Os 20 marks question

Slip 1

Q.2 Write the simulation program to implement demand paging and show the page scheduling and total number of page faults for the following given page reference string. Give input n=3 as the number of memory frames.

Reference String :3, 4, 5, 6, 3, 4, 7, 3, 4, 5, 6, 7, 2, 4, 6

```
#include <stdio.h>
int main()
{
  int rstring[] = {3, 4, 5, 6, 3, 4, 7, 3, 4, 5, 6, 7, 2, 4, 6 };
  int pf = 0, frames, m, n, s, pages = 15;
  printf("\nFIFO Page Replacement Algorithm\n\n");
  printf("Enter no of frames: ");
  scanf("%d", &frames);
  int temp[frames];
  for (m = 0; m < frames; m++)
  {
    temp[m] = -1;
  }
```

```
for (m = 0; m < pages; m++)
{
  s = 0;
  for (n = 0; n < frames; n++)
  {
    if (rstring[m] == temp[n])
    {
      s++;
      pf--;
    }
  }
  pf++;
  if ((pf <= frames) && (s == 0))
  {
    temp[m] = rstring[m];
  }
  else if (s == 0)
  {
    temp[(pf - 1) % frames] = rstring[m];
  }

  printf("\n");

  // printf("%d\t", rstring[m]);

  for (n = 0; n < frames; n++)

  {

    if (temp[n] != -1)

      printf("%d\t", temp[n]);

    else

      printf("-\t");

  }

}

printf("\n\n Total Page Faults = %d\n\n", pf);
```

```
}
```

OR

Write a C program to simulate Banker's algorithm for the purpose of deadlock avoidance.

Consider the following snapshot of system, A, B, C and D are the resource type.

a) Calculate and display the content of need matrix?

b) Is the system in safe state? If display the safe sequence.

