(1){
    ch=fgetc(fr);
    if(ch==EOF){     // how many lines in the code will be counted here
    k=k-1;

    break;
    }
    if(ch=='\n')
    k++;
  }

  rewind(fr);
  while(1)
  {                     //code for reading the source file and writing on destination file
    ch=fgetc(fr);

    if(ch==EOF)
         break;

    if(i==0){         // for 1st row only
         while(ch!='\n'){
             fputc(ch,fw);
             word++;
             ch=fgetc(fr);
             if(ch==','){
               ch=fgetc(fr);            // works only for 1st row.
               fputc(' ',fw);           // each column's length will be stored in an array
               word++;                  // so that values can be inserted just under their own
                                        // column
                fputc(' ',fw);
               word++;
               fputc(' ',fw);
               word++;
               fputc(' ',fw);
               word++;
               ar[c]=word;
               c++;
               word=0;

             }
         }
```

```
      if(ch=='\n'){
         i++;
         fputc(ch,fw);              //when newline is found in 1st row
           ar[c]=word;              //that means array has got its values for each column
           c=0;
           word=0;

      }
  }

      else{
        count++;

        while(ch!=','&& ch!='\n'){
          fputc(ch,fw);
          count++;                  // after 1st row other rows will be printed sequentially
          ch=fgetc(fr);             // under their own columns
          if(ch==','){
          if(ar[c]==count){
                fputc(' ',fw);
                fputc(' ',fw);
                count=0;
                c++;
                }
                if(ar[c]>count){
                    num=ar[c]-count;
                    for(j=0;j<=num;j++)
                    fputc(' ',fw);
                    count=0;
                    c++;
                    }
                    if(ar[c]<count){
                         fputc(' ',fw);
                         count=0;
                         c++;
                    }
                }

                if(ch=='\n'&& line<k){
                line++;
                  fputc(ch,fw);
                  count=0;
                  c=0;
                if(line==k){
                 ch=fgetc(fr);              // that is only for last row
```

```
                                    // it is done in individual way following same
                    while(ch!=EOF){      // procedure to stop infinite looping of code
                    fputc(ch,fw);
                    ch=fgetc(fr);
                    count++;

                        if(ch==','){
                         ch=fgetc(fr);
                        if(ar[c]==count){
                         fputc(' ',fw);
                         count=0;
                         c++;
                        }
                        if(ar[c]>count){
                         num=ar[c]-count;
                         for(j=0;j<num;j++)
                         fputc(' ',fw);
                         count=0;
                         c++;
                        }
                        if(ar[c]<count){
                        fputc(' ',fw);
                        count=0;
                        c++;
                        }
                    }

                     }
                    if(ch==EOF){
                     exit();
                     break;
                    }

                    }

                }
                  }
              }          //else
     }

      fclose(fw);
      fclose(fr);
  }
```

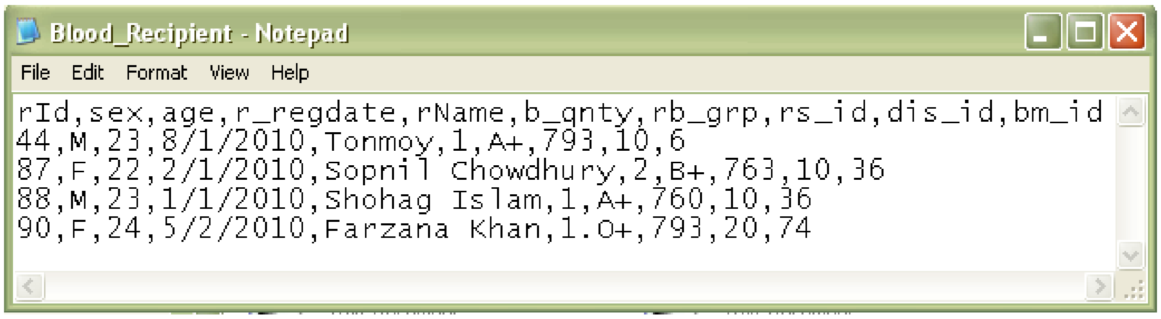
3.3 Converted Text Files

The tables which are converted into text files using software are given below----


Blood_Processing_Manager - Notepad
File Edit Format View Help

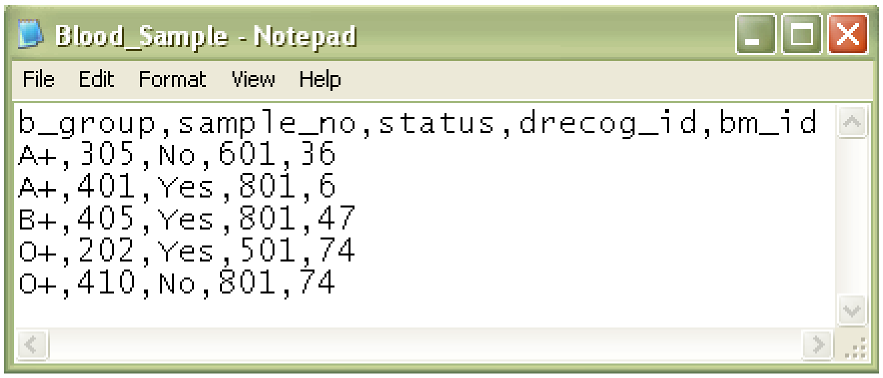
```
bm_id,bm_name,sex
6,Deepa,F
36,Mehrab,M
47,Urmi,F
74,Dinar,M
```

Figure 3.3.1



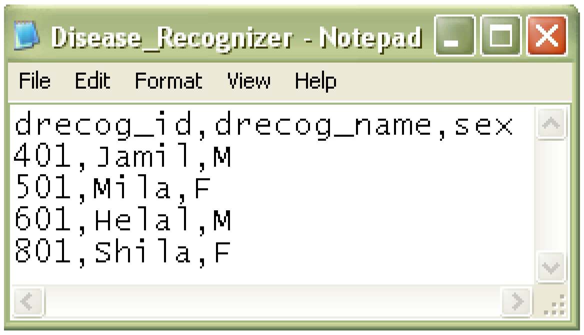
```
rId,sex,age,r_regdate,rName,b_qnty,rb_grp,rs_id,dis_id,bm_id
44,M,23,8/1/2010,Tonmoy,1,A+,793,10,6
87,F,22,2/1/2010,Sopnil Chowdhury,2,B+,763,10,36
88,M,23,1/1/2010,Shohag Islam,1,A+,760,10,36
90,F,24,5/2/2010,Farzana Khan,1.O+,793,20,74
```

Figure 3.3.2



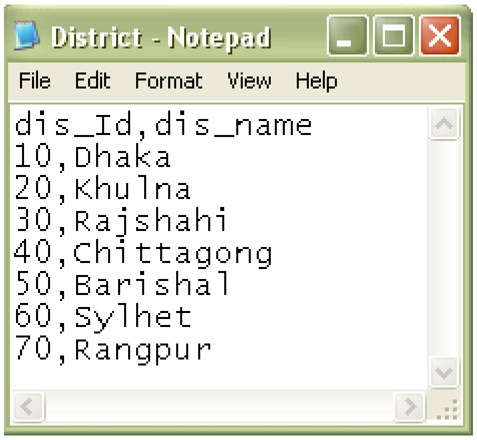
```
b_group,sample_no,status,drecog_id,bm_id
A+,305,No,601,36
A+,401,Yes,801,6
B+,405,Yes,801,47
O+,202,Yes,501,74
O+,410,No,801,74
```

Figure 3.3.3



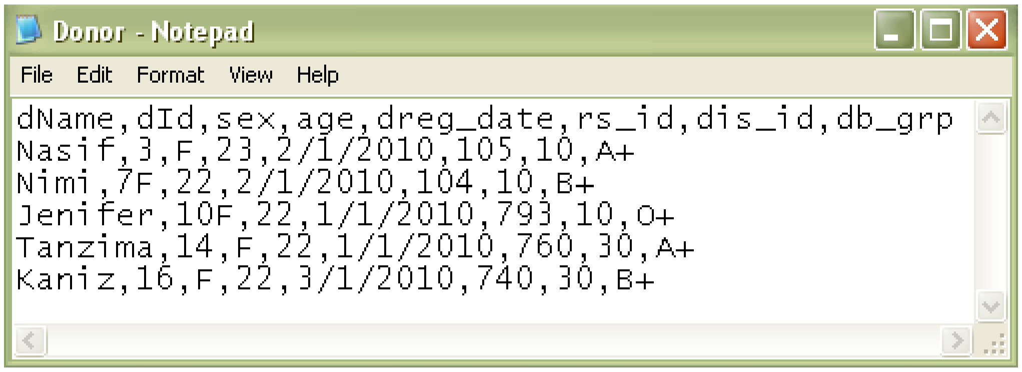
```
drecog_id,drecog_name,sex
401,Jamil,M
501,Mila,F
601,Helal,M
801,Shila,F
```

Figure 3.3.4



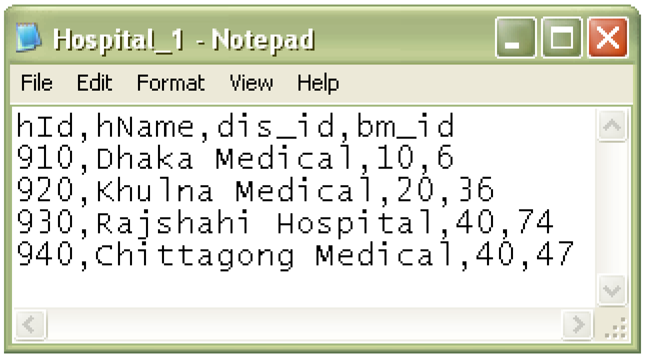
```
dis_Id,dis_name
10,Dhaka
20,Khulna
30,Rajshahi
40,Chittagong
50,Barishal
60,Sylhet
70,Rangpur
```

Figure 3.3.5



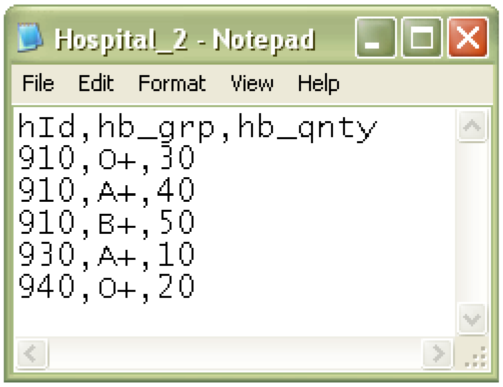
```
dName,dId,sex,age,dreg_date,rs_id,dis_id,db_grp
Nasif,3,F,23,2/1/2010,105,10,A+
Nimi,7F,22,2/1/2010,104,10,B+
Jenifer,10F,22,1/1/2010,793,10,O+
Tanzima,14,F,22,1/1/2010,760,30,A+
Kaniz,16,F,22,3/1/2010,740,30,B+
```

Figure 3.3.6



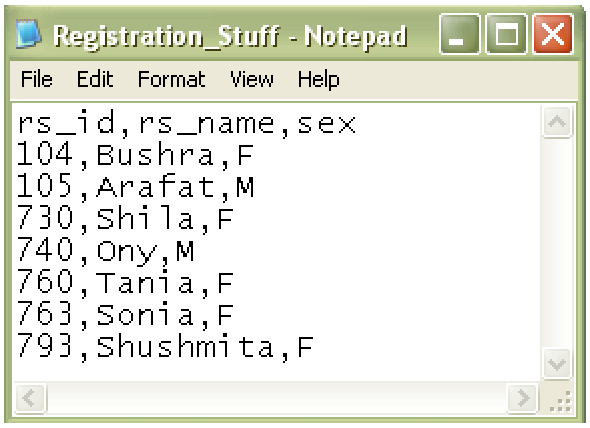
```
hId,hName,dis_id,bm_id
910,Dhaka Medical,10,6
920,Khulna Medical,20,36
930,Rajshahi Hospital,40,74
940,Chittagong Medical,40,47
```

Figure 3.3.7



```
hId,hb_grp,hb_qnty
910,O+,30
910,A+,40
910,B+,50
930,A+,10
940,O+,20
```

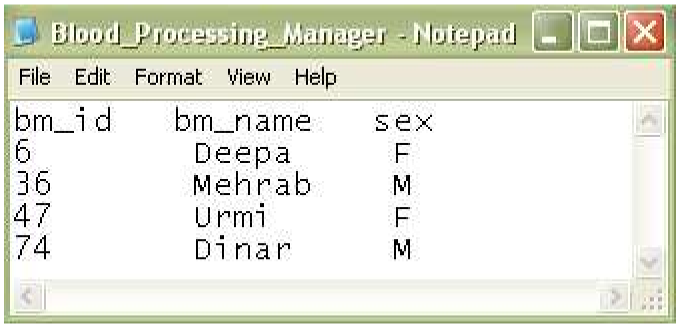
Figure 3.3.8



```
rs_id,rs_name,sex
104,Bushra,F
105,Arafat,M
730,Shila,F
740,Ony,M
760,Tania,F
763,Sonia,F
793,Shushmita,F
```

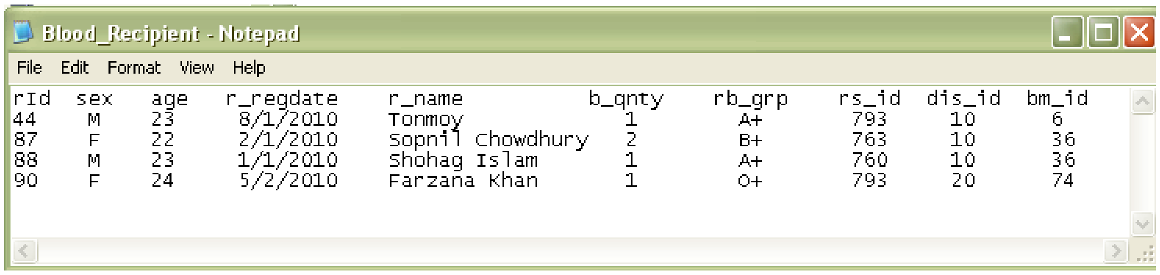
Figure 3.3.9

Final Outputs of Formatted Text Files


Blood_Processing_Manager - Notepad
File Edit Format View Help

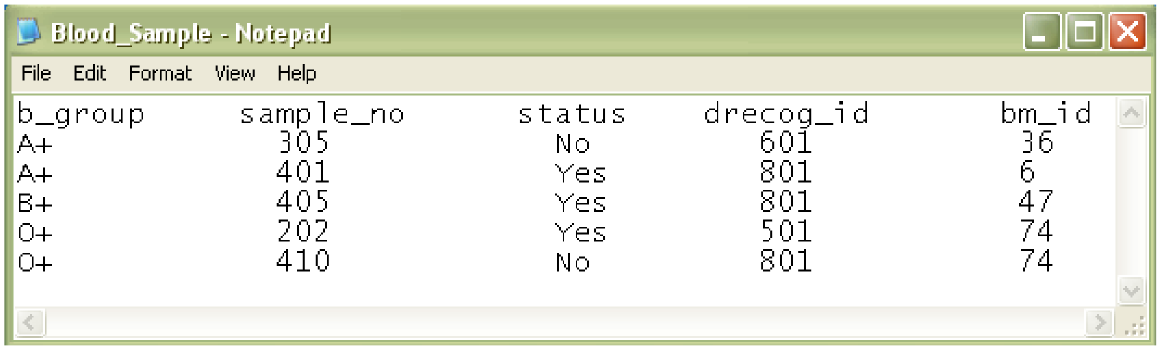
```
bm_id      bm_name    sex
6           Deepa      F
36          Mehrab     M
47          Urmi       F
74          Dinar      M
```

Figure 3.3.10



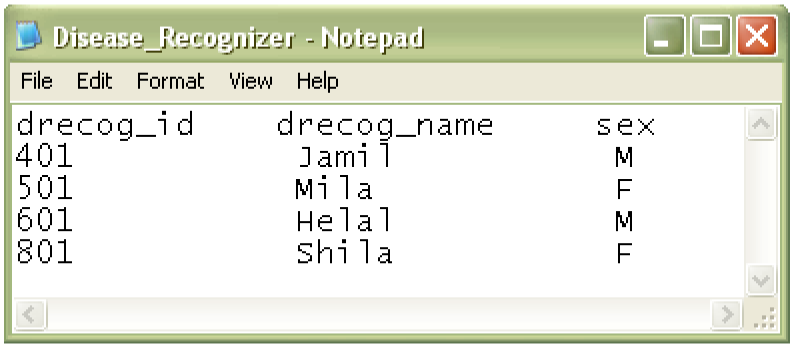
```
rId   sex   age   r_regdate    r_name           b_qnty    rb_grp    rs_id  dis_id  bm_id
44     M    23     8/1/2010    Tonmoy              1        A+       793    10      6
87     F    22     2/1/2010    Sopnil Chowdhury    2        B+       763    10      36
88     M    23     1/1/2010    Shohag Islam        1        A+       760    10      36
90     F    24     5/2/2010    Farzana Khan        1        O+       793    20      74
```

Figure 3.3.11



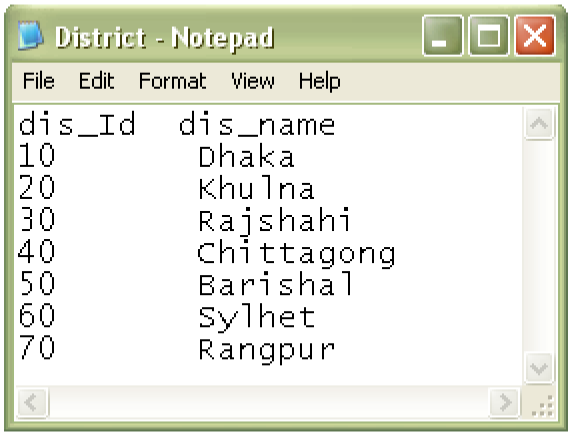
```
b_group      sample_no       status     drecog_id        bm_id
A+              305            No          601             36
A+              401            Yes         801             6
B+              405            Yes         801             47
O+              202            Yes         501             74
O+              410            No          801             74
```

Figure 3.3.12



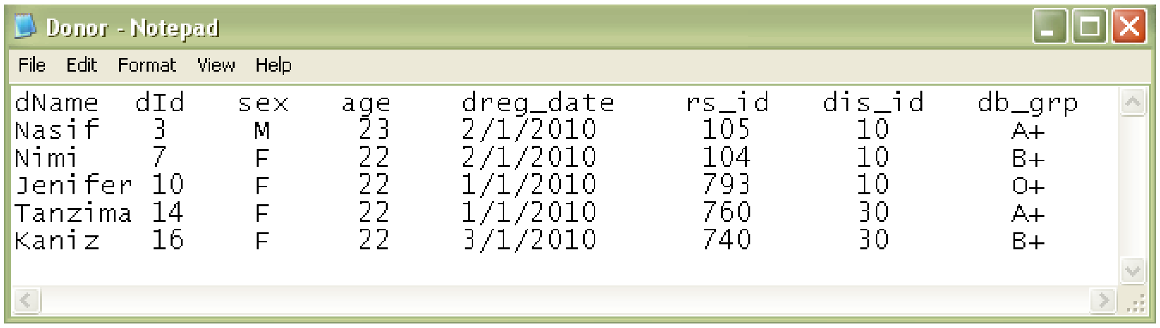
```
drecog_id     drecog_name     sex
401            Jamil           M
501            Mila            F
601            Helal           M
801            Shila           F
```

Figure 3.3.13



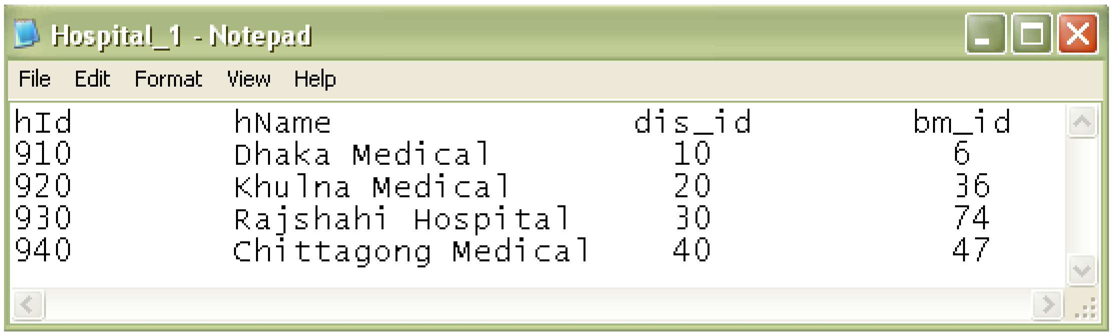
```
dis_Id   dis_name
10        Dhaka
20        Khulna
30        Rajshahi
40        Chittagong
50        Barishal
60        Sylhet
70        Rangpur
```

Figure 3.3.14



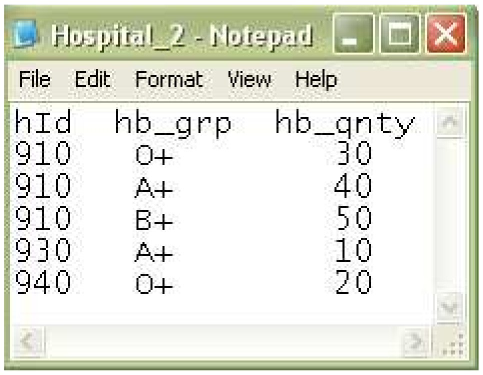
```
dName   dId   sex   age   dreg_date   rs_id   dis_id   db_grp
Nasif    3     M     23   2/1/2010     105     10       A+
Nimi     7     F     22   2/1/2010     104     10       B+
Jenifer 10     F     22   1/1/2010     793     10       O+
Tanzima 14     F     22   1/1/2010     760     30       A+
Kaniz   16     F     22   3/1/2010     740     30       B+
```

Figure 3.3.15



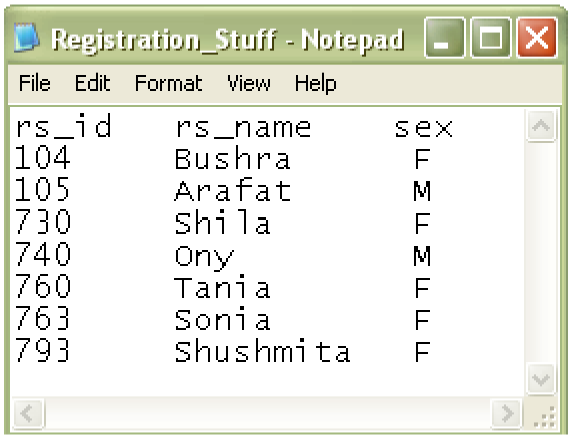
```
hId        hName                  dis_id       bm_id
910        Dhaka Medical           10           6
920        Khulna Medical          20           36
930        Rajshahi Hospital       30           74
940        Chittagong Medical      40           47
```

Figure 3.3.16



```
hId   hb_grp  hb_qnty
910    O+       30
910    A+       40
910    B+       50
930    A+       10
940    O+       20
```

Figure 3.3.17



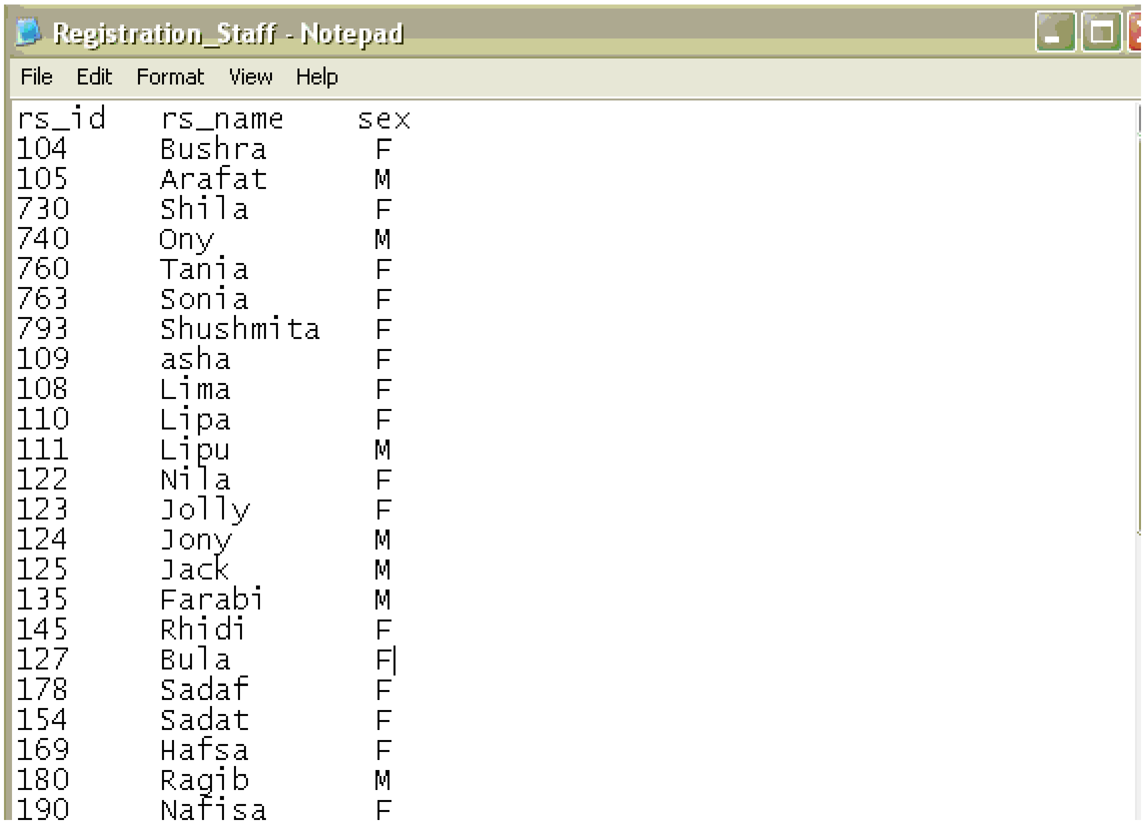
```
rs_id    rs_name     sex
104      Bushra       F
105      Arafat       M
730      Shila        F
740      Ony          M
760      Tania        F
763      Sonia        F
793      Shushmita    F
```

Figure 3.3.18

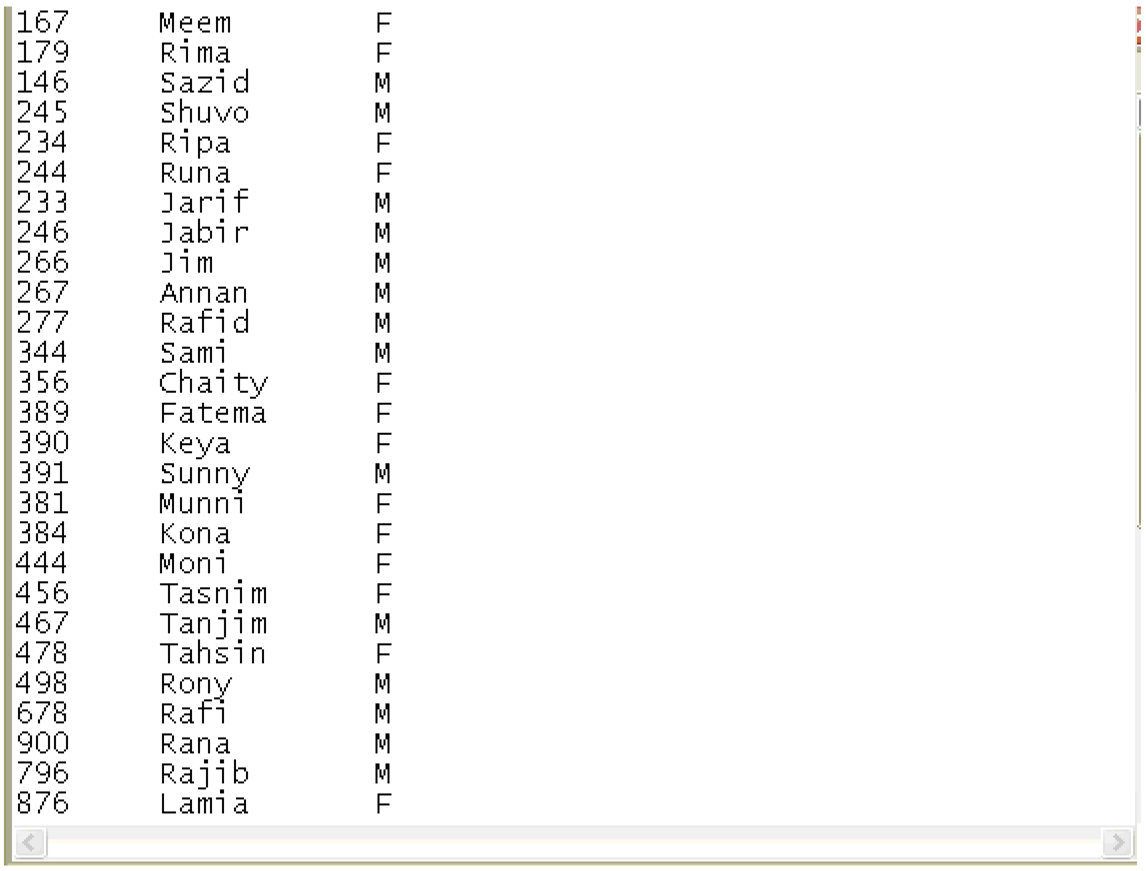
3.4 Selecting a text file for mining

In this code, a text file's strings are taken as input strings and the output will also be a text file. The total column is counted from the first row of the text file.
If the first row's first string (indicating 1st column) is numbered as **"1",** then the corresponding strings (in other rows) of this column will be numbered as—4, 7, 10, 13…….etc. The second string is numbered as **"2"** (2nd column) and corresponding strings of this column will be numbered as—5, 8, 11, 14…..etc. In this way, each of the strings will be numbered following their own columns and the column (with corresponding data strings) that is, the 2nd column (with 50 strings) which we want as output will show at the runtime. The output file will be created of this column where its corresponding values/strings will be written. The output file named as Staff_name will eventually be mined using exact and approximate string matching algorithms. Similarly other database tables which have been converted to text files in the previous section could also be mined.

Input file:

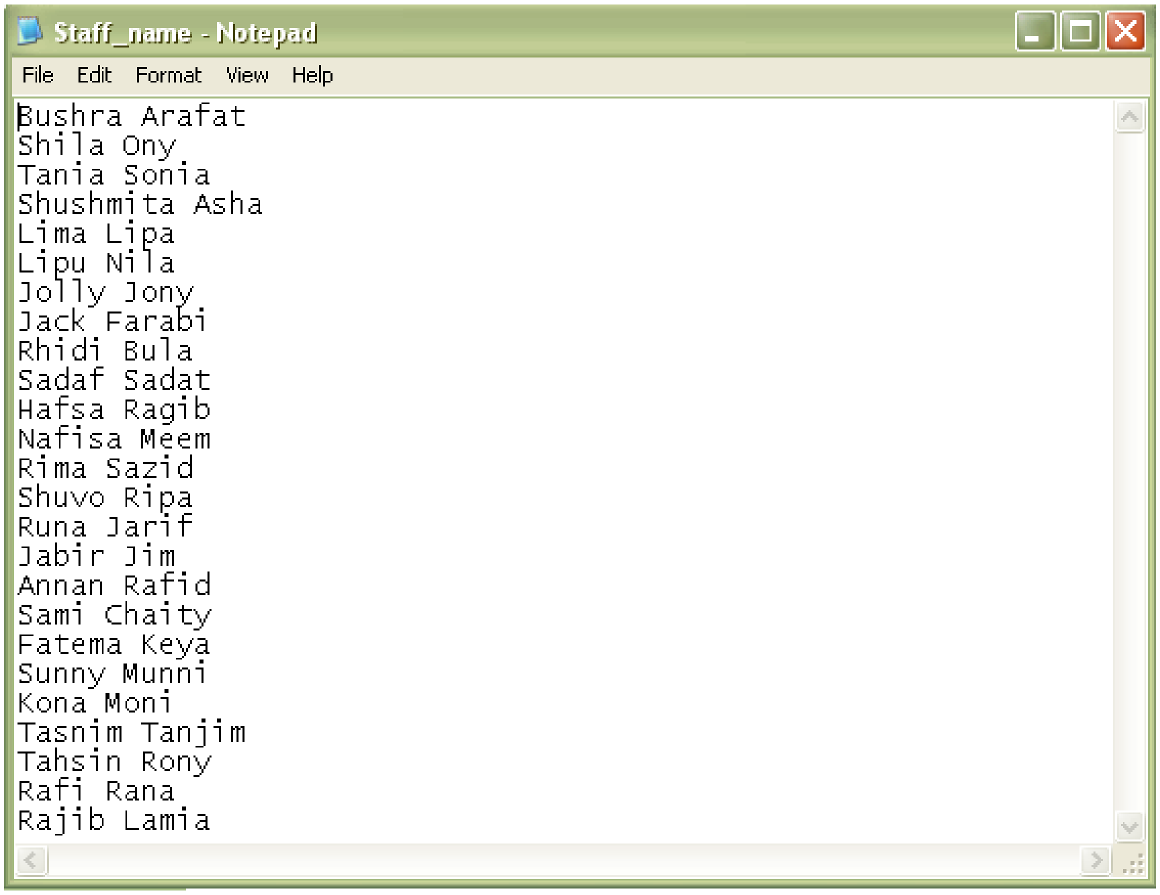


```
rs_id   rs_name    sex
104     Bushra      F
105     Arafat      M
730     Shila       F
740     Ony         M
760     Tania       F
763     Sonia       F
793     Shushmita   F
109     asha        F
108     Lima        F
110     Lipa        F
111     Lipu        M
122     Nila        F
123     Jolly       F
124     Jony        M
125     Jack        M
135     Farabi      M
145     Rhidi       F
127     Bula        F
178     Sadaf       F
154     Sadat       F
169     Hafsa       F
180     Ragib       M
190     Nafisa      F
```



```
167     Meem     F
179     Rima     F
146     Sazid    M
245     Shuvo    M
234     Ripa     F
244     Runa     F
233     Jarif    M
246     Jabir    M
266     Jim      M
267     Annan    M
277     Rafid    M
344     Sami     M
356     Chaity   F
389     Fatema   F
390     Keya     F
391     Sunny    M
381     Munni    F
384     Kona     F
444     Moni     F
456     Tasnim   F
467     Tanjim   M
478     Tahsin   F
498     Rony     M
678     Rafi     M
900     Rana     M
796     Rajib    M
876     Lamia    F
```

Output file:



```
Bushra Arafat
Shila Ony
Tania Sonia
Shushmita Asha
Lima Lipa
Lipu Nila
Jolly Jony
Jack Farabi
Rhidi Bula
Sadaf Sadat
Hafsa Ragib
Nafisa Meem
Rima Sazid
Shuvo Ripa
Runa Jarif
Jabir Jim
Annan Rafid
Sami Chaity
Fatema Keya
Sunny Munni
Kona Moni
Tasnim Tanjim
Tahsin Rony
Rafi Rana
Rajib Lamia
```

The code for selecting a text file for mining:

```
#include<stdio.h>
#include<conio.h>
#include<string.h>

void main(){

struct readf{
  char mm[20];
}r[170];

int i=0,j=1,count=0,k=0,l,flag=0,fl=0,point=0,first=0,col=0,p=0;
char c,newa[20],fname[10],dname[20],column[20];
FILE *fp;
FILE *fm;
```

```
clrscr();

printf("\n Give the input filename : ");
gets(fname);

printf("\n Give the output filename : ");
gets(dname);

fp=fopen(fname,"r");
  if(fp==NULL)
     printf("\n Error in opening file!!");

     for(k=0;k<20;k++)
              newa[k]='\0';

while(1)
 {
   c=fgetc(fp);
   if(point==1){
   point=2;
   col=j-1;
   }

   if(c==EOF){
       strcpy(r[j].mm,newa);
       j++;
       i=0;
       count=0;
       break;
   }

      if(c!=' '&&c!='\n'){
           flag=0;
           fl=0;
           newa[i]=c;
           i++;
           count++;
      }

      if(c==' '&&fl!=1){
             fl=1;
             strcpy(r[j].mm,newa);
             j++;
             i=0;
             for(k=0;k<20;k++)
             newa[k]='\0';
```

```
                count=0;
        }

         if(c=='\n'&&flag!=1){
                flag=1;
                strcpy(r[j].mm,newa);
                j++;
                for(k=0;k<20;k++)
                newa[k]='\0';
                i=0;
                count=0;
                if(point==0)
                point=1;
                p++;
        }
    }//while ends
   fclose(fp);
   printf("\n\n");
   for(i=1;i<=col;i++)
   printf("\t\t %s",r[i].mm);
   printf("\n Give the column name : ");
   gets(column);

   for(i=1;i<=col;i++){
   if(strcmp(r[i].mm,column)==0){
     point=i;
     break;
     }
   }
   printf("\n %s----position : %d",r[i].mm,point);
   fm=fopen(dname,"w");
   for(i=point;i<=j-2-(col*3);){

   if(flag==1){
   i=i+col;
   }
   if(flag==0)
   i=point+col;
   flag=1;

   fprintf(fm,"%s ",r[i].mm);
   first++;

   if(first==2){
   first=0;
   c='\n';
```

```
fputc(c,fm);
}

}

i=i+col;
fprintf(fm,"%s ",r[i].mm);
i=i+col+1;
fprintf(fm,"%s",r[i].mm);
fclose(fm);
getch();