c) If a request from process P arrives for (0, 4, 2, 0) can it be granted immediately by keeping the system in safe state. Print a message

```
#include<stdio.h>

#define MAX 10

int m,n,total[MAX],avail[MAX],alloc[MAX][MAX],

    max[MAX][MAX],need[MAX][MAX],work[MAX],finish[MAX],

    seq[MAX],request[MAX];


void accept()

{

 int i,j;

 printf("Enter no.of process:");

 scanf("%d",&n);

 printf("Enter no.of resource types:");
```

```
scanf("%d",&m);

printf("Enter total no.of resources of each resource type:\n");

for(i=0;i<m;i++)

{

 printf("%c:",65+i);

 scanf("%d",&total[i]);

}

printf("Enter no.of allocated resources of each resource type by each process:\n");

for(i=0;i<n;i++)

{

 printf("P%d:\n",i);

 for(j=0;j<m;j++)

 {

  printf("%c:",65+j);

  scanf("%d",&alloc[i][j]);

 }

}

printf("Enter no.of maximum resources of each resource type by each process:\n");

for(i=0;i<n;i++)

{

 printf("P%d:\n",i);

 for(j=0;j<m;j++)
```

```
  {
   printf("%c:",65+j);
   scanf("%d",&max[i][j]);
  }
 }
}
void calc_avail_accept()
{
 int b,j;
 printf("Enter the avail:");
 for(b=0;b<j;b++)
    scanf("%d",&avail[b]);
 for(b=0;b<j;b++)
    printf("%d\t",avail[b]);
}

void calc_need()
{
 int i,j;
 for(i=0;i<n;i++)
  for(j=0;j<m;j++)
   need[i][j]=max[i][j]-alloc[i][j];
```

```
}

void print()

{

 int i,j;

 printf("\tAllocation\tMax\tNeed\n\t");

 for(i=0;i<3;i++)

 {

  for(j=0;j<m;j++)

   printf("%3c",65+j);

  printf("\t");

 }

 printf("\n");

 for(i=0;i<n;i++)

 {

  printf("P%d\t",i);

  for(j=0;j<m;j++)

   printf("%3d",alloc[i][j]);

  printf("\t");

  for(j=0;j<m;j++)

   printf("%3d",max[i][j]);

  printf("\t");

  for(j=0;j<m;j++)
```

```
   printf("%3d",need[i][j]);
  printf("\n");
 }printf("Available\n");
 for(j=0;j<m;j++)
  printf("%3c",65+j);
 printf("\n");
 for(j=0;j<m;j++)
  printf("%3d",avail[j]);
 printf("\n");
}
int check(int s)
{
 int i,j;
 i = s;
 do
 {
  if(!finish[i])
  {
   for(j=0;j<m;j++)
   {
    if(need[i][j]>work[j])
     break;
   }
   if(j==m) return i;
  }
  i=(i+1)%n;
 }while(i!=s);
 return -1;
}
void banker()
{
 int i,j,k=0;
 for(i=0;i<n;i++)
  finish[i]=0;

 for(j=0;j<m;j++)
  work[j] = avail[j];
 i=0;
 while((i=check(i))!=-1)
 {
  printf("Process P%d resource granted.\n",i);
  finish[i] = 1;
  for(j=0;j<m;j++)
   work[j] += alloc[i][j];
  printf("finish(");
  for(j=0;j<n;j++)
   printf("%d,",finish[j]);
  printf("\b)\nwork(");
  for(j=0;j<m;j++)
   printf("%d,",work[j]);
  printf("\b)\n")
  seq[k++]=i;
  i=(i+1)%n;
 }
 if(k==n)
 {
  printf("System is in safe state.\n");
  printf("Safe sequence:");
  for(j=0;j<n;j++)
   printf("P%d ",seq[j]);
 }
 else
 {
  printf("System is not in safe state.");
 }
 printf("\n");
```

```
}
int main()
{
 int i,j,pno;
 accept();
 calc_avail_accept();
 printf("\n");
 calc_need();
 print();
 banker()
 printf("Enter process no:");
 scanf("%d",&pno);
 printf("Enter resource request of process P%d\n",pno);
 for(j=0;j<m;j++)
 {
  printf("%c:",65+j);
  scanf("%d",&request[j]);
 }
 for(j=0;j<m;j++)
 {
  if(request[j]>need[pno][j])
   break;
 }
 if(j==m)
 {
  for(j=0;j<m;j++)
  {
   if(request[j]>avail[j])
    break;
  }
  if(j==m)
  {
   for(j=0;j<m;j++)
   {
    avail[j]-=request[j];
    alloc[pno][j]+=request[j];
    need[pno][j]-=request[j];
    print();
    banker();
   }
  }
else
   printf("Process P%d must wait.\n",pno);
 }
 else

  printf("Process P%d has exceeded its maximum claim\n",pno);

 return 0;

}
```

Slip 2

Q.2 Write the simulation program using SJF (non-preemptive). The arrival time and first CPU bursts of different jobs should be input to the system. Assume the fixed I/O waiting time (2units). The next CPU burst should be generated using random function. The output should give the Gantt chart, Turnaround Time and Waiting time for each process and average times. [20marks]

OR

Partially implement the Menu driven Banker's algorithm for accepting Allocation, Max from user.

a) Accept Available b) Display Allocation, Max c) Find Need and display It,d) Display Available Consider the system with 3 resources types A,B, and C with 7,2,6

instances respectively.

Consider the following snapshot:

Answer the following questions:

a) Display the contents of Available array?

b) Is there any deadlock? Print the message

```
#include<stdio.h>

#include<string.h>

int main()

{

   int num,b,i,j,n,m,Need[10][10],avail[10],max[10][10],alloc[10][10];

   printf("Enter the number of processes and resources:");

   scanf("%d%d",&i,&j);

   printf("Enter the elements for alloc martrix:");

   for(n=0;n<i;n++)

    for(m=0;m<j;m++)

   scanf("%d",&alloc[n][m]);

   printf("Enter the elements for max matrix:");

   for(n=0;n<i;n++)

    for(m=0;m<j;m++)

      scanf("%d",&max[n][m]);

  do{

     printf("Enter a case(1,2,3,4):");

     scanf("%d",&num);

    switch(num)

   {
```

```
case 1:

     //  printf("Enter the number of rows and col:");

      // scanf("%d%d",&i,&j);

       printf("Enter the resource instances:");

       //for(n=0;n<i;n++)

       for(b=0;b<j;b++)

       scanf("%d",&avail[b]);

       break;

case 2:

      printf("Allocation Matrix\n");

       for(n=0;n<i;n++)

       {

         for(m=0;m<j;m++)

         {

           printf("%d\t",alloc[n][m]);

         }

         printf("\n");

       }

       printf("Max Matrix\n");

        for(n=0;n<i;n++)

      {

        for(m=0;m<j;m++)
        {
          printf("%d\t",max[n][m]);
         }
       printf("\n");
      }
      break;
 case 3:
       printf("Need content Matrix\n");
       for(n=0;n<i;n++)
       {
         for(m=0;m<j;m++)
         {
           Need[n][m]=max[n][m]-alloc[n][m];
           printf("%d\t",Need[n][m]);
         }
         printf("\n");
       }
       break;
 case 4:
         for(b=0;b<j;b++)
          printf("%d\t",avail[b]);
       break;
```

```
    default:
      printf("Error");
    }
  }
  while(num<=4);
  return 0;
}
```

Slip 3

Q.2 Write the simulation program using FCFS. The arrival time and first CPU bursts of different jobs should be input to the system. Assume the fixed I/O waiting time (2 units). The next CPU burst should be generated using random function. The output should give the Gantt chart,Turnaround Time and Waiting time for each process and average times. [20 marks]

```
#include <stdio.h>
#include <string.h>
int n, Bu[20], Twt, Ttt, A[10], wt[10], w;
float Awt, Att;
char pname[20][20], c[20][20];
void getdata();
void gantt_chart();
void calculate();
void fcfs();
void getdata()
```

```
{
  int i;
  printf("\nEnter the number of processes:");
  scanf("%d", &n);
  for (i = 1; i <= n; i++)
  {
    printf("Enter the process Name:");
    scanf("%s", &pname[i]);
    printf("Enter the Arrival time %s=", pname[i]);
    scanf("%d", &A[i]);
    printf("Enter the Burst time %s=", pname[i]);
    scanf("%d", &Bu[i]);
  }
}
void gantt_chart()
{
  int i;
  // printf("\nGantt chart\n");
  for (i = 1; i <= n; i++)
    printf("  %s |", pname[i]);
  printf("\n--------------\n");
  printf("");
  for (i = 1; i <= n; i++)

  {

    printf("  %d  |", wt[i]);

    printf("  %d  |\n", wt[n] + Bu[n]);

  }

}

void calculate()

{

  int i;

  wt[1] = 0;

  for (i = 2; i <= n; i++)

  {

    wt[i] = Bu[i - 1] + wt[i - 1];

  }

  for (i = 1; i <= n; i++)

  {

    Twt = Twt + (wt[i] - A[i]);

    Ttt = Ttt + ((wt[i] + Bu[i]) - A[i]);

  }

  Att = (float)Ttt / n;

  Awt = (float)Twt / n;

  printf("\nAverage Turn around time=%3.2fms \n", Att);
  printf("\nAverage Waiting  time=%3.2fms \n", Awt);

}

void fcfs()

{

  int i, j, temp, temp1;

  Twt = 0;

  Ttt = 0;

  for (i = 1; i <= n; i++)

  {

    for (j = i + 1; j <= n; j++)

    {

      if (A[i] > A[j])

      {

        temp = Bu[i];

        temp1 = A[i];

        Bu[i] = Bu[j];

        A[i] = A[j];

        Bu[j] = temp;

        A[j] = temp1;

        strcpy(c[i], pname[i]);

        strcpy(pname[i], pname[j]);

        strcpy(pname[j], c[i]);
      }
    }
  }
  calculate();
  gantt_chart();
}
int main()
{
  getdata();
  fcfs();
}
```