}
```

Chapter 4

Exact String Matching Algorithms

Why are we researching on exact string matching algorithms?

A given string is called a pattern and a text contains a number of strings. Using exact string matching algorithms, we can match a string in question with the strings in the text file and get the index of exactly matched strings from the latter. In short, we are trying to mine the database text file by matching a query string exactly with a string in the text file.

4.1 Research on Exact String Matching Algorithms

4.1.1 Brute Force Algorithm

Main features

- no preprocessing phase;
- constant extra space needed;
- always shifts the window by exactly 1 position to the right;
- comparisons can be done in any order;
- searching phase in ***O***(*mn*) time complexity;
- 2*n* expected text characters comparisons.

Description

The brute force algorithm consists in checking, at all positions in the text between 0 and *n*-*m*, whether an occurrence of the pattern starts there or not. Then, after each attempt, it shifts the pattern by exactly one position to the right.

The brute force algorithm requires no preprocessing phase, and a constant extra space in addition to the pattern and the text. During the searching phase the text character comparisons can be done in any order. The time complexity of this searching phase is ***O***(*mn*) (when searching for $a^{m-1}b$ in a^n for instance). The expected number of text character comparisons is 2*n*. [3]

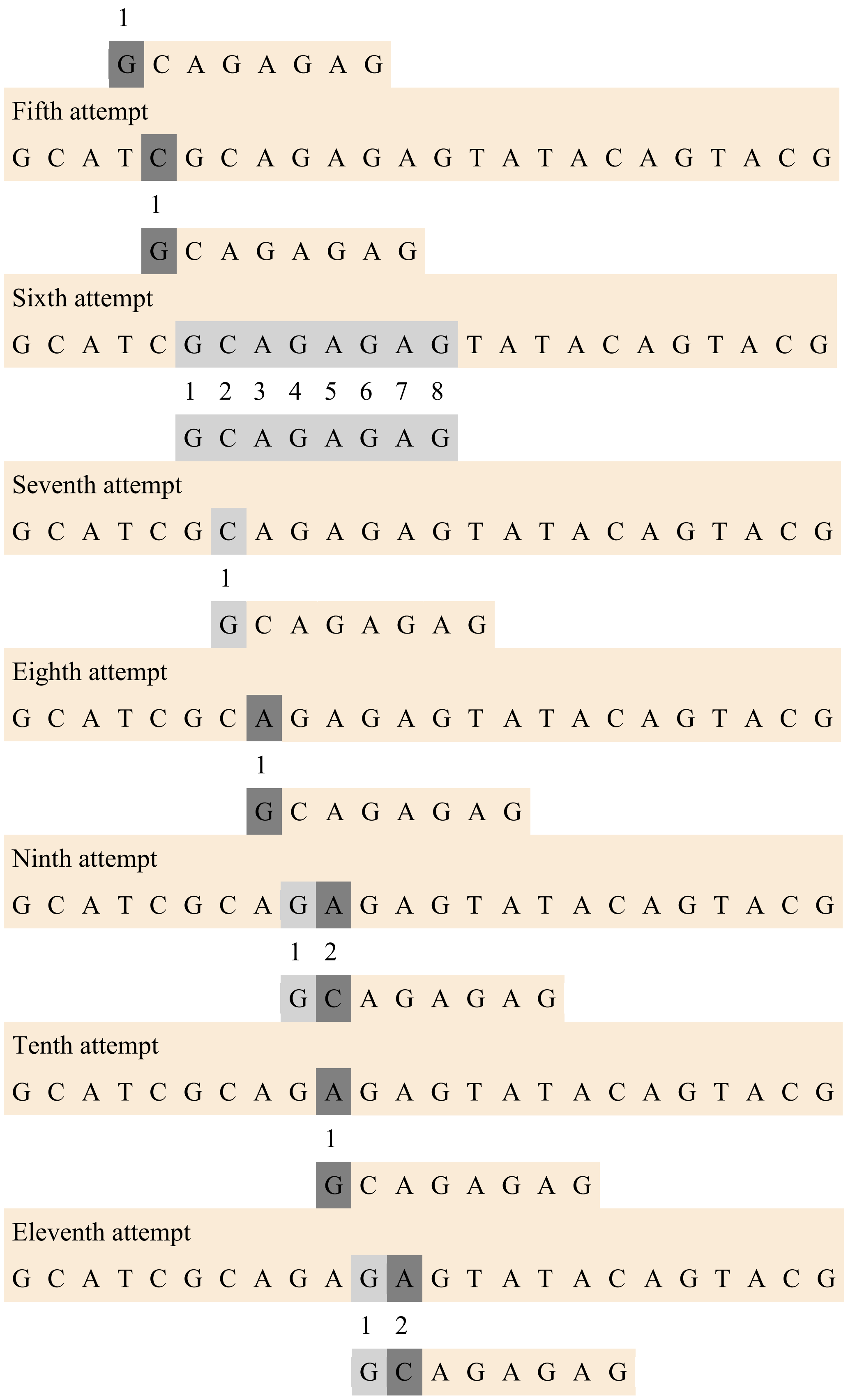
Brute Force Algorithm

```
for i=0 to i=n-m
    j=0
    while j<m and t[i+j] equal to p[j]
        j++
        if j equal to m
            return i      //return the index of main string
            printf(" Match found ");
otherwise
    printf(" No match found ");
```

Here,
t[] is used to keep the characters of a file.
p[] is the array where given pattern is kept
n= length of string kept in array t
m= length of string kept in array p

BRUTE FORCE Dry Run:

```
First attempt
G C A T C G C A G A G A G T A T A C A G T A C G
1 2 3 4
G C A G A G A G
Second attempt
G C A T C G C A G A G A G T A T A C A G T A C G
  1
  G C A G A G A G
Third attempt
G C A T C G C A G A G A G T A T A C A G T A C G
    1
    G C A G A G A G
Fourth attempt
G C A T C G C A G A G A G T A T A C A G T A C G
```


1
G C A G A G A G
Fifth attempt
G C A T C G C A G A G A G T A T A C A G T A C G
1
G C A G A G A G
Sixth attempt
G C A T C G C A G A G A G T A T A C A G T A C G
1 2 3 4 5 6 7 8
G C A G A G A G
Seventh attempt
G C A T C G C A G A G A G T A T A C A G T A C G
1
G C A G A G A G
Eighth attempt
G C A T C G C A G A G A G T A T A C A G T A C G
1
G C A G A G A G
Ninth attempt
G C A T C G C A G A G A G T A T A C A G T A C G
1 2
G C A G A G A G
Tenth attempt
G C A T C G C A G A G A G T A T A C A G T A C G
1
G C A G A G A G
Eleventh attempt
G C A T C G C A G A G A G T A T A C A G T A C G
1 2
G C A G A G A G

Twelfth attempt

```
G C A T C G C A G A G A G T A T A C A G T A C G
                      1
                      G C A G A G A G
```

Thirteenth attempt

```
G C A T C G C A G A G A G T A T A C A G T A C G
                        1 2
                        G C A G A G A G
```

Fourteenth attempt

```
G C A T C G C A G A G A G T A T A C A G T A C G
                          1
                          G C A G A G A G
```

Fifteenth attempt

```
G C A T C G C A G A G A G T A T A C A G T A C G
                            1
                            G C A G A G A G
```

Sixteenth attempt

```
G C A T C G C A G A G A G T A T A C A G T A C G
                              1
                              G C A G A G A G
```

Seventeenth attempt

```
G C A T C G C A G A G A G T A T A C A G T A C G
                                1
                                G C A G A G A G
```

4.1.2 Morris-Pratt Algorithm

Main features

1. performs the comparisons from left to right;
2. preprocessing phase in O(m) space and time complexity;
3. searching phase in O(n+m) time complexity (independent from the alphabet size);
4. performs at most 2n-1 information gathered during the scan of the text;
5. delay bounded by m.

Description

The design of the Morris-Pratt algorithm follows a tight analysis of the Brute Force algorithm, and especially on the way this latter wastes the information gathered during the scan of the text.

Let us look more closely at the brute force algorithm. It is possible to improve the length of the shifts and simultaneously remember some portions of the text that match the pattern. This saves comparisons between characters of the pattern and characters of the text and consequently increases the speed of the search.

Consider an attempt at a left position j on y, that is when the window is positioned on the text factor y[j .. j+m-1]. Assume that the first mismatch occurs between x[i] and y[i+j] with 0 < i < m. Then, x[0..i-1] = y[j .. i+j-1] = u and a = x[i] not equal to y[i+j]=b.

When shifting, it is reasonable to expect that a prefix v of the pattern matches some suffix of the portion u of the text. The longest such prefix v is called the border of u (it occurs at both ends of u). This introduces the notation: let mpNext[i] be the length of the longest border of x[0 .. i-1] for 0 < i <= m. Then, after a shift, the comparisons can resume between characters c=x[mpNext[i]] and y[i+j]=b without missing any occurrence of x in y, and avoiding a backtrack on the text (see figure). The value of mpNext[0] is set to -1.

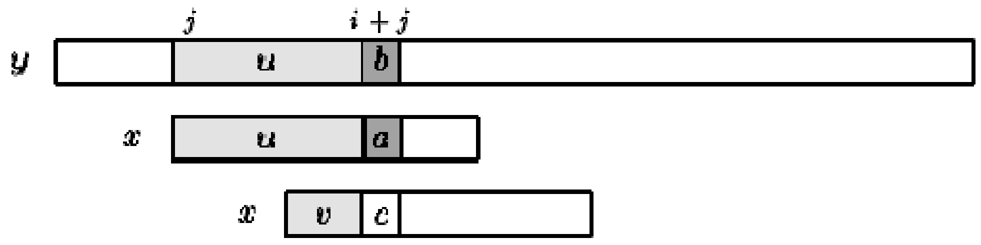


Figure: Shift in the Morris-Pratt algorithm (v border of u).

The table mpNext can be computed in O(m) space and time before the searching phase, applying the same searching algorithm to the pattern itself, as if x=y.

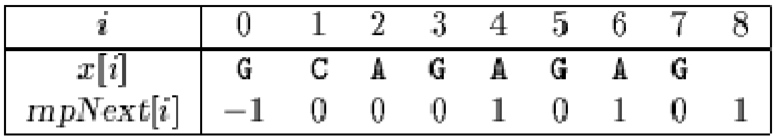
Then the searching phase can be done in O(m+n) time. The Morris-Pratt algorithm performs at most 2n-1 text character comparisons during the searching phase. The delay (maximal number of comparisons for a single text character) is bounded by m. [4]

Morris-Pratt Algorithm:

```
/* Preprocessing */

   i = 0;
   j = mpNext[0] = -1;
   while (i < m) {
   while (j > -1 && x[i] != x[j])
   j = mpNext[j];
   mpNext[++i] = mpNext[++j];
  }

/* Searching */
i = j = 0;
while (j < n) {
  while (i > -1 && x[i] != y[j])
    i = mpNext[i];
  i++;
  j++;
  if (i >= m) {
    OUTPUT(j - i);
    i = mpNext[i];
  }
}
```

y[] = main array where input is taken from a text file (length n)
x[] = given pattern of length m

The Preprocessing Table—



i	0	1	2	3	4	5	6	7	8
$x[i]$	G	C	A	G	A	G	A	G	
$mpNext[i]$	-1	0	0	0	1	0	1	0	1

The *mpNext* table

Morris-Pratt Dry Run:

First attempt

```
G C A T C G C A G A G A G T A T A C A G T A C G
1 2 3 4
G C A G A G A G
```

Shift by: 3 (*i*-*mpNext*[*i*]=3-0)

Second attempt

```
G C A T C G C A G A G A G T A T A C A G T A C G
      1
      G C A G A G A G
```

Shift by: 1 (*i*-*mpNext*[*i*]=0- -1)

Third attempt

```
G C A T C G C A G A G A G T A T A C A G T A C G
        1
        G C A G A G A G
```

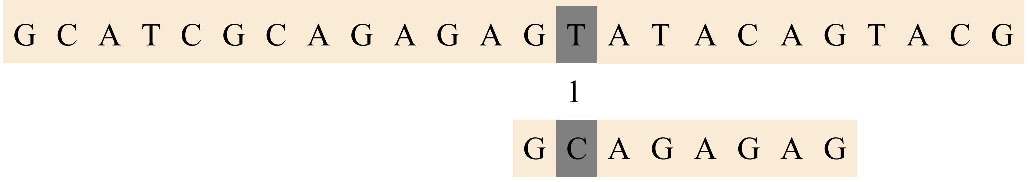
Shift by: 1 (*i*-*mpNext*[*i*]=0- -1)

Fourth attempt

```
G C A T C G C A G A G A G T A T A C A G T A C G
          1 2 3 4 5 6 7 8
          G C A G A G A G
```

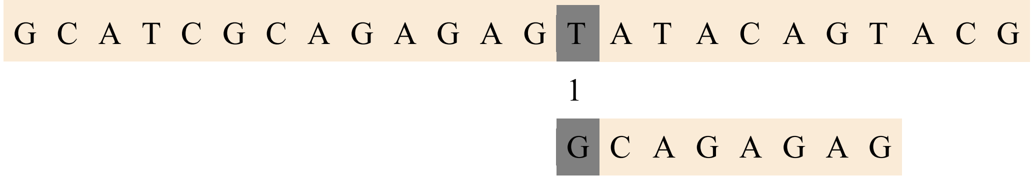
Shift by: 7 (*i*-*mpNext*[*i*]=8-1)

Fifth attempt



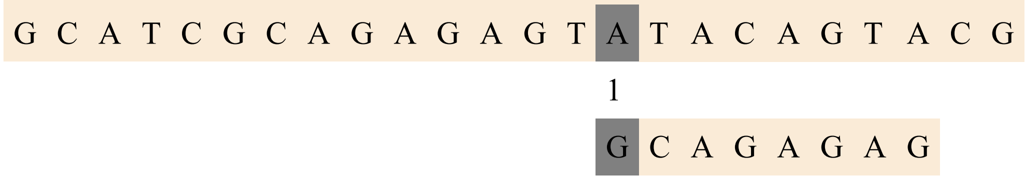
Shift by: 1 (*i*-*mpNext*[*i*]=1-0)

Sixth attempt



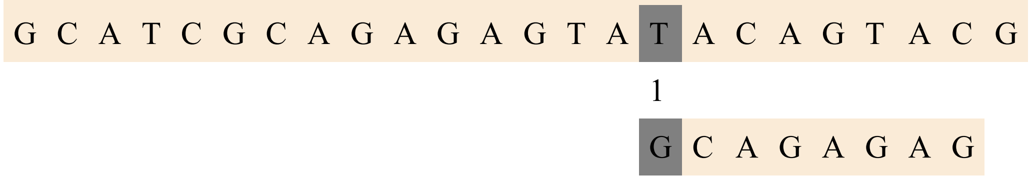
Shift by: 1 (*i*-*mpNext*[*i*]=0- -1)

Seventh attempt

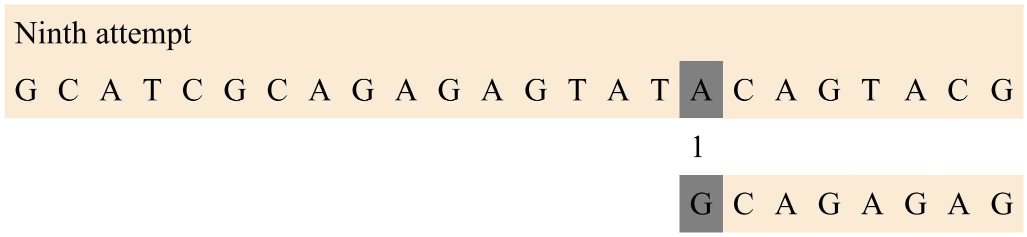


Shift by: 1 (*i*-*mpNext*[*i*]=0- -1)

Eighth attempt



Shift by: 1 (*i*-*mpNext*[*i*]=0- -1)



Shift by: 1 (i-mpNext[i]=0- -1)

4.1.3 Knuth-Morris-Pratt Algorithm

Main features

- performs the comparisons from left to right;
- preprocessing phase in ***O***(*m*) space and time complexity;
- searching phase in ***O***(*n*+*m*) time complexity (independent from the alphabet size);

Description

The design of the Knuth-Morris-Pratt algorithm follows a tight analysis of the Morris-Pratt algorithm. Let us look more closely at the Morris-Pratt algorithm. It is possible to improve the length of the shifts.

Consider an attempt at a left position *j*, that is when the the window is positioned on the text factor *y*[*j* .. *j*+*m*-1]. Assume that the first mismatch occurs between *x*[*i*] and *y*[*i*+*j*] with 0 < i < *m*. Then, *x*[0 .. *i*-1] = *y*[*j* .. *i*+*j*-1] =u and *a* = *x*[*i*] , *y*[*i*+*j*]=*b*.

When shifting, it is reasonable to expect that a prefix *v* of the pattern matches some suffix of the portion *u* of the text. Moreover, if we want to avoid another immediate mismatch, the character following the prefix *v* in the pattern must be different from *a*. The longest such prefix *v* is called the ***tagged border*** of *u* (it occurs at both ends of *u* followed by different characters in *x*).

This introduces the notation: let *kmpNext*[*i*] be the length of the longest border of *x*[0 .. *i*-1] followed by a character *c* different from *x*[*i*] and -1 if no such tagged border exits, for 0 < *i* ⩽ *m*. Then, after a shift, the comparisons can resume between characters *x*[*kmpNext*[*i*]] and *y*[*i*+*j*] without missing any occurrence of *x* in *y*, and avoiding a backtrack on the text . The value of *kmpNext*[0] is set to -1.

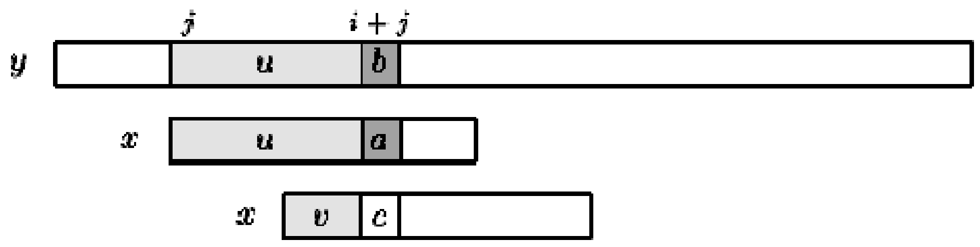


Figure 4.1.3.1 : Shift in the Knuth-Morris-Pratt algorithm (v border of u).

The table *kmpNext* can be computed in ***O***(*m*) space and time before the searching phase, applying the same searching algorithm to the pattern itself, as if *x*=*y*.

The searching phase can be performed in ***O***(*m*+*n*) time. The Knuth-Morris-Pratt algorithm performs at most 2*n*-1 text character comparisons during the searching phase. [5]

Knuth-Morris-Pratt Algorithm:

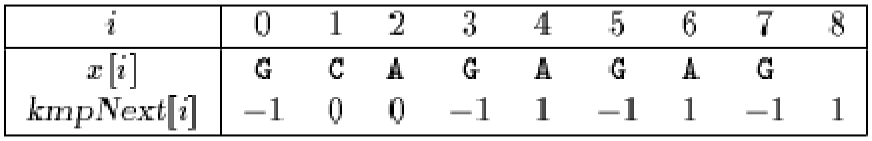
```
/* Preprocessing */

  i = 0;
 j = kmpNext[0] = -1;
  while (i < m) {
     while (j > -1 && x[i] != x[j])
      j = kmpNext[j];
   i++;
   j++;
   if (x[i] == x[j])
      kmpNext[i] = kmpNext[j];
   else
      kmpNext[i] = j;
}
```

```
/* Searching */

i = j = 0;
while (j < n) {
   while (i > -1 && x[i] != y[j])
      i = kmpNext[i];
   i++;
```

```
    j++;
    if (i >= m) {
      OUTPUT(j - i);
      i = kmpNext[i];
    }
  }
```

Here , y[] = main array where input is taken from a text file (length n)
x[] = given pattern of length m

The Preprocessing Table—



i	0	1	2	3	4	5	6	7	8
$x[i]$	G	C	A	G	A	G	A	G	
$kmpNext[i]$	−1	0	0	−1	1	−1	1	−1	1

The *kmpNext* table

Knuth-Morris-Pratt Dry Run:

First attempt

G C A T C G C A G A G A G T A T A C A G T A C G

1 2 3 4

G C A G A G A G

Shift by: 4 (*i*-*kmpNext*[*i*]=3- -1)

Second attempt

G C A T C G C A G A G A G T A T A C A G T A C G

1

G C A G A G A G

Shift by: 1 (*i*-*kmpNext*[*i*]=0- -1)

```
Third attempt
G C A T C G C A G A G A G T A T A C A G T A C G
          1 2 3 4 5 6 7 8
          G C A G A G A G
```

Shift by: 7 (*i*-*kmpNext*[*i*]=8-1)

```
Fourth attempt
G C A T C G C A G A G A G T A T A C A G T A C G
                          1
                        G C A G A G A G
```

Shift by: 1 (*i*-*kmpNext*[*i*]=1-0)

```
Fifth attempt
G C A T C G C A G A G A G T A T A C A G T A C G
                          1
                          G C A G A G A G
```

Shift by: 1 (*i*-*kmpNext*[*i*]=0- -1)

```
Sixth attempt
G C A T C G C A G A G A G T A T A C A G T A C G
                            1
                            G C A G A G A G
```

Shift by: 1 (*i*-*kmpNext*[*i*]=0- -1)

```
Seventh attempt
G C A T C G C A G A G A G T A T A C A G T A C G
                              1
                              G C A G A G A G
```

Shift by: 1 (*i*-*kmpNext*[*i*]=0- -1)

Eighth attempt

G	C	A	T	C	G	C	A	G	A	G	A	G	T	A	T	A	C	A	G	T	A	C	G
																1							
																G	C	A	G	A	G	A	G

Shift by: 1 (*i*-*kmpNext*[*i*]=0- -1)

4.1.4 Boyer-Moore Algorithm

The Boyer-Moore algorithm is considered the most efficient string-matching algorithm in usual applications, for example, in text editors and commands substitutions. The reason is that it works the fastest when the alphabet is moderately sized and the pattern is relatively long.

The algorithm scans the characters of the pattern from right to left beginning with the rightmost character. During the testing of a possible placement of pattern P against text T, a mismatch of text character T[i] = c with the corresponding pattern character P[j] is handled as follows: If c is not contained anywhere in P, then shift the pattern P completely past T[i]. Otherwise, shift P until an occurrence of character c in P gets aligned with T[i].

This technique is likely to avoid lots of needless comparisons by significantly shifting pattern relative to text. [6]

Boyer-Moore algorithm:

BOYER_MOORE_MATCHER (T, P)

Input: Text T with n characters and Pattern P with m characters

Output: Index of the first substring of T matching P

```
1. Compute function last
2. i ← m-1
3. j ← m-1
4. Repeat
5.     If P[j] = T[i] then
6.        if j=0 then
7.           return i // we have a match
8.        else
9.           i ← i -1
10.          j ← j -1
11.    else
12.       i ← i + m - Min(j, 1 + last[T[i]])
13.       j ← m -1
14.until i > n -1
15.Return "no match"
```

Illustration of the algorithm:

Say, the inputs are as follow:

The text, T is VSSHABNAMOUH and its length, n=12

Index no =>	0	1	2	3	4	5	6	7	8	9	10	11
Text, T	V	S	S	H	A	B	N	A	M	O	U	H

The pattern, P is HABNAMO and its length, m=7

Index no	0	1	2	3	4	5	6
Pattern, P	H	A	B	N	A	M	O

So, according to the algorithm the output should be 3 which is the index number of the text (T), from where the pattern matches with the text. If we add the pattern length -1 with this index number (which is 3) of the text (T) then we will get the matching string.

Step 1:

last function:

We define a function last(c) that takes a character c from the alphabet (text) and specifies how far may shift the pattern P if a character equal to c is found in the text that does not match the pattern. Using the following rule, the last function will be calculated:

last (c) = { index of the last occurrence of c in pattern P, if c is in P.
-1, otherwise

So, the computation of last function will be like this for the input pattern and text:

V	S	S	H	A	B	N	A	M	O	U	H
-1	-1	-1	0	4	2	3	4	5	6	-1	0

Step 2:

i = m-1
=7-1
= 6

Step 3:

j=m-1
=7-1
=6

Iteration 1:

Step 5: P[6]=T[6]? No.
go to step 12:

i= i+m-Min(j,1+last(T[i])) //Min function find the minimum between 2 numbers
=6+7-Min(6,1+last(T[6]))
=6+7-Min(6,1+3)
=6+7-Min(6,4)
=6+7-4
=9
So, i=9, j=m-1=7-1=6

return to step 5: (**Iteration 2**)
P[6]=T[9]? true
step 6: j=0? false
go to step 9,10
i=i-1=9-1=8
j=j-1=6-1=5

Iteration 3:

Step5: P[5]=T[8]? true

```
  step 6:j=0? false
  go to step 9,10
  i=i-1=8-1=7
  j=j-1=5-1=4
```

Iteration 4:

Step5: P[4]=T[7]? true

```
  step 6:j=0? false
  go to step 9,10
  i=i-1=7-1=6
  j=j-1=4-1=3
```

Iteration 5:

Step5: P[3]=T[6]? true

```
  step 6:j=0? false
  go to step 9,10
  i=i-1=6-1=5
  j=j-1=3-1=2
```

Iteration 6:

Step5: P[2]=T[5]? true

```
  step 6:j=0? false
  go to step 9,10
  i=i-1=5-1=4
  j=j-1=2-1=1
```

Iteration 7:

Step5: P[1]=T[4]? true

```
  step 6:j=0? false
  go to step 9,10
  i=i-1=4-1=3
```

j=j-1=1-1=0

Iteration 8:

Step5: P[0]=T[3]? true
step 6:j=0? true
step 7: return i=3

Summary

- The computation of the last function takes O(m+|∑|) time and actual search takes O(mn) time. Therefore the worst case running time of Boyer-Moore algorithm is O(nm + |∑|). Implies that the worst-case running time is quadratic, in case of n = m, the same as the naïve algorithm.

- Boyer-Moore algorithm is extremely fast on large alphabet (relative to the length of the pattern).

- The payoff is not as for binary strings or for very short patterns.

- For binary strings Knuth-Morris-Pratt algorithm is recommended.

- For the very shortest patterns, the naive algorithm may be better. [6]

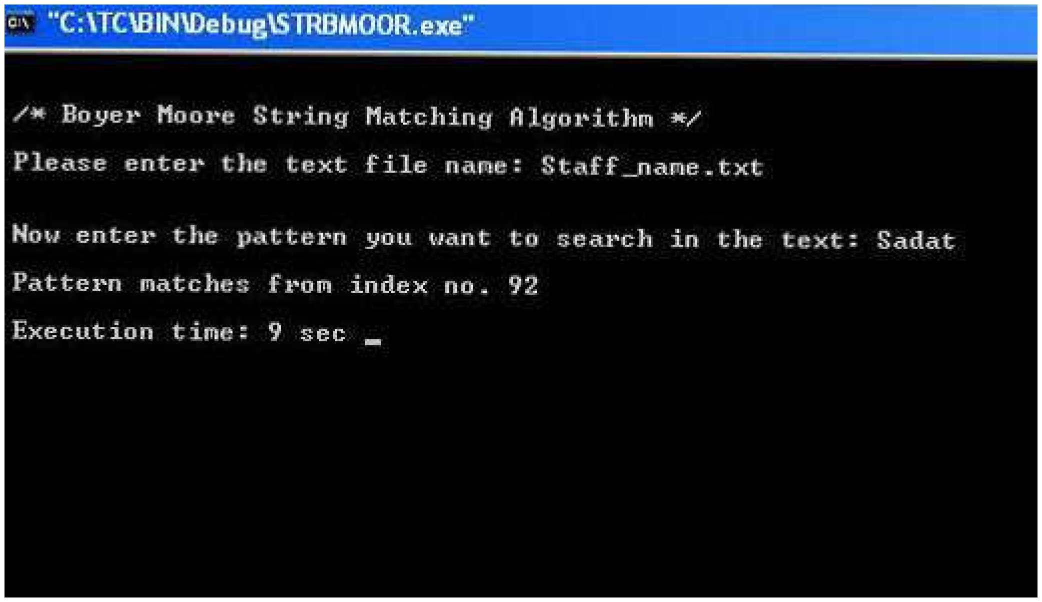
4.1.5 Analysis

There are numerous exact string matching algorithms that have almost similar performance characteristics. Which algorithm is best depends on the length of the pattern being searched for, the number of letters in the alphabet and the particular architecture where the program is being executed.

The measure of text character inspections is an objective parameter of the performance of string matching algorithms. In general, the Boyer Moore algorithm is extremely fast on large alphabets (relatively to the length of the pattern) among the four exact string matching algorithms that we have demonstrated, whereas Knuth Morris Pratt is the fastest on small alphabets. Morris Pratt algorithm is slower than Knuth Morris Pratt and Brute Force is the slowest among the four algorithms.For an example,if we take small text "VSSHABNAMOUH" and small pattern " HABNAMO",we find that Brute Force Algorithm requires 6 iterations, Morris-Pratt Algorithm requires 5 iterations, Knuth-Morris-Pratt Algorithm requires 4 iterations and Boyer-Moore Algorithm requires 8 iterations.So, Knuth-Morris-Pratt Algorithm is the fastest among the other algorithms on text having small alphabets.

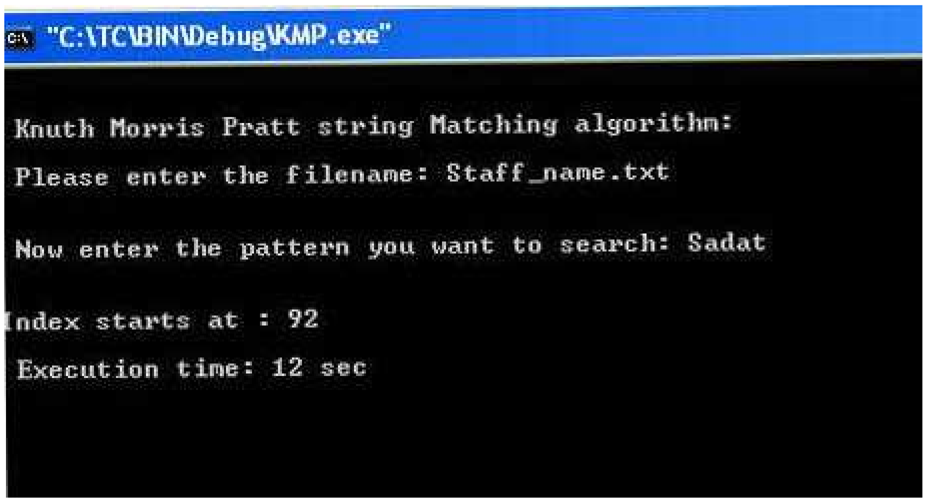
In order to evaluate the practical performances of string matching algorithms, we have implemented them in C in a homogeneous way on the same text file (Staff_name.txt) having 50 strings to make the comparisons significant. Snapshots of the results are given below according to their execution time (from the fastest to the slowest) :

1.Boyer Moore Algorithm:



```
"C:\TC\BIN\Debug\STRBMOOR.exe"

/* Boyer Moore String Matching Algorithm */
Please enter the text file name: Staff_name.txt

Now enter the pattern you want to search in the text: Sadat
Pattern matches from index no. 92
Execution time: 9 sec _
```

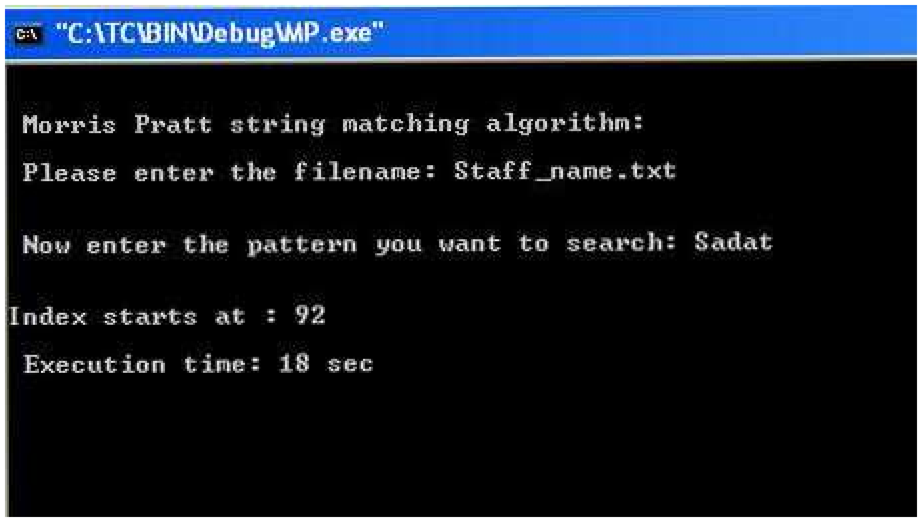
2.Knuth Morris Pratt Algorithm:



"C:\TC\BIN\Debug\KMP.exe"

```
Knuth Morris Pratt string Matching algorithm:
Please enter the filename: Staff_name.txt

Now enter the pattern you want to search: Sadat

Index starts at : 92
Execution time: 12 sec
```

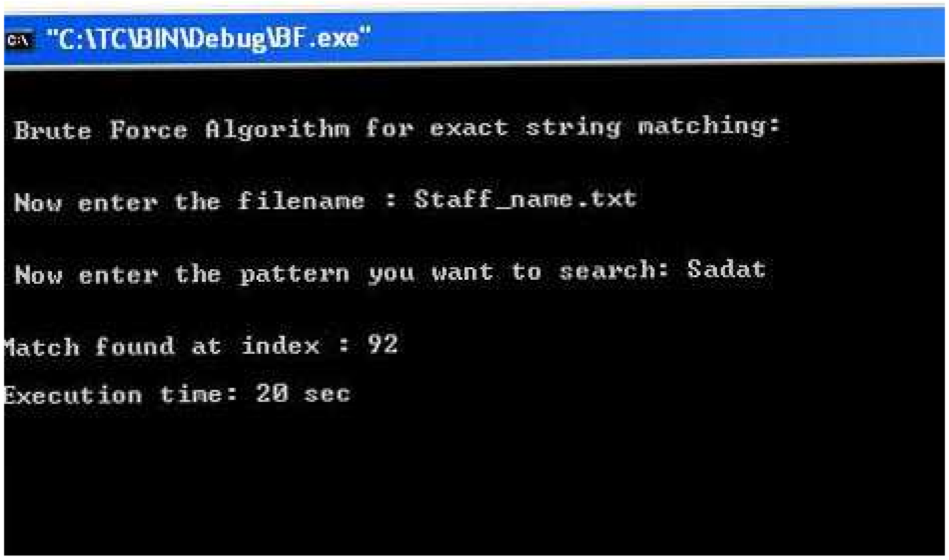
3.Morris Pratt Algorithm:



"C:\TC\BIN\Debug\MP.exe"

```
Morris Pratt string matching algorithm:
Please enter the filename: Staff_name.txt

Now enter the pattern you want to search: Sadat

Index starts at : 92
Execution time: 18 sec
```

4.Brute Force Algorithm:



```
"C:\TC\BIN\Debug\BF.exe"

 Brute Force Algorithm for exact string matching:

 Now enter the filename : Staff_name.txt

 Now enter the pattern you want to search: Sadat

Match found at index : 92

Execution time: 20 sec
```

Chapter 5:

Approximate String Matching Algorithms

5.1 Algorithm on finding Edit Distance between two strings

How will edit distance help in our thesis work?

By approximate string matching algorithms, we find the nearest matched string from a text file in response to a query string. Edit distance helps to find out the distance between the query string and the nearest matched string from the text file. Most commonly the edit operations allowed for this purpose:

a) insert a character into a string
b) delete a character from a string
c) replace a character of a string by another character [8]

5.1.1 Formal Definition of String Edit Distance:

The string edit distance between two strings, ed(x,y), is the minimum number of character insertions, deletions and replacements that transforms x to y.

Example:
hello→hallo: replace e by a
hello→hell: delete o
hello→shell: delete o, insert s
Also called Levenshtein distance.

5.1.2 Algorithm:

Step	Description
1	Set n to be the length of s. Set m to be the length of t. If n = 0, return m and exit If m = 0, return n and exit Construct a matrix containing 0..m rows and 0..n columns
2	Initialize the first row to 0..n. Initialize the first column to 0..m
3	Examine each character of s (i from 1 to n).
4	Examine each character of t (j from 1 to m).
5	If s[i] equals t[j], the cost is 0. If s[i] doesn't equal t[j], the cost is 1.
6	Set cell d[i,j] of the matrix equal to the minimum of: a. The cell immediately above plus 1: d[i-1,j] + 1. b. The cell immediately to the left plus 1: d[i,j-1] + 1. c. The cell diagonally above and to the left plus the cost: d[i-1,j-1] + cost.
7	After the iteration steps (3, 4, 5, 6) are complete, the distance is found in cell d[n,m].

5.1.3 Dry Run:

This section shows how the Levenshtein distance is computed when the source string is "GUMBO" and the target string is "GAMBOL".

Steps 1 and 2

		G	U	M	B	O
	0	1	2	3	4	5
G	1					
A	2					
M	3					
B	4					
O	5					
L	6					

Stps 3 to 6 When i=1

		G	U	M	B	O
	0	1	2	3	4	5
G	1	0				
A	2	1				
M	3	2				
B	4	3				
O	5	4				
L	6	5				

Steps 3 to 6 When i = 2

		G	U	M	B	O
	0	1	2	3	4	5
G	1	0	1			
A	2	1	1			
M	3	2	2			
B	4	3	3			
O	5	4	4			
L	6	5	5			

Steps 3 to 6 When i = 3

		G	U	M	B	O
	0	1	2	3	4	5
G	1	0	1	2		
A	2	1	1	2		
M	3	2	2	1		
B	4	3	3	2		
O	5	4	4	3		
L	6	5	5	4		

Steps 3 to 6 When i = 4

		G	U	M	B	O
	0	1	2	3	4	5
G	1	0	1	2	3	
A	2	1	1	2	3	
M	3	2	2	1	2	
B	4	3	3	2	1	
O	5	4	4	3	2	
L	6	5	5	4	3	

Steps 3 to 6 When i = 5

		G	U	M	B	O
	0	1	2	3	4	5
G	1	0	1	2	3	4
A	2	1	1	2	3	4
M	3	2	2	1	2	3
B	4	3	3	2	1	2
O	5	4	4	3	2	1
L	6	5	5	4	3	2

Step 7

The distance is in the lower right hand corner of the matrix, i.e. 2. This corresponds to our intuitive realization that "GUMBO" can be transformed into "GAMBOL" by substituting "U" for "A" and adding "L" (one substitution and 1 insertion = 2 changes). [3]

5.2 Introduction to Approximate String Matching Algorithms

In computing, **approximate string matching** (often colloquially referred to as **fuzzy string searching**) is the technique of finding approximate matches to a pattern in a string.

One possible definition of the approximate string matching problem is the following: Given a pattern string $P = p_1p_2...p_m$ and a text string, $T = t_1t_2...t_n$ find a substring $T_{j`,j} = t_{j`}...t_j$ in T, which, of all substrings of T, has the smallest edit distance to the pattern P.

The most common application of approximate matchers until recently has been spell checking. With the availability of large amounts of DNA data, matching of nucleotide sequences has become an important application. Approximate matching is also used to identify pieces of music from small snatches and in spam filtering. [9]

There are different algorithms for approximate string matching. Such three algorithms are demonstrated below:

5.2.1 Brute Force algorithm for approximate string matching:

A brute-force approach would be to compute the edit distances to query object(q) from all N substrings of text(T), and then choose the substring with the minimum distance. The sequential steps are given below:

Steps:

1. Read N (=50) strings from the text file T.
2. Take an input (query object q) from the user.
3. Calculate the edit distances to query object, q from all N strings of T.
4. Find out the minimum edit distance.
5. Output will be the string which has the minimum edit distance.

Let N=5 and T is:{been, bid, moon, sun, star}
Say, q='seen'
The edit distances to query object q from all N strings of T will be respectively:
{1,4,3,2,3}
The minimum distance is: 1
Hence the output will be: been

5.2.2 Lipschitz Embeddings Algorithm

A Lipschitz embedding is defined in terms of a set R of subsets of S(the set of objects), R={A_1,A_2,……..A_k}. The subsets A_i are termed the reference sets of the embedding. Let d(o,A) be an extension of the distance function d to a subset A of S, such that d (o,A)= min{d(o,x)},where x is an element of A. An embedding with respect to R is defined as a mapping F such that
F(o) =((d(o ,A_1),d(o,A_2),…….d(o, A_k)).

To elaborate on how a query is implemented, suppose that we want to find the nearest object to a query object q. We first determine the point F(q) corresponding to q. Next, we examine the objects in the order of their distances from F(q) in the embedding space. When using a multidimensional index, this can be achieved by using an incremental nearest neighbor algorithm. Suppose that point F(a) corresponding to object a is the closest point to F(q) at a distance of ∂(F(a),F(q)). We compute the distance d(a, q) between the corresponding objects. At this point, we know that any object x with ∂(F(x),F(q)) > d(a,q) cannot be the nearest neighbor of q since the contractive property then guarantees that d(x, q) > d(a,q). Therefore, d(a,q) now serves as an upper bound on the nearest-neighbor search in the embedding space. We now find the next closest point F(b) corresponding to object b, subject to our distance constraint d(a, q).

If d(b,q) < d(a, q), then b and d(b,q) replace object a and d(a,q) as the current closest object and upper bound distance, respectively; otherwise, a and d(a,q) are retained. This search continues until encountering a point F(x) with ∂(F(x),F(q)) > d(y,q),where y is the current closest object which is now guaranteed to be the actual closest object to q. [10]

Algorithm:

Input: A text file T containing N strings (O_1, O_2,…….O_N) and a query object q.
Output: Nearest string to q with corresponding edit distance.
Steps:

1. Construct a text file R of k strings (A_1,A_2,……..A_k) by choosing randomly from N strings of T.

2. Compute F(q),the array of edit distances of the query object q to k strings of R, that is, F(q) = (d(q,A_1),d(q,A_2),……,d(q,A_k)) where, d(q,A_j) is the edit distance between q and A_j .Here,$1 \le j \le k$.

3. Compute F(O_i),the array of edit distances of each string O_i in T to k reference strings in R.
that is, F(O_i) =((d(O_i ,A_1),(d(O_i , A_2),…….d(O_i , A_k)).Here,$1 \le i \le N$.

4. Calculate $\partial(F(O_j),F(q))$,the distance between each string O_j of T to query string q in embedding space where,

$$\partial(F(O_j),F(q)) = \left(\sum_{i=1}^{k} \left((d(O_j, Ai) - d(q, Ai)) / k^{1/p} \right)^{p} \right)^{1/p}$$

where $1 \leq j \leq N$, $1 \leq i \leq k$.

5. Find the minimum value among $\partial(F(O_i),F(q))$, if the minimum is $\partial(F(O_m),F(q))$, then find $d(O_m,q)$. ($1 \leq i \leq N$).

6. For i=1 to N,

```
    if($\partial(F(O_i),F(q)) > d(O_m,q)$ )
        then edit_distance= $d(O_m,q)$ and nearest_string= $O_m$.
    else if ($d(O_i,q) < d(O_m,q)$)
        then edit_distance= $d(O_i,q)$ and nearest_string = $O_i$
```

7. Show the edit_distance and nearest_string.

5.2.3 Ball Partitioning Algorithm

The vp-tree (Vantage Point Tree) uses ball partitioning (and thus is a variant of the metric tree. In this method, we pick a pivot *p* randomly from *S* containing 50 strings/objects (*p* is termed as a vantage point; compute the median *r* of the distances of the other objects to *p*, and then divide the remaining objects into roughly equal sized subsets *S*1 and *S*2 as follows:

$$S_1 = \{o \in S \setminus \{p\} \mid d(p,o) < r\}, \quad \text{and}$$
$$S_2 = \{o \in S \setminus \{p\} \mid d(p,o) \geq r\}.$$

Thus, the objects in *S*1 are *inside* the ball of radius *r* around *p*, while the objects in *S*2 are *outside* this ball. Applying this rule recursively leads to a binary tree, where a pivot object is stored in each internal node, with the left and right subtrees corresponding to the subsets inside and outside the corresponding ball, respectively. In the leaf nodes of the tree we would store one or more objects, depending on the desired capacity.

Pivot Selection: Pivot is chosen randomly in this algorithm and in the vp-tree, the ball radius is always chosen as the median so that the two subsets are roughly equal in size.

Search: We visit the left child if and only if
$\max\{d(q,p)-r,0\} <= \epsilon$ and the right child if and only if $\max\{ r-d(q,p),0\} <= \epsilon$

Definitions of some keywords:

Radius of pivot- r
Query object- q
Pivot object- p
Edit distance of pivot and query object- ϵ
Another notation of edit distance between pivot and query- d{q,p} also. [11]

Algorithm is divided into two parts:

Creating VP-TREE:
1. Read N (=50) strings from a text file.
2. Select a pivot (**p**) randomly from N strings.
3. Compute edit-distance (**d**) of pivot from the remaining strings.
4. Values of edit-distances (from pivot to others) will be kept in an array.
5. Sort the values of the array and find out median using formula of median.
6. The median is the radius (**r**) of pivot. A flag value will be increased.
7. Take this pivot and radius as the input of VP-tree's first node.
8. Similarly, repeat steps 2-6 and find out pivot for each step if the pivot's radius is less than its parent node's radius **r** and **>= 0,** then put it on left node and if **>= r** then put it on right node. In step 2 each time N decreases by 1.
9. This process will continue until each internal node ends at leaf with a child or we can keep a cluster of strings at each leaf.

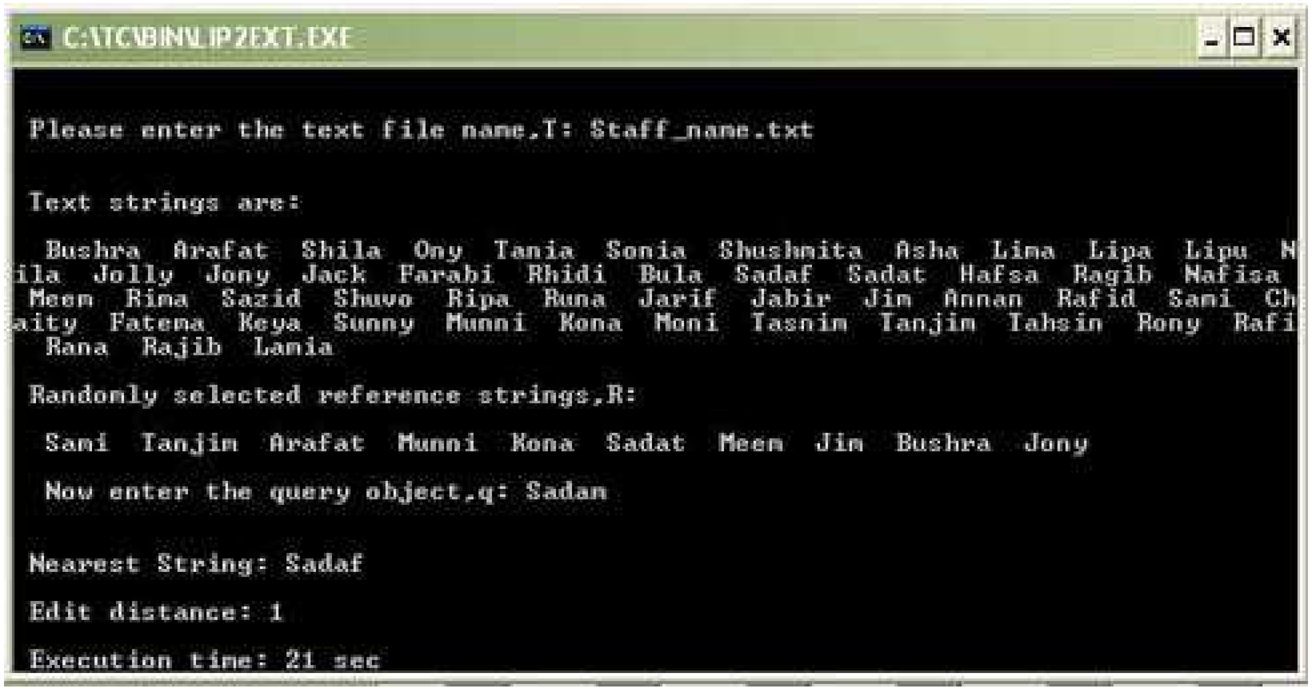
Searching Phase:
1. Take an input of query object (**q**).
2. When a value will be assigned in VP-tree (pivot and radius) then find the edit distance of pivot and query object. And check if ,
   Max {**d(q, p) - r, 0**} <= **E,** then visit left node, otherwise not.
   Max {**r – d(q, p), 0**} <= **E,** then visit right node, otherwise not.
   E= the edit-distance between pivot and query object.
3. Keep the edit-distance and the pivot in an array of structure.
4. Repeat the steps of searching phase 1-3 until we get child of each node.
5. Now find the minimum edit distance of all edit distances, and the corresponding pivot which will be the nearest neighbor of query object.

5.2.4 Analysis

Lipschitz Embedding Algorithm uses a straight forward procedure to match an approximate string to the query string. It creates reference strings, uses mathematical formula to find out minimum edit distance and the corresponding nearest string to the query string. Ball partition algorithm creates a VP tree and visits left and right nodes of the tree to find out minimum edit distance and the corresponding string which is nearest to the query string. Execution time increases with the increase of visited nodes. Brute force algorithm for approximate string matching uses most inefficient way to find out nearest string.It calculates edit distances of all the text strings from the query string and then checks all the distances to find out the minimum distance.As a result, among the three approximate string matching algorithms,the fastest is Lipschitz Embeddings Algorithm, then the Ball partitioning algorithm and the slowest is Brute Force algorithm for approximate string matching.

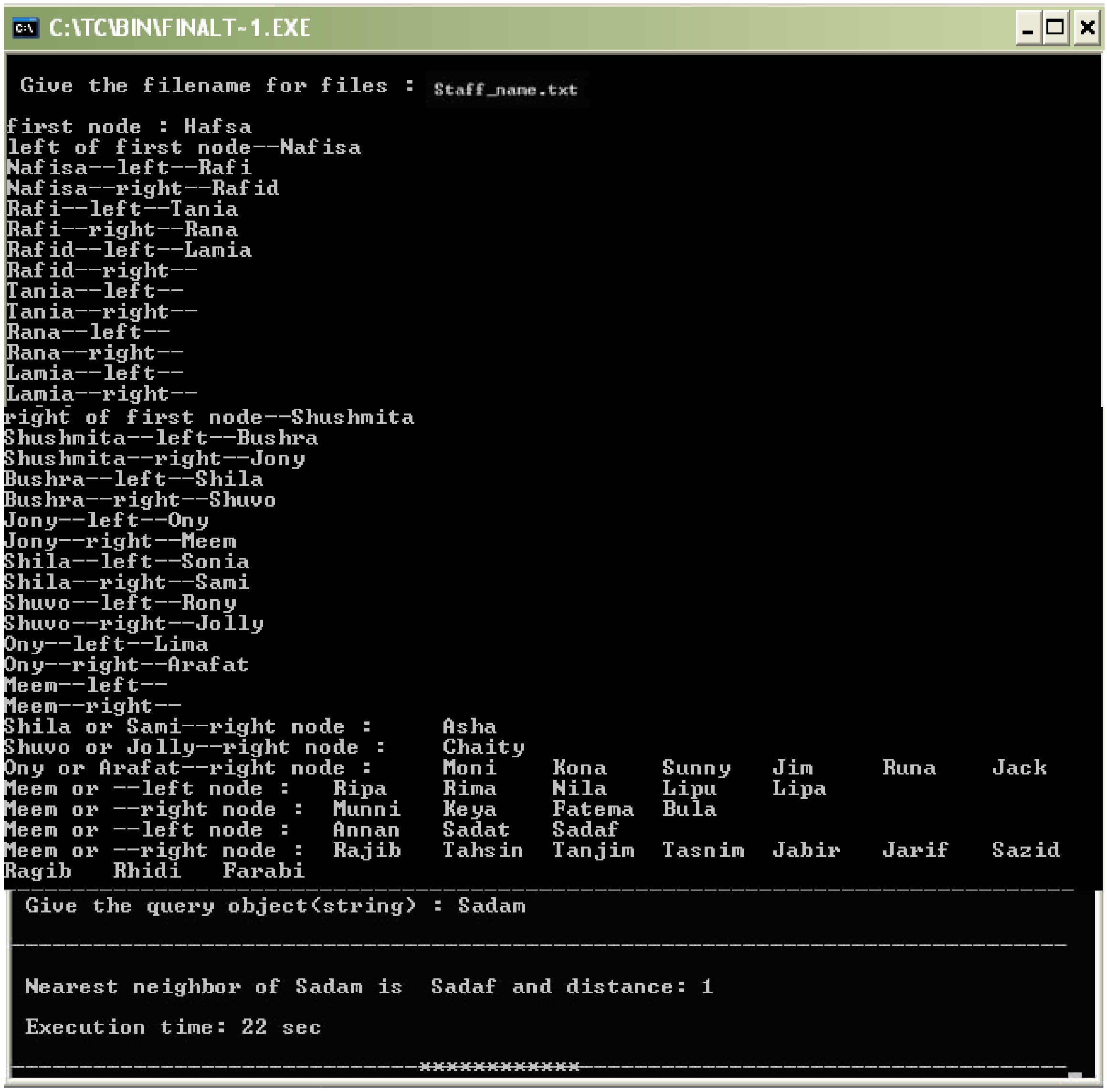
In order to evaluate the practical performances of approximate string matching algorithms (like we have done for exact string matching algorithms), we have implemented them in C in a homogeneous way on the same text file (Staff_name.txt) having 50 strings to make the comparisons significant.
Snapshots of the results are given below according to their execution time (from the fastest to the slowest):

1. Lipschitz Embeddings Algorithm:



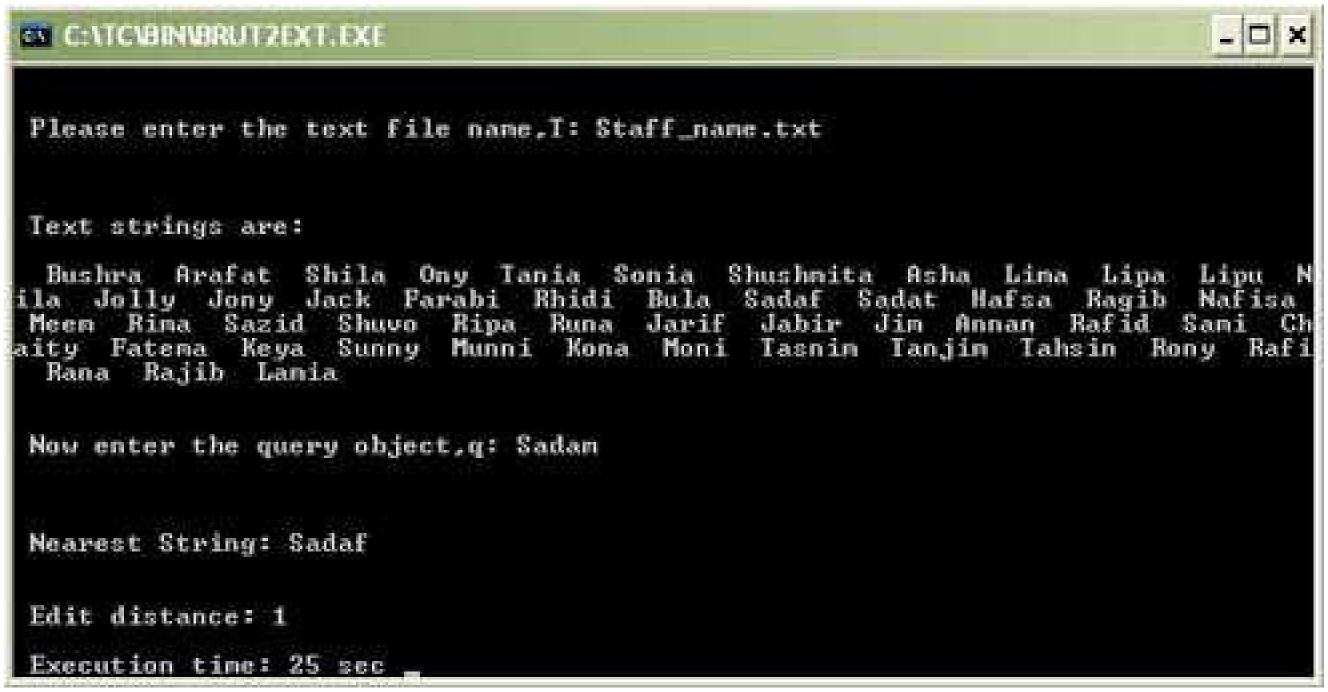
```
C:\TC\BIN\LIP2EXT.EXE