OR

Q.2 Given an initial state of a 8-puzzle problem and

final state to be reached

Find the most cost-effective path to reach the final state from initial state using A* Algorithm in C/Python

```
def print_in_format(matrix):
    for i in range(9):
        if i%3 == 0 and i > 0:
            print("")
        print(str(matrix[i])+"")


def count(s):
    c = 0


    ideal = [1, 2, 3,
             8, 0, 4,
             7, 6, 5]


    for i in range(9):
        if s[i] != 0 and s[i] != ideal[i]:
            c += 1
    return c
```

```
def move(ar, p, st):

    rh = 999999
    store_st = list(st)

    for i in range(len(ar)):

        dupl_st = list(st)

        temp = dupl_st[p]
        dupl_st[p] = dupl_st[arr[i]]
        dupl_st[arr[i]] = temp
        tmp_rh = count(dupl_st)
        if tmp_rh < rh:
            rh = tmp_rh
            store_st = list(dupl_st)
    return store_st, rh
state = [2, 8, 3,
         1, 6, 4,
         7, 0, 5]
h = count(state)
Level = 1
```

```
print("\n------ Level "+str(Level)+" ------")
print_in_format(state)
print("\nHeuristic Value(Misplaced) : "+str(h))

while h>0:
    pos = int(state.index(0))
    Level += 1

    if pos == 0:
      arr = [1, 3]
      state, h = move(arr, pos, state)

    elif pos == 1:
        arr = [0, 2, 4]
        state, h = move(arr, pos, state)

    elif pos==2:
        arr = [1, 5]
        state, h = move(arr, pos, state)

    elif pos==3:
```

```
    arr = [0, 4, 6]
    state, h = move(arr, pos, state)

elif pos==4:
    arr = [1, 3, 5, 7]
    state, h = move(arr, pos, state)

elif pos==5:
    arr = [2, 4, 8]
    state, h = move(arr, pos, state)

elif pos==6:
    arr = [3, 7]
    state, h = move(arr, pos, state)

elif pos==7:
    arr = [4, 6, 8]
    state, h = move(arr, pos, state)

elif pos==8:
    arr = [5, 6]
    state, h = move(arr, pos, state)
print("\n------ Level "+str(Level)+" ------")

print_in_format(state)

print("\nHeuristic Value(Misplaced) : "+str(h))
```

Slip 4

Q.2 Write the program to simulate Non-preemptive

Priority scheduling. The arrival time and first CPU burst and priority for different n number of

processes should be input to the algorithm.

Assume the fixed IO waiting time (2 units). The next

CPU-burst should be generated randomly.

The output should give Gantt chart, turnaround time

and waiting time for each process. Also find

the average waiting time and turnaround time..

OR

Q.2 Write a C program to simulate Banker’s

algorithm for the purpose of deadlock avoidance.

Consider the following snapshot of system, A, B, C

and D are the resource type.a) Calculate and display the content of need matrix?b) Is the system in safe state? If display the safe

sequence.

c) If a request from process P arrives for (0, 4, 2, 0)