 Please enter the text file name,T: Staff_name.txt


 Text strings are:

  Bushra  Arafat  Shila  Ony  Tania  Sonia  Shushmita  Asha  Lima  Lipa  Lipu  N
ila  Jolly  Jony  Jack  Farabi  Rhidi  Bula  Sadaf  Sadat  Hafsa  Ragib  Nafisa
 Meem  Rima  Sazid  Shuvo  Ripa  Runa  Jarif  Jabir  Jim  Annan  Rafid  Sami  Ch
aity  Fatema  Keya  Sunny  Munni  Kona  Moni  Tasnim  Tanjim  Tahsin  Rony  Rafi
  Rana  Rajib  Lamia

 Randomly selected reference strings,R:

  Sami  Tanjim  Arafat  Munni  Kona  Sadat  Meem  Jim  Bushra  Jony

  Now enter the query object,q: Sadam


 Nearest String: Sadaf

 Edit distance: 1

 Execution time: 21 sec
```

2. Ball Partitioning Algorithm:



```
C:\TC\BIN\FINALT~1.EXE

 Give the filename for files : Staff_name.txt
first node : Hafsa
left of first node--Nafisa
Nafisa--left--Rafi
Nafisa--right--Rafid
Rafi--left--Tania
Rafi--right--Rana
Rafid--left--Lamia
Rafid--right--
Tania--left--
Tania--right--
Rana--left--
Rana--right--
Lamia--left--
Lamia--right--
right of first node--Shushmita
Shushmita--left--Bushra
Shushmita--right--Jony
Bushra--left--Shila
Bushra--right--Shuvo
Jony--left--Ony
Jony--right--Meem
Shila--left--Sonia
Shila--right--Sami
Shuvo--left--Rony
Shuvo--right--Jolly
Ony--left--Lima
Ony--right--Arafat
Meem--left--
Meem--right--
Shila or Sami--right node :     Asha
Shuvo or Jolly--right node :    Chaity
Ony or Arafat--right node :     Moni    Kona    Sunny   Jim     Runa    Jack
Meem or --left node :   Ripa    Rima    Nila    Lipu    Lipa
Meem or --right node :  Munni   Keya    Fatema  Bula
Meem or --left node :   Annan   Sadat   Sadaf
Meem or --right node :  Rajib   Tahsin  Tanjim  Tasnim  Jabir   Jarif   Sazid
Ragib   Rhidi   Farabi
-----------------------------------------------------------------------------
 Give the query object(string) : Sadam

-----------------------------------------------------------------------------

 Nearest neighbor of Sadam is  Sadaf and distance: 1

 Execution time: 22 sec

------------------------------*************-----------------------------------
```

3. Brute Force algorithm for approximate string matching:



```
C:\TC\BIN\BRUT2EXT.EXE

 Please enter the text file name,T: Staff_name.txt


 Text strings are:

  Bushra  Arafat  Shila  Ony  Tania  Sonia  Shushmita  Asha  Lima  Lipa  Lipu  N
ila  Jolly  Jony  Jack  Farabi  Rhidi  Bula  Sadaf  Sadat  Hafsa  Ragib  Nafisa
 Meem  Rima  Sazid  Shuvo  Ripa  Runa  Jarif  Jabir  Jim  Annan  Rafid  Sami  Ch
aity  Fatema  Keya  Sunny  Munni  Kona  Moni  Tasnim  Tanjim  Tahsin  Rony  Rafi
  Rana  Rajib  Lamia


 Now enter the query object,q: Sadam


 Nearest String: Sadaf

 Edit distance: 1

 Execution time: 25 sec _
```

6.Conclusion and Future work

6.1 Conclusion

By the grace of Allah we have come to the end of our thesis.This thesis has addressed two issues: designing a well organized database management system, which is one of the key challenges of modern era and exploring the characteristics of some existing string matching algorithms to use it in mining our database.

In the 1st part of our thesis, we have emphasized on creating an efficient database management system.We have built a database for a Blood Bank using Microsoft SQL Server. Before implementing the database, in the design phase, we have explored various features, operations of a blood bank to figure out required entities, attributes and the relationship among entities to make an efficient Entity Relationship Diagram(ERD). After analysing all the requirements, we have created our ERD and then converted the ERD to relational model and normalized the tables. Using SQL Server we have created the tables for our database and inserted some sample values in the tables. Finally, we have executed sample queries on our database to check its performance to retrieve useful information accurately and speedily.

We have presented some existing string matching algorithms and explored their characteristics in the 2nd part of our thesis. We have also provided background, pseudo codes, and clear explanations of the algorithms and comparisons among them. We have pursued research both on exact and approximate string matching algorithms and implemented them in C. We have converted our database tables into text files and applied our exact and approximate string matching codes on these text files to mine our database.

6.2 Future Work

A vast amount of future research can be possible by following the investigations and strategies described in this thesis.We haven't included all features of a Blood bank in our database.More features can be included to create the database and use it in a more versatile way.We have focused our research on four exact string matching algorithms and three approximate string matching algorithms only. There are many other advanced approximate and exact string matching algorithms having different characteristics and usefulness which can be applied for more wider comparisons.We have worked on text files having 50 strings each, which can be expanded by including more strings about 1000 strings or more. In Ball Partitioning Method, clusters or a single string can be represented at the leaves. For the sake of saving the storage space, clustering is used. In this way, we can keep up to 1000 strings or more.

7.References

[1] Database System Concepts - Silberschatz, Korth, Sudarshan (5th edition)
[2] Notes from Prof. Dorothee Koch, Hft-Stuttgart, Germany
[3] Brute Force : http://www-igm.univ-mlv.fr/~lecroq/string/node3.html
[4] Morris Pratt : http://www-igm.univ-mlv.fr/~lecroq/string/node7.html
[5] Knuth Morris Pratt: http://www-igm.univ-mlv.fr/~lecroq/string/node8.html
[6] Boyer Moore : http://www.personal.kent.edu/~rmuhamma/Algorithms/MyAlgorithms/StringMatch/boyerMoore.htm
[7] C tutorial : www.CProgrammingLanguage Tutorial.com
[8] Edit Distance : http://nlp.stanford.edu/IR-book/html/htmledition/edit-distance-1.html
[9] Approximate string match : http://en.wikipedia.org/wiki/Approximate_string_matching

[10] Lipschitz Embeddings Algorithm : Properties of Embedding Methods for Similarity Searching in Metric Space -Gı´sli R. Hjaltason, Hanan Samet, IEEE transactions on pattern analysis and machine intelligence, Vol. 25, No. 5, 2003

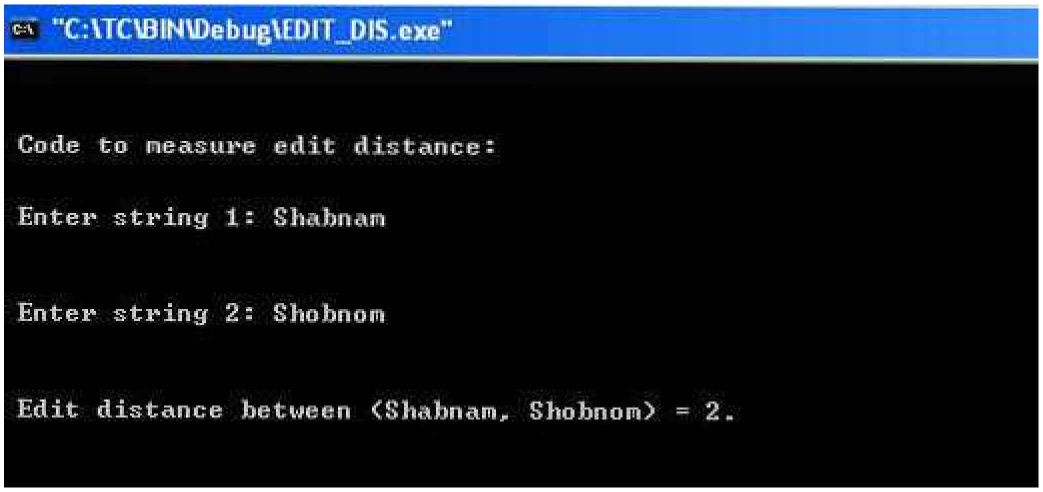
[11] Ball Partioning Algorithm: Index-Driven Similarity Search in Metric Spaces - Gı´sli R. Hjaltason, Hanan Samet, ACM Transactions on Database Systems,Vol. 28, No. 4, 2003.

8. Appendix

8.1 Code to measure Edit Distance

```
/* Code to find edit_distance */

#include <stdio.h>
#include <conio.h>
#include <string.h>
#include <stdlib.h>


#define MAXLEN 80


int findMin(int d1, int d2, int d3) {
  /*
   * return min of d1, d2 and d3.
   */
  if(d1 < d2 && d1 < d3)
     return d1;
  else if(d1 < d3)
     return d2;
  else if(d2 < d3)
     return d2;
  else
    return d3;
}

int findEditDistance(char *s1, char *s2) {
   /*
    * returns edit distance between s1 and s2.
    */
  int d1, d2, d3;

  if(*s1 == 0)
     return strlen(s2);
  if(*s2 == 0)
     return strlen(s1);
  if(*s1 == *s2)
     d1 = findEditDistance(s1+1, s2+1);
  else
     d1 = 1 + findEditDistance(s1+1, s2+1);    // update.
  d2 = 1+findEditDistance(s1, s2+1);              // insert.
  d3 = 1+findEditDistance(s1+1, s2);              // delete.
  return findMin(d1, d2, d3);
}

void main() {
  clrscr();
  char s1[MAXLEN], s2[MAXLEN];
   printf("\n \n \n Code to measure edit distance: ");
   printf("\n \n \n Enter string 1: ");
   gets(s1);


         printf("\n \n \n Enter string 2: ");
      gets(s2);
         printf("\n \n \n Edit distance between (%s, %s) = %d.\n", s1, s2,
findEditDistance(s1, s2));


getch();
}
```

Snap shot:



```
"C:\TC\BIN\Debug\EDIT_DIS.exe"


Code to measure edit distance:

Enter string 1: Shabnam


Enter string 2: Shobnom


Edit distance between (Shabnam, Shobnom) = 2.
```

8.2 Implementation of Exact String Matching Algorithms:

8.2.1 Brute force Algorithm :

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#include<string.h>
#include <time.h>

void main() {
//clrscr();
 time_t t1,t2;
 time(&t1);

char c,T[100],P[50],fname[50];

FILE *fp;
int loc=0,i,j,n,m,r=0,tm;
//clrscr();
 printf("\n \n \n Brute force Algorithm for exact string matching");
 printf("\n \n \n Please enter the text file name,T: ");
  gets(fname);

  fp=fopen(fname,"r");
  if(fp==NULL)
   {
    printf("\n Error in opening file!!");
    exit(1);
   }

printf("\n Now enter the pattern you want to search: ");
gets(P);

while(1)
{
c=fgetc(fp);
if(c==EOF)
break;

else{
  if(c!='\n'&&c!=' '){
  T[loc]=c;
  loc++;
  }
 }
}
 n=loc;
 m=strlen(P);
 for (i=0;i<=n-m;i++){
     j=0;
     while (j<m && T[i+j] == P[j])
               j++;



if(j==m){
      printf("\n\nMatch found at index : %d",i);
      r++;
      }
 }
      if(j<m && r==0)
      printf("\n\nMatch is not found");


fclose(fp);
 time(&t2);
tm= difftime(t2,t1);
 printf("\n \n Execution time: %d sec ",tm);
getch();
}
```

8.2.2 Boyer Moore Algorithm:

```
                /* BOYER_MOORE_MATCHER(T,P)

                     Input:Text(T) with n characters and Pattern(P)
                            with m characters.
                     Output:Index of the first substring of T matching P. */

#include <stdio.h>
#include <conio.h>
#include <string.h>
#include <time.h>

char T[1000],P[50];  //T:array for text,P:array for pattern
int m,n,i,j;        // m:length of pattern,n:length of text
int Min(int,int);  //function for calculating the minimum between two numbers
int last(char c); //last function which is defined in the algorithm
int boyer_moore_matcher();
```

```
void main()
{
 clrscr();
 time_t t1,t2;
 time(&t1);
 char c,fname[50];
 FILE *fp;
 int loc=0,I,tm;

 printf("\n \n /* Boyer Moore String Matching Algorithm */ \n \n ");
 printf("Please enter the text file name: ");
 gets(fname);
 fp=fopen(fname,"r");
 if(fp==NULL)
  {
   printf("\n Error in opening file!!");
  }
 printf("\n \n Now enter the pattern you want to search in the text: ");
 gets(P);

while(1)
 {
  c=fgetc(fp);
  if(c==EOF)
  break;

else
   {
    if((c!='\n')&(c!=' ')) //Ignoring spaces & newlines in the text
     {
        T[loc]=c;   //Capturing the text's data into T array
        loc++;
     }
   }
 }

n=loc;       //length of the text
m=strlen(P); // length of the pattern
I=boyer_moore_matcher();
if(I==-1)
printf("\n \n No match found ");
else
printf("\n Pattern matches from index no. %d ",I); //indexing started from zero
fclose(fp);
time(&t2);
tm=difftime(t2,t1);
```

```
printf("\n \n Execution time: %d sec ",tm);
getch();

}


int boyer_moore_matcher()
{
 int l,l1,F=-1,M;
 i=m-1;
 j=m-1;
do
 {
 if(P[j]==T[i])
  {
   if(j==0)
   { F=i;
   return F;
   }
   else
   {
    i=i-1;
    j=j-1;
    }
   }

else
 {
   l=last(T[i]);
   l1=1+l;
   M=Min(j,l1);
   i=i+m-M;
   j=m-1;
  }
}while(i<=n-1);
return F;
}

int last(char c)
{
int q,F2=-1;
for(q=m-1;q>=0;q--)
 {
  if(P[q]==c)
   {
    F2=q;
    return F2;
   }
 }
return F2;
}

int Min( int a,int b)
{
 if(a<b)
  return a;
 else
  return b;
}
```

8.2.3 Knuth Morris Pratt Algorithm:

```
/* Knuth Morris Pratt string Matching algorithm */

#include<stdio.h>
#include<conio.h>
#include<string.h>
#include<time.h>

void main() {
//clrscr();
time_t t1,t2;
time(&t1);
char u,y[1000],x[50],fname[10];

FILE *fp;
int loc=0,tm,i,j,n,m,flag=0,kmpNext[50];

printf("\n \n Knuth Morris Pratt string Matching algorithm:");
printf("\n \n Please enter the filename: ");
gets(fname);
fp=fopen(fname,"r");
while(1)
{
u=fgetc(fp);
if(u==EOF)
break;

else{
  if(u!='\n'&&u!=' '){
  y[loc]=u;
```

```
    loc++;
  }
 }
}

printf("\n \n Now enter the pattern you want to search: ");
gets(x);
 n=loc;
 m=strlen(x);
  /* Preprocessing */
      i = 0;
      j = kmpNext[0] = -1;
      while (i < m) {
      while (j > -1 && x[i] != x[j])
            j = kmpNext[j];
      i++;
      j++;
      if(x[i]==x[j])
      kmpNext[i] = kmpNext[j];
      else
      kmpNext[i]=j;

     }

   /* Searching */
   i = 0;
   j = 0;
   while (j < n) {
     while (i > -1 && x[i] != y[j])
           i = kmpNext[i];
     i++;
     j++;
     if (i >= m) {
           printf("\n\nIndex starts at : %d",j-i);
           i = kmpNext[i];
           flag++;
     }
   }

   if(flag==0)
   printf("\n\nMatch not found");

fclose(fp);

time(&t2);
tm=difftime(t2,t1);
```

```
printf("\n\n Execution time: %d sec",tm);
getch();

}
```

8.2.4 Morris Pratt Algorithm:

```
/* Morris Pratt string matching algorithm */

#include<stdio.h>
#include<conio.h>
#include<time.h>
#include<string.h>


void main() {
//clrscr();
time_t t1,t2;
time(&t1);

char x[50],y[1000];
int i=0,j=0,n=0,indx=0,tm;

char c,fname[10];


FILE *fp;
int m,r=0,mpNext[50];
printf("\n \n Morris Pratt string matching algorithm: ");
printf("\n \n Please enter the filename: ");
gets(fname);
fp=fopen(fname,"r");
while(1)
{
c=fgetc(fp);
if(c==EOF)
break;

else{
  if(c!='\n'&&c!=' '){
  y[indx]=c;
  indx++;
  }
 }
}
 n=indx;
printf("\n \n Now enter the pattern you want to search: ");
gets(x);
 m=strlen(x);
  /* Preprocessing */
      i = 0;
      j = mpNext[0] = -1;
      while (i < m) {
      while (j > -1 && x[i] != x[j])
            j = mpNext[j];

      mpNext[++i] = mpNext[++j];
     }
   /* Searching */
   i = 0;
   j = 0;
   while (j < n) {
      while (i > -1 && x[i] != y[j])
            i = mpNext[i];
      i++;
      j++;
      if (i >= m) {
            printf("\n\nIndex starts at : %d",j-i);
            i = mpNext[i];
            r++;          //flag for checking a match.
      }
   }

   if(r==0)
   printf("\n\n Match does not exist");

fclose(fp);
time(&t2);
tm=difftime(t2,t1);
printf("\n\n Execution time: %d sec",tm);
getch();

}
```

8.3 Implementation of Approximate string matching algorithms

8.3.1 Brute Force Algorithm :

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <conio.h>
#include <string.h>