can it be granted immediately by keeping the

system in safe state. Print a message

```
#include<stdio.h>

#define MAX 10

int m,n,total[MAX],avail[MAX],alloc[MAX][MAX],

    max[MAX][MAX],need[MAX][MAX],work[MAX],finish[MAX],

    seq[MAX],request[MAX];
```

```
void accept()

{

 int i,j;

 printf("Enter no.of process:");

 scanf("%d",&n);

 printf("Enter no.of resource types:");

 scanf("%d",&m);

 printf("Enter total no.of resources of each resource type:\n");

 for(i=0;i<m;i++)

 {

  printf("%c:",65+i);

  scanf("%d",&total[i]);

 }

 printf("Enter no.of allocated resources of each resource type by each process:\n");

 for(i=0;i<n;i++)

 {

  printf("P%d:\n",i);

  for(j=0;j<m;j++)

  {

   printf("%c:",65+j);

   scanf("%d",&alloc[i][j]);
  }
 }
 printf("Enter no.of maximum resources of each resource type by each process:\n");
 for(i=0;i<n;i++)
 {
  printf("P%d:\n",i);
  for(j=0;j<m;j++)
  {
   printf("%c:",65+j);
   scanf("%d",&max[i][j]);
  }
 }
}
void calc_avail_accept()
{
 int b,j;
 printf("Enter the avail:");
 for(b=0;b<j;b++)
    scanf("%d",&avail[b]);
 for(b=0;b<j;b++)
    printf("%d\t",avail[b]);
}
```

```
void calc_need()
{
 int i,j;
 for(i=0;i<n;i++)
  for(j=0;j<m;j++)
   need[i][j]=max[i][j]-alloc[i][j];
}
void print()
{
 int i,j;
 printf("\tAllocation\tMax\tNeed\n\t");
 for(i=0;i<3;i++)
 {
  for(j=0;j<m;j++)
   printf("%3c",65+j);
  printf("\t");
 }
 printf("\n");
 for(i=0;i<n;i++)
 {
  printf("P%d\t",i);
  for(j=0;j<m;j++)
   printf("%3d",alloc[i][j]);
  printf("\t");
  for(j=0;j<m;j++)
   printf("%3d",max[i][j]);
  printf("\t");
  for(j=0;j<m;j++)
   printf("%3d",need[i][j]);
  printf("\n");
 }printf("Available\n");
 for(j=0;j<m;j++)
  printf("%3c",65+j);
 printf("\n");
 for(j=0;j<m;j++)
  printf("%3d",avail[j]);
 printf("\n");
}
int check(int s)
{
 int i,j;
 i = s;
 do
 {
  if(!finish[i])
  {
   for(j=0;j<m;j++)
   {
    if(need[i][j]>work[j])
     break;
   }
   if(j==m) return i;
  }
  i=(i+1)%n;
 }while(i!=s);
 return -1;
}
void banker()
{
 int i,j,k=0;
 for(i=0;i<n;i++)
  finish[i]=0;

 for(j=0;j<m;j++)
  work[j] = avail[j];
```

```
i=0;

while((i=check(i))!=-1)

{

 printf("Process P%d resource granted.\n",i);

 finish[i] = 1;

 for(j=0;j<m;j++)

  work[j] += alloc[i][j];

 printf("finish(");

 for(j=0;j<n;j++)

  printf("%d,",finish[j]);

 printf("\b)\nwork(");

 for(j=0;j<m;j++)

  printf("%d,",work[j]);

 printf("\b)\n")

 seq[k++]=i;

 i=(i+1)%n;

}

if(k==n)

{

 printf("System is in safe state.\n");

 printf("Safe sequence:");

 for(j=0;j<n;j++)
```

```
   printf("P%d ",seq[j]);
 }
 else
 {
  printf("System is not in safe state.");
 }
 printf("\n");
}
int main()
{
 int i,j,pno;
 accept();
 calc_avail_accept();
 printf("\n");
 calc_need();
 print();
 banker()
 printf("Enter process no:");
 scanf("%d",&pno);
 printf("Enter resource request of process P%d\n",pno);
 for(j=0;j<m;j++)
 {
```

```
 printf("%c:",65+j);
 scanf("%d",&request[j]);
}
for(j=0;j<m;j++)
{
 if(request[j]>need[pno][j])
  break;
}
if(j==m)
{
 for(j=0;j<m;j++)
 {
  if(request[j]>avail[j])
   break;
 }
 if(j==m)
 {
  for(j=0;j<m;j++)
  {
   avail[j]-=request[j];
   alloc[pno][j]+=request[j];
   need[pno][j]-=request[j];
```

```
    print();

    banker();

   }

  }

else

   printf("Process P%d must wait.\n",pno);

 }

 else

  printf("Process P%d has exceeded its maximum claim\n",pno);

 return 0;

}
```

Slip 5

Q.2 Write the simulation program to implement

demand paging and show the page scheduling

and total number of page faults for the following

given page reference string. Give input n as the

number of memory frames.