#define MAX 50


int findMin(int d1, int d2, int d3) {
   /*
    * return min of d1, d2 and d3.
    */
   if(d1 < d2 && d1 < d3)
       return d1;
   else if(d1 < d3)
       return d2;
   else if(d2 < d3)
       return d2;
   else
      return d3;
}

int findEditDistance(char *s1, char *s2) {
    /*
     * returns edit distance between s1 and s2.
     */
   int d1, d2, d3;

   if(*s1 == 0)
       return strlen(s2);
   if(*s2 == 0)
       return strlen(s1);
   if(*s1 == *s2)
       d1 = findEditDistance(s1+1, s2+1);
   else
       d1 = 1 + findEditDistance(s1+1, s2+1);    // update.
   d2 = 1+findEditDistance(s1, s2+1);            // insert.
   d3 = 1+findEditDistance(s1+1, s2);            // delete.
   return findMin(d1, d2, d3);
}


void main()
{
  //clrscr();
 time_t t1,t2;
 time(&t1);

  int loc,index,tm,txtq_edit[50],edit_distance,edit_min=1000,max;

  char fname[50],q[80],nearest_string[80];
  FILE *fp;

  struct sinfo{     //Array of structures for text strings
    char sn[80];
  }txt[MAX];

  printf("\n \n Approximate String Search Algorithm Using");
  printf(" Brute Force Algorithm:");
  printf("\n \n \n Please enter the text file name,T: ");
  gets(fname);

  fp=fopen(fname,"r");
  if(fp==NULL)
   {
     printf("\n Error in opening file!!");
     exit(1);
   }

loc=0;
max=MAX-1;
printf("\n \n \n Text strings are: \n \n ");

while(!feof(fp)){
if (loc>max)
  break;
fscanf(fp,"  %s  ",txt[loc].sn);
printf(" %s ",txt[loc].sn);
loc++;
}


printf("\n \n \n Now enter the query object,q: ");
```

```
gets(q);


/*Edit distance between each of N text strings to query string,q */
for(int s=0;s<MAX;s++)
{

 txtq_edit[s]=findEditDistance(txt[s].sn,q);

}

/*Minimum edit distance */
for(int z=0;z<MAX;z++)
{
 if(txtq_edit[z]<edit_min)
  {
   edit_min=txtq_edit[z];
   index=z;
  }
}
   strcpy(nearest_string,txt[index].sn);

printf("\n \n \n Nearest String: %s ",nearest_string);
printf("\n \n \n Edit distance: %d ",edit_min);
fclose(fp);

time(&t2);
tm= difftime(t2,t1);
 printf("\n \n Execution time: %d sec ",tm);
getch();
}
```

8.3.2 Lipschitz Embeddings Algorithm:

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <conio.h>
#include <string.h>
#include <math.h>


#define MAX 50
#define K 10
#define P 2
```

```
float pf=0.5,k_p=pow(K,pf);
static int seen[MAX];
static int vector[K];


int findMin(int d1, int d2, int d3) {
  /*
   * return min of d1, d2 and d3.
   */
  if(d1 < d2 && d1 < d3)
     return d1;
  else if(d1 < d3)
     return d2;
  else if(d2 < d3)
     return d2;
  else
    return d3;
}

int findEditDistance(char *s1, char *s2) {
   /*
    * returns edit distance between s1 and s2.
    */
  int d1, d2, d3;

  if(*s1 == 0)
     return strlen(s2);
  if(*s2 == 0)
     return strlen(s1);
  if(*s1 == *s2)
     d1 = findEditDistance(s1+1, s2+1);
  else
     d1 = 1 + findEditDistance(s1+1, s2+1);    // update.
  d2 = 1+findEditDistance(s1, s2+1);           // insert.
  d3 = 1+findEditDistance(s1+1, s2);           // delete.


return findMin(d1, d2, d3);
}


void main()
{
 clrscr();
  time_t t1,t2;
```

```
// time(&t1);

  int loc,I,temp,b,original_min,index,e_time,max;
  int txt_edit[MAX][K],ref_edit[K],txtq_edit[MAX];
  float res1,res2,res3,sum,embed_edit[MAX],edit_distance,embed_min=1000;


  char fname[80],q[80],nearest_string[80];
  FILE *fp,*fp1;

  struct sinfo{     //Array of structures for text strings
    char sn[80];
  }txt[MAX];

  struct rinfo{    //Array of structures for reference strings
    char rn[80];
  }ref[K];

  printf("\n \n Nearest Neighbor Search Algorithm Using");
  printf(" Lipschitz embeddings:");
  printf("\n \n \n Please enter the text file name,T: ");
  gets(fname);

  fp=fopen(fname,"r");
  if(fp==NULL)
   {
     printf("\n Error in opening file!!");
     exit(1);
   }

  fp1=fopen("ref_file.txt","w");
  if(fp1==NULL)
   {
     printf("\n Error in opening file!!");
     exit(1);
   }

loc=0;
max=MAX-1;
printf("\n \n \n Text strings are: \n \n ");

while(!feof(fp)){
if (loc>max)
  break;
fscanf(fp,"  %s  ",txt[loc].sn);
printf(" %s ",txt[loc].sn);
loc++;
}


  srand(time(NULL));   /* Seed the random number generator. */

  for (i=0; i<K; i++) {   //create unique random numbers
    int r;
    do {
      r = rand() / (RAND_MAX / MAX + 1);
    } while (seen[r]);
    seen[r] = 1;
    if(r<max)
    vector[i] = r + 1;
  }

for(int k=0;k<K;k++)   //random reference strings selection
 {
   temp=vector[k];
   strcpy(ref[k].rn,txt[temp].sn);
 }


printf("\n \n \n Randomly selected reference strings,R: \n \n ");
for (int k2=0;k2<K;k2++)
{
  fprintf(fp1," %s ",ref[k2].rn);
  printf(" %s ",ref[k2].rn);
}

printf("\n \n \n Now enter the query object,q: ");
gets(q);

/* Edit distance between each of N(MAX) text strings to k(K)
reference strings */

for(b=0;b<MAX;b++)
{
 for(int c=0;c<K;c++)
  {

   txt_edit[b][c]=findEditDistance(txt[b].sn,ref[c].rn);
  }
}
```

```
/*Edit distance between each of k reference strings to query string(q) */
for (int a=0;a<K;a++)
{
  ref_edit[a]=findEditDistance(ref[a].rn,q);

}

/*Edit distance between each of N text strings to query string,q */
for(int s=0;s<MAX;s++)
{

 txtq_edit[s]=findEditDistance(txt[s].sn,q);

}

/* The distance between each string of T to query
   string q in embedding space
  */
for(int x=0;x<MAX;x++)
{
 res3=0;
 for (int y=0;y<K;y++)
  {
    res1=txt_edit[x][y]-ref_edit[y];
    res2=res1/k_p;
    res3+=pow(res2,P);

  }

time(&t1);
sum=pow(res3,pf);
embed_edit[x]=sum;

}
/*Minimum distance in embedding space */
for(int z=0;z<MAX;z++)
{
 if(embed_edit[z]<embed_min)
  {
    embed_min=embed_edit[z];
    index=z;
  }
}
```

```
 original_min=txtq_edit[index]; //original minimum distance

for(int w=0;w<MAX;w++)
{
 if(embed_edit[w]>original_min)
   {
     edit_distance=original_min;
     strcpy(nearest_string,txt[index].sn);
   }

else if (txtq_edit[w]<original_min)
    {
      edit_distance=txtq_edit[w];
      strcpy(nearest_string,txt[w].sn);
     }
}

printf("\n \n \n Nearest String: %s ",nearest_string);
printf("\n \n \n Edit distance: %d ",(int)edit_distance);
fclose(fp);
fclose(fp1);

 time(&t2);
e_time= difftime(t2,t1);
 printf("\n \n Execution time: %d sec ",e_time);
getch();
}
```

8.3.3 Ball Partitioning Algorithm:

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#include<string.h>
#include<time.h>
#define MAXLEN 80

int findMin(int d1, int d2, int d3) {
// return min of d1, d2 and d3.
if(d1 < d2 && d1 < d3)
return d1;

else if(d1 < d3)
```

```
return d2;

else if(d2 < d3)
return d2;

else
return d3;
}

int findEditDistance(char *s1, char *s2) {
//returns edit distance between s1 and s2.
int d1, d2, d3;

if(*s1 == 0)
return strlen(s2);

if(*s2 == 0)
return strlen(s1);

if(*s1 == *s2)
d1 = findEditDistance(s1+1, s2+1);

else
d1 = 1 + findEditDistance(s1+1, s2+1);    // update.
d2 = 1+findEditDistance(s1, s2+1);        // insert.
d3 = 1+findEditDistance(s1+1, s2);        // delete.

return findMin(d1, d2, d3);

}

//sorting using quick sort.

void sort(int array[MAXLEN],int left, int right){
  int i=0,j=0,flag=0,part=0,temp=0;


  if(right>left) {
    i=left;
    j=right;
    part=array[left];
    flag=0;

    while(!flag){
      do{
           ++i;
```

```
            }while((array[i]<=part)&&(i<=right));
        while((array[j]>=part)&&(j>left)){
            --j;
        }
        if(j<i){
            flag=1;
        }
        else{
             temp=array[i];
             array[i]=array[j];
             array[j]=temp;
        }
      }

      temp=array[left];
      array[left]=array[j];
      array[j]=temp;

      //call recursively
      sort(array,left,j-1);
      sort(array,i,right);
    }
      return;
}



int findRandom(){

int i;
char s0='0';
char s1='1';
char s2='2';
char s3='3';
char s4='4';
char s5='5';
char s6='6';
char s7='7';
char s8='8';
char s9='9';

char s10='a';
char s11='b';
char s12='c';
char s13='d';
char s14='e';
char s15='f';
char s16='g';
char s17='h';
char s18='i';
char s19='j';



FILE *fr;
char ch;
fr=fopen("AB.TXT","r");
while(1){
ch=fgetc(fr);
if(ch==EOF){
fclose(fr);
break;
}
else{
if(ch=='0'){
fclose(fr);
fr=fopen("AB.TXT","w");
  i=1;
  fputc(s1,fr);
  fclose(fr);
}//if
if(ch=='1'){
fclose(fr);
fr=fopen("AB.TXT","w");
  i=2;
  fputc(s2,fr);
  fclose(fr);
}//if
if(ch=='2'){
fclose(fr);
fr=fopen("AB.TXT","w");
  i=3;
  fputc(s3,fr);
  fclose(fr);
}//if

if(ch=='3'){
fclose(fr);
fr=fopen("AB.TXT","w");
  i=4;
  fputc(s4,fr);
  fclose(fr);
```

```
}//if
if(ch=='4'){
fclose(fr);
fr=fopen("AB.TXT","w");
  i=5;
  fputc(s5,fr);
  fclose(fr);
}//if
if(ch=='5'){
fclose(fr);
fr=fopen("AB.TXT","w");
  i=6;
  fputc(s6,fr);
  fclose(fr);
}//if
if(ch=='6'){
fclose(fr);
fr=fopen("AB.TXT","w");
  i=7;
  fputc(s7,fr);
  fclose(fr);
}//if
if(ch=='7'){
fclose(fr);
fr=fopen("AB.TXT","w");
  i=8;
  fputc(s8,fr);
  fclose(fr);
}//if
if(ch=='8'){
fclose(fr);
fr=fopen("AB.TXT","w");
  i=9;
  fputc(s9,fr);
  fclose(fr);
}//if
if(ch=='9'){
fclose(fr);
fr=fopen("AB.TXT","w");
  i=10;
  fputc(s10,fr);
  fclose(fr);
}//if


if(ch=='a'){
```

```
fclose(fr);
fr=fopen("AB.TXT","w");
  i=11;
  fputc(s11,fr);
  fclose(fr);
}//if
if(ch=='b'){
fclose(fr);
fr=fopen("AB.TXT","w");
  i=12;
  fputc(s12,fr);
  fclose(fr);
}//if
if(ch=='c'){
fclose(fr);
fr=fopen("AB.TXT","w");
  i=13;
  fputc(s13,fr);
  fclose(fr);
}//if

if(ch=='d'){
fclose(fr);
fr=fopen("AB.TXT","w");
  i=14;
  fputc(s14,fr);
  fclose(fr);
}//if
if(ch=='e'){
fclose(fr);
fr=fopen("AB.TXT","w");
  i=15;
  fputc(s15,fr);
  fclose(fr);
}//if
if(ch=='f'){
fclose(fr);
fr=fopen("AB.TXT","w");
  i=16;
  fputc(s16,fr);
  fclose(fr);
}//if
if(ch=='g'){
fclose(fr);
fr=fopen("AB.TXT","w");
  i=17;
```

```
   fputc(s17,fr);
   fclose(fr);
}//if
if(ch=='h'){
fclose(fr);
fr=fopen("AB.TXT","w");
   i=18;
   fputc(s18,fr);
   fclose(fr);
}//if
if(ch=='i'){
fclose(fr);
fr=fopen("AB.TXT","w");
   i=19;
   fputc(s19,fr);
   fclose(fr);
}//if
if(ch=='j'){
fclose(fr);
fr=fopen("AB.TXT","w");
   i=20;
   fputc(s0,fr);
   fclose(fr);
}//if

} //else
} //while

return i;

}




//main code
struct readf0{  //only pivot and it's radius will be stored here
    char mm[20];
    float dd;
}r[80];

struct readf{   //strings and will be stored here from others
   char mm[20];
   int dd;
}r1[55];
struct readf1{
    char mm[20];//for sorting
    int dd;
}r2[80];
struct readf2{
    char mm[20];//keep the strings from file
}r3[55];



struct l0{
    char mm[20];
}l0[50];

struct r0{
char mm[20];
}r0[50];


struct a0{
    char mm[20];
}a0[50];

struct a1{
char mm[20];
}a1[50];

struct a2{
    char mm[20];
}a2[45];

struct a3{
char mm[20];
}a3[50];

struct a4{
char mm[20];
}a4[45];

struct a5{
    char mm[20];
}a5[50];

struct a6{
char mm[20];
}a6[40];
```

```
struct a7{
   char mm[20];
}a7[45];

struct a8{
char mm[20];
}a8[50];

struct a9{
char mm[20];
}a9[50];

struct a10{
   char mm[20];
}a10[50];

struct a11{
char mm[20];
}a11[50];

struct a12{
   char mm[20];
}a12[45];

struct a13{
char mm[20];
}a13[50];

struct a14{
char mm[20];
}a14[45];

struct a15{
   char mm[20];
}a15[50];

struct a16{
char mm[20];
}a16[40];

struct a17{
   char mm[20];
}a17[45];

struct a18{
char mm[20];
}a18[50];

struct a19{
char mm[20];
}a19[50];
//
struct a20{
   char mm[20];
}a20[50];

struct a21{
char mm[20];
}a21[50];

struct a22{
   char mm[20];
}a22[45];

struct a23{
char mm[20];
}a23[50];

struct a24{
char mm[20];
}a24[45];

struct a25{
   char mm[20];
}a25[50];

struct a26{
char mm[20];
}a26[40];

struct a27{
   char mm[20];
}a27[45];

struct a28{
char mm[20];
}a28[50];

struct a29{
char mm[20];
}a29[50];
```

```
int array[MAXLEN],val=0,n=0,p=0,q=0,count=0,W=0,lim=0,cnt=0,A=0,B=0,C=0;

void main(){

time_t t1,t2;

int tm;

int i=0,j=0,k=0,l,flag=0,fl=0,left=0,right=0,mid=0,aj=0;
int midev1=0,midev2=0,midod1=0;

int x0=0,x1=0,x2=0,x3=0,x4=0,x5=0,x6=0,x7=0,x8=0,x9=0,x10=0;
int x11=0,x12=0,x13=0,x14=0,x15=0,x16=0,x17=0,x18=0,x19=0,x20=0;
int x21=0,x22=0,x23=0,x24=0,x25=0,x26=0,x27=0,x28=0,x29=0,x30=0;
int x31=0;
int st=0,in=0;

float midev,midod;
char c,newa[20],fname[10],query[20],dname[20];
char gh[20];

FILE *fm;

clrscr();

printf("\n Give the filename for files : ");
gets(fname);


for(k=0;k<16;k++){
   newa[k]='\0';
}

 for(k=0;k<MAXLEN;k++)
     array[k]=0;

fm=fopen(fname,"r");

while(1)
 {
   c=fgetc(fm);       //reading strings from a text file

   if(c==EOF){
       strcpy(r3[j].mm,newa);
       j++;
     i=0;
     break;
  }

    if(c!=' '&&c!='\n'){
          flag=0;
          fl=0;
          newa[i]=c;
          i++;
    }

    if(c==' '&&fl!=1){
            fl=1;
            strcpy(r3[j].mm,newa);
            j++;
            i=0;
            for(k=0;k<20;k++)
            newa[k]='\0';
    }

     if(c=='\n'&&flag!=1){
            flag=1;
            strcpy(r3[j].mm,newa);
            j++;
            i=0;
            for(k=0;k<20;k++)
            newa[k]='\0';
    }
 }

  fclose(fm);




aj=0;
for(q=0;q<17;q++){
aj=0;

if(q==0){
for(i=0;i<j;i++)
strcpy(r1[i].mm,r3[i].mm);
aj=j-1;
}

if(q==1){
```

```
for(i=0;i<x0;i++)
strcpy(r1[i].mm,l0[i].mm);
aj=x0-1;
}

if(q==2){
for(i=0;i<x2;i++)
strcpy(r1[i].mm,a0[i].mm);
aj=x2-1;
}

if(q==3){
for(i=0;i<x3;i++)
strcpy(r1[i].mm,a1[i].mm);
aj=x3-1;
}

if(q==4){
for(i=0;i<x4;i++)
strcpy(r1[i].mm,a2[i].mm);
aj=x4-1;
}

if(q==5){
for(i=0;i<x5;i++)
strcpy(r1[i].mm,a3[i].mm);
aj=x5-1;
}

if(q==6){
for(i=0;i<x6;i++)
strcpy(r1[i].mm,a4[i].mm);
aj=x6-1;
}

if(q==7){
for(i=0;i<x7;i++)
strcpy(r1[i].mm,a5[i].mm);
aj=x7-1;
}

if(q==8){
for(i=0;i<x8;i++)
strcpy(r1[i].mm,a6[i].mm);
aj=x8-1;
}
```

```
if(q==9){
for(i=0;i<x9;i++)
strcpy(r1[i].mm,a7[i].mm);
aj=x9-1;
}

if(q==10){
for(i=0;i<x10;i++)
strcpy(r1[i].mm,a8[i].mm);
aj=x10-1;
}

if(q==11){
for(i=0;i<x11;i++)
strcpy(r1[i].mm,a9[i].mm);
aj=x11-1;
}

if(q==12){
for(i=0;i<x12;i++)
strcpy(r1[i].mm,a10[i].mm);
aj=x12-1;
}

if(q==13){
for(i=0;i<x13;i++)
strcpy(r1[i].mm,a11[i].mm);
aj=x13-1;
}

if(q==14){
for(i=0;i<x14;i++)
strcpy(r1[i].mm,a12[i].mm);
aj=x14-1;
}

if(q==15){
for(i=0;i<x15;i++)
strcpy(r1[i].mm,a13[i].mm);
aj=x15-1;
}



if(q==16&&cnt==0){
```

```
for(i=0;i<x1;i++)
strcpy(l0[i].mm,r0[i].mm);
cnt=1;
x0=x1;
aj=-1;
st=1;
q=0;
x2=0,x3=0,x4=0,x5=0,x6=0,x7=0,x8=0,x9=0,x10=0;
x11=0,x12=0,x13=0,x14=0,x15=0,x16=0,x17=0,x18=0,x19=0,x20=0;
x21=0,x22=0,x23=0,x24=0,x25=0,x26=0,x27=0,x28=0,x29=0,x30=0;
x31=0;
}

if(aj>=1&&q<16){
//more than one

 val=findRandom();
  if(val>=aj)
  val=0;


  for(i=0;i<=aj;i++){             //edit distance part
    if(i!=val)
            r1[i].dd=findEditDistance(r1[i].mm,r1[val].mm);

    if(i==val)
             r1[i].dd=100;
  }

     for(i=0;i<=aj;i++)        //distances keep in array[] for sorting
           array[i]=r1[i].dd;


     left=0;
     right=aj;
     sort(array,left,right);

      mid=aj; //mid is index without value=100

      if(mid==0){ //means pivot element
       strcpy(r[p].mm,r1[0].mm);
       r[p].dd=0;
      }

      if(mid==1){  //means 1 element+pivot
              strcpy(r[p].mm,r1[val].mm);
```

```
        r[p].dd=array[0];
}

 if(mid==2){  //means 2 elements
        strcpy(r[p].mm,r1[val].mm);
        r[p].dd=(array[0]+array[1])/2;
}


//odd-even                         //formula of median (even value)
if(mid>2&&mid%2==0){  //even
        midev1=mid/2;
        midev2=(mid/2+1);
        midev=(array[midev1]+array[midev2])/2;
        strcpy(r[p].mm,r1[val].mm);
        r[p].dd=midev;
}


if(mid>2&&mid%2!=0){  //odd        //formula of median (odd value)
         midod1=(mid+1)/2;
         midod=array[midod1];
         strcpy(r[p].mm,r1[val].mm);
         r[p].dd=midod;
      }

          if(mid>=1){
          for(i=0;i<=aj;i++){
           if(strcmp(r1[i].mm,r[p].mm)!=0){              //left child: written in an array
            if(r1[i].dd>=0 && r1[i].dd<r[p].dd){

                  if(q==0){
                  strcpy(l0[x0].mm,r1[i].mm);
                  x0++;
                  }

                  if(q==1){
                  strcpy(a0[x2].mm,r1[i].mm);
                  x2++;
                  }

                  if(q==2){
                  strcpy(a2[x4].mm,r1[i].mm);
                  x4++;
                  }
```

```
if(q==3){
strcpy(a4[x6].mm,r1[i].mm);
x6++;
}

if(q==4){
strcpy(a6[x8].mm,r1[i].mm);
x8++;
}

if(q==5){
strcpy(a8[x10].mm,r1[i].mm);
x10++;
}

if(q==6){
strcpy(a10[x12].mm,r1[i].mm);
x12++;
}

if(q==7){
strcpy(a12[x14].mm,r1[i].mm);
x14++;
}



if(q==8){
strcpy(a14[x16].mm,r1[i].mm);
x16++;
}

if(q==9){
strcpy(a16[x18].mm,r1[i].mm);
x18++;
}

if(q==10){
strcpy(a18[x20].mm,r1[i].mm);
x20++;
}
if(q==11){
strcpy(a20[x22].mm,r1[i].mm);
x22++;
}
```

```
        if(q==12){
        strcpy(a22[x24].mm,r1[i].mm);
        x24++;
        }

        if(q==13){
        strcpy(a24[x26].mm,r1[i].mm);
        x26++;
        }

        if(q==14){
        strcpy(a26[x28].mm,r1[i].mm);
        x28++;
        }

        if(q==15){
        strcpy(a28[x30].mm,r1[i].mm);
        x30++;
        }


        }

    if(r1[i].dd>=r[p].dd){              //right child is storing in another array

        if(q==0){
        strcpy(r0[x1].mm,r1[i].mm);
        x1++;
        }

        if(q==1){
        strcpy(a1[x3].mm,r1[i].mm);
        x3++;
        }

        if(q==2){
        strcpy(a3[x5].mm,r1[i].mm);
        x5++;
        }

        if(q==3){
        strcpy(a5[x7].mm,r1[i].mm);
        x7++;
        }

        if(q==4){
```

```
strcpy(a7[x9].mm,r1[i].mm);
x9++;
}

if(q==5){
strcpy(a9[x11].mm,r1[i].mm);
x11++;
}

if(q==6){
strcpy(a11[x13].mm,r1[i].mm);
x13++;
}

if(q==7){
strcpy(a13[x15].mm,r1[i].mm);
x15++;
}

if(q==8){
strcpy(a15[x17].mm,r1[i].mm);
x17++;
}

if(q==9){
strcpy(a17[x19].mm,r1[i].mm);
x19++;
}

if(q==10){
strcpy(a19[x21].mm,r1[i].mm);
x21++;
}

if(q==11){
strcpy(a21[x23].mm,r1[i].mm);
x23++;
}

if(q==12){
strcpy(a23[x25].mm,r1[i].mm);
x25++;
}

if(q==13){
strcpy(a25[x27].mm,r1[i].mm);
```

```
                    x27++;
                    }

                    if(q==14){
                    strcpy(a27[x29].mm,r1[i].mm);
                    x29++;
                    }

                    if(q==15){
                    strcpy(a29[x31].mm,r1[i].mm);
                    x31++;
                    }



                    }
                }    //if

              }   //for

            }

    p++;

   }//aj>1

if(aj==0&&q<16){
strcpy(r[p].mm,r1[0].mm);
r[p].dd=0;
 p++;
}

//printf


if(st==0&&q==15){


printf("\nfirst node : %s",r[0].mm);//0

printf("\nleft of first node--%s",r[1].mm);//1



in++;
printf("\n%s--left--%s",r[1].mm,r[2].mm);//2
```

```
in++;
printf("\n%s--right--%s",r[1].mm,r[3].mm);//3

in++;
printf("\n%s--left--%s",r[2].mm,r[4].mm);//4
in++;
printf("\n%s--right--%s",r[2].mm,r[5].mm);//5

in++;
printf("\n%s--left--%s",r[3].mm,r[6].mm);//6
in++;
printf("\n%s--right--%s",r[3].mm,r[7].mm);//7

in++;
printf("\n%s--left--%s",r[4].mm,r[8].mm);//8
in++;
printf("\n%s--right--%s",r[4].mm,r[9].mm);//9

in++;
printf("\n%s--left--%s",r[5].mm,r[10].mm);//10
in++;
printf("\n%s--right--%s",r[5].mm,r[11].mm);//11

in++;
printf("\n%s--left--%s",r[6].mm,r[12].mm);//12
in++;
printf("\n%s--right--%s",r[6].mm,r[13].mm);//13

in++;
printf("\n%s--left--%s",r[7].mm,r[14].mm);//14
in++;
printf("\n%s--right--%s",r[7].mm,r[15].mm);//15


x16=x16-1;
if(x16>=0){
printf("\n%s or %s--left node : ",r[4].mm,r[8].mm);//8
while(x16!=-1){
in++;
strcpy(r[in].mm,a14[x16].mm);
printf("\t%s",r[in].mm);
x16--;
}
}

x17=x17-1;
```

```
if(x17>=0){
printf("\n%s or %s--right node : ",r[4].mm,r[8].mm);//8
while(x17!=-1){
in++;
strcpy(r[in].mm,a15[x17].mm);
printf("\t%s",r[in].mm);
x17--;
}
}


x18=x18-1;
if(x18>=0){
printf("\n%s or %s--left node : ",r[4].mm,r[9].mm);//9
while(x16!=-1){
in++;
strcpy(r[in].mm,a16[x18].mm);
printf("\t%s",r[in].mm);
x18--;
}
}

x19=x19-1;
if(x19>=0){
printf("\n%s or %s--right node : ",r[4].mm,r[9].mm);//9
while(x19!=-1){
in++;
strcpy(r[in].mm,a17[x19].mm);
printf("\t%s",r[in].mm);
x19--;
}
}

x20=x20-1;
if(x20>=0){
printf("\n%s or %s--left node : ",r[5].mm,r[10].mm);//10
while(x20!=-1){
in++;
strcpy(r[in].mm,a18[x20].mm);
printf("\t%s",r[in].mm);
x20--;
}
}

x21=x21-1;
if(x21>=0){
```

```
printf("\n%s or %s--right node : ",r[5].mm,r[10].mm);//10
while(x21!=-1){
in++;
strcpy(r[in].mm,a19[x21].mm);
printf("\t%s",r[in].mm);
x21--;
}
}


x22=x22-1;
if(x22>=0){
printf("\n%s or %s--left node : ",r[5].mm,r[11].mm);//11
while(x22!=-1){
in++;
strcpy(r[in].mm,a20[x22].mm);
printf("\t%s",r[in].mm);
x22--;
}
}

x23=x23-1;
if(x23>=0){
printf("\n%s or %s--right node : ",r[5].mm,r[11].mm);//11
while(x23!=-1){
in++;
strcpy(r[in].mm,a21[x23].mm);
printf("\t%s",r[in].mm);
x23--;
}
}


x24=x24-1;
if(x24>=0){
printf("\n%s or %s--left node : ",r[6].mm,r[12].mm);//12
while(x24!=-1){
in++;
strcpy(r[in].mm,a22[x24].mm);
printf("\t%s",r[in].mm);
x24--;
}
}

x25=x25-1;
if(x25>=0){
```

```
printf("\n%s or %s--right node : ",r[6].mm,r[12].mm);//12
while(x25!=-1){
in++;
strcpy(r[in].mm,a23[x25].mm);
printf("\t%s",r[in].mm);
x25--;
}
}


x26=x26-1;
if(x26>=0){
printf("\n%s or %s--left node : ",r[6].mm,r[13].mm);//13
while(x26!=-1){
in++;
strcpy(r[in].mm,a24[x26].mm);
printf("\t%s",r[in].mm);
x26--;
}
}

x27=x27-1;
if(x27>=0){
printf("\n%s or %s--right node : ",r[6].mm,r[13].mm);//13
while(x27!=-1){
in++;
strcpy(r[in].mm,a25[x27].mm);
printf("\t%s",r[in].mm);
x27--;
}
}

x28=x28-1;
if(x28>=0){
printf("\n%s or %s--left node : ",r[7].mm,r[14].mm);//14
while(x28!=-1){
in++;
strcpy(r[in].mm,a26[x28].mm);
printf("\t%s",r[in].mm);
x28--;
}
}

x29=x29-1;
if(x29>=0){
printf("\n%s or %s--right node : ",r[7].mm,r[14].mm);//14
```

```
while(x29!=-1){
in++;
strcpy(r[in].mm,a27[x29].mm);
printf("\t%s",r[in].mm);
x29--;
}
}


x30=x30-1;
if(x30>=0){
printf("\n%s or %s--left node : ",r[7].mm,r[15].mm);//15
while(x30!=-1){
in++;
strcpy(r[in].mm,a28[x30].mm);
printf("\t%s",r[in].mm);
x30--;
}
}

x31=x31-1;
if(x31>=0){
printf("\n%s or %s--right node : ",r[7].mm,r[15].mm);//15
while(x31!=-1){
in++;
strcpy(r[in].mm,a29[x31].mm);
printf("\t%s",r[in].mm);
x31--;
}
}


p=31;


}


// next one for the right one
if(st==1&&q==15){
in=31;
q=17;
printf("\nright of first node--%s",r[31].mm);//1

in++;
printf("\n%s--left--%s",r[31].mm,r[32].mm);//2
```

```
in++;
printf("\n%s--right--%s",r[31].mm,r[33].mm);//3

in++;
printf("\n%s--left--%s",r[32].mm,r[34].mm);//4
in++;
printf("\n%s--right--%s",r[32].mm,r[35].mm);//5

in++;
printf("\n%s--left--%s",r[33].mm,r[36].mm);//6
in++;
printf("\n%s--right--%s",r[33].mm,r[37].mm);//7

in++;
printf("\n%s--left--%s",r[34].mm,r[38].mm);//8
in++;
printf("\n%s--right--%s",r[34].mm,r[39].mm);//9

in++;
printf("\n%s--left--%s",r[35].mm,r[40].mm);//10
in++;
printf("\n%s--right--%s",r[35].mm,r[41].mm);//11

in++;
printf("\n%s--left--%s",r[36].mm,r[42].mm);//12
in++;
printf("\n%s--right--%s",r[36].mm,r[43].mm);//13

in++;
printf("\n%s--left--%s",r[37].mm,r[44].mm);//14
in++;
printf("\n%s--right--%s",r[37].mm,r[45].mm);//15


x16=x16-1;
if(x16>=0){
printf("\n%s or %s--left node : ",r[34].mm,r[38].mm);//8
while(x16!=-1){
in++;
strcpy(r[in].mm,a14[x16].mm);
printf("\t%s",r[in].mm);
x16--;
}
}

x17=x17-1;
```

```
if(x17>=0){
printf("\n%s or %s--right node : ",r[34].mm,r[38].mm);//8
while(x17!=-1){
in++;
strcpy(r[in].mm,a15[x17].mm);
printf("\t%s",r[in].mm);
x17--;
}
}


x18=x18-1;
if(x18>=0){
printf("\n%s or %s--left node : ",r[34].mm,r[39].mm);//9
while(x16!=-1){
in++;
strcpy(r[in].mm,a16[x18].mm);
printf("\t%s",r[in].mm);
x18--;
}
}

x19=x19-1;
if(x19>=0){
printf("\n%s or %s--right node : ",r[34].mm,r[39].mm);//9
while(x19!=-1){
in++;
strcpy(r[in].mm,a17[x19].mm);
printf("\t%s",r[in].mm);
x19--;
}
}

x20=x20-1;
if(x20>=0){
printf("\n%s or %s--left node : ",r[35].mm,r[40].mm);//10
while(x20!=-1){
in++;
strcpy(r[in].mm,a18[x20].mm);
printf("\t%s",r[in].mm);
x20--;
}
}

x21=x21-1;
if(x21>=0){
```

```
printf("\n%s or %s--right node : ",r[35].mm,r[40].mm);//10
while(x21!=-1){
in++;
strcpy(r[in].mm,a19[x21].mm);
printf("\t%s",r[in].mm);
x21--;
}
}


x22=x22-1;
if(x22>=0){
printf("\n%s or %s--left node : ",r[35].mm,r[41].mm);//11
while(x22!=-1){
in++;
strcpy(r[in].mm,a20[x22].mm);
printf("\t%s",r[in].mm);
x22--;
}
}

x23=x23-1;
if(x23>=0){
printf("\n%s or %s--right node : ",r[35].mm,r[41].mm);//11
while(x23!=-1){
in++;
strcpy(r[in].mm,a21[x23].mm);
printf("\t%s",r[in].mm);
x23--;
}
}


x24=x24-1;
if(x24>=0){
printf("\n%s or %s--left node : ",r[36].mm,r[42].mm);//12
while(x24!=-1){
in++;
strcpy(r[in].mm,a22[x24].mm);
printf("\t%s",r[in].mm);
x24--;
}
}

x25=x25-1;
if(x25>=0){
```

```
printf("\n%s or %s--right node : ",r[36].mm,r[42].mm);//12
while(x25!=-1){
in++;
strcpy(r[in].mm,a23[x25].mm);
printf("\t%s",r[in].mm);
x25--;
}
}

x26=x26-1;
if(x26>=0){
printf("\n%s or %s--left node : ",r[36].mm,r[43].mm);//13
while(x26!=-1){
in++;
strcpy(r[in].mm,a24[x26].mm);
printf("\t%s",r[in].mm);
x26--;
}
}

x27=x27-1;
if(x27>=0){
printf("\n%s or %s--right node : ",r[36].mm,r[43].mm);//13
while(x27!=-1){
in++;
strcpy(r[in].mm,a25[x27].mm);
printf("\t%s",r[in].mm);
x27--;
}
}

x28=x28-1;
if(x28>=0){
printf("\n%s or %s--left node : ",r[37].mm,r[44].mm);//14
while(x28!=-1){
in++;
strcpy(r[in].mm,a26[x28].mm);
printf("\t%s",r[in].mm);
x28--;
}
}

x29=x29-1;
if(x29>=0){
printf("\n%s or %s--right node : ",r[37].mm,r[44].mm);//14
while(x29!=-1){
```

```
in++;
strcpy(r[in].mm,a27[x29].mm);
printf("\t%s",r[in].mm);
x29--;
}
}


x30=x30-1;
if(x30>=0){
printf("\n%s or %s--left node : ",r[37].mm,r[45].mm);//15
while(x30!=-1){
in++;
strcpy(r[in].mm,a28[x30].mm);
printf("\t%s",r[in].mm);
x30--;
}
}

x31=x31-1;
if(x31>=0){
printf("\n%s or %s--right node : ",r[37].mm,r[45].mm);//15
while(x31!=-1){
in++;
strcpy(r[in].mm,a29[x31].mm);
printf("\t%s",r[in].mm);
x31--;
}
}

}


}  //for loop



//printf("\nTotal object : %d",p);
time(&t1);
printf("\n---------------------------------------------------------------------------");
printf("\n Give the query object(string) : ");
gets(query);


p=80;
count=0;
```

```
r2[0].dd=findEditDistance(query,r[0].mm);
strcpy(r2[0].mm,r[0].mm);

count++;
for(i=1;i<p;i++){
lim=0;
B=r2[i-1].dd-r[i-1].dd;
W=r[i-1].dd-r2[i-1].dd;


if(lim==0&&B>=0&&B<=r2[i-1].dd){
lim=1;
r2[i].dd=findEditDistance(query,r[i].mm);
strcpy(r2[i].mm,r[i].mm);

}

if(lim==0&&B<=0&&0<=r2[i-1].dd){
lim=1;
r2[i].dd=findEditDistance(query,r[i].mm);
strcpy(r2[i].mm,r[i].mm);

}

//right
if(lim==0&&W>=0&&W<=r2[i-1].dd){
lim=1;
r2[i].dd=findEditDistance(query,r[i].mm);
strcpy(r2[i].mm,r[i].mm);
}

if(lim==0&&W<=0&&0<=r2[i-1].dd){
lim=1;
r2[i].dd=findEditDistance(query,r[i].mm);
strcpy(r2[i].mm,r[i].mm);

}
count++;
}


//nearest neighbor code

for(i=0;i<MAXLEN;i++)
   array[i]=100;
```

```
 for(i=0;i<count;i++)
   array[i]=r2[i].dd;

   left=0;
   right=count-1;
   sort(array,left,right);
   A=0;

   for(i=0;i<count;i++){

   if(r2[i].dd==array[0]&&A==0){
   printf("\n Nearest neighbor of %s is  %s and distance: %d",query,r2[i].mm,array[0]);
   printf("\n------------------------------************------------------------------------");
   A=1;
   break;
   }
   }



time(&t2);
tm=difftime(t2,t1);
printf("\n Execution time: %d sec",tm);

getch();

}
```

THE END