Reference String: 3, 4, 5, 6, 3, 4, 7, 3, 4, 5, 6, 7, 2, 4, 6

i. Implement FIFO

```
#include <stdio.h>

int main()

{

  int rstring[] = {3, 4, 5, 6, 3, 4, 7, 3, 4, 5, 6, 7, 2, 4, 6 };

  int pf = 0, frames, m, n, s, pages = 15;

  printf("\nFIFO Page Replacement Algorithm\n\n");

  printf("Enter no of frames: ");

  scanf("%d", &frames);

  int temp[frames];

  for (m = 0; m < frames; m++)

  {

    temp[m] = -1;

  }

  for (m = 0; m < pages; m++)

  {

    s = 0;

    for (n = 0; n < frames; n++)

    {

      if (rstring[m] == temp[n])

      {

        s++;

        pf--;
      }

    }

    pf++;

    if ((pf <= frames) && (s == 0))

    {

      temp[m] = rstring[m];

    }

    else if (s == 0)

    {

      temp[(pf - 1) % frames] = rstring[m];

    }

    printf("\n");

    // printf("%d\t", rstring[m]);

    for (n = 0; n < frames; n++)

    {

      if (temp[n] != -1)

        printf("%d\t", temp[n]);

      else

        printf("-\t");

    }

  }

  printf("\n\n Total Page Faults = %d\n\n", pf);
```

}

OR

Q.2 partially implement the Menu driven Banker's

algorithm for accepting Allocation, Max from

user.

a) Accept Available

b) Display Allocation, Max

c) Find Need and display It,

d) Display Available Consider the system with 3

resources types A,B, and C with 7,2,6 instances

respectively.

Consider the following snapshot:

Answer the following questions:

a) Display the contents of Available array?

b) Is there any deadlock? Print the message

Slip 6

Q.2 Write the simulation program to implement

demand paging and show the page scheduling

and total number of page faults for the following

given page reference string. Give input n as thenumber of memory frames.

Reference String :3, 4, 5, 6, 3, 4, 7, 3, 4, 5, 6, 7, 2, 4, 6

Implement FIFO

OR

Consider the following graph

The numbers written on edges represent the distance between the nodes.

The numbers written on nodes represent the heuristic value.

Implement A* algorithm in C/Python for above graph and find out most cost-effective path from A to J

```
def aStarAlgo(start_node, stop_node):

    open_set = set(start_node)

    closed_set = set()

    g = {}

    parents = {}

    g[start_node] = 0

    parents[start_node] = start_node


    while len(open_set) > 0:

      n = None
```

```
for v in open_set:
    if n == None or g[v] + heuristic(v) < g[n] + heuristic(n):
        n = v

if n == stop_node or Graph_nodes[n] == None:
    pass
else:
    for (m, weight) in get_neighbors(n):
        if m not in open_set and m not in closed_set:
            open_set.add(m)
            parents[m] = n
            g[m] = g[n] + weight
        else:
            if g[m] > g[n] + weight:
                g[m] = g[n] + weight
                parents[m] = n


                if m in closed_set:
                    closed_set.remove(m)
                    open_set.add(m)
```

```
      if n == None:
        print('Path does not exist!')
        return None

      if n == stop_node:
        path = []

        while parents[n] != n:
          path.append(n)
          n = parents[n]

        path.append(start_node)
        path.reverse()
        print('Path found:{0}'.format(path))
        return path

      open_set.remove(n)
      closed_set.add(n)
    print('Path does not exist!')
    return None

def get_neighbors(v):
    if v in Graph_nodes:

        return Graph_nodes[v]

    else:

        return None


def heuristic(n):

        H_dist = {

            'A': 10,

            'B': 8,

            'C': 5,

            'D': 7,

            'E': 3,

            'F': 6,

            'G': 5,

            'H':3,

            'I':1,

            'J':0

        }


        return H_dist[n]


Graph_nodes = {
    'A': [('B', 6), ('F', 3)],
    'B': [('C', 3),('D', 2),('A',6)],
    'C': [('B',3),('D',1),('E',5)],
    'D': [('B', 2),('C',1),('E',8)],
    'E': [('D', 8),('C',5),('J',5),('I',5)],
    'F': [('A',3),('G',1),('H',7)],
    'G': [('F',1),('I',3)],
    'H': [('F',7),('I',2)],
    'I': [('G',3),('H',2)],
    'J': [('E',5),('I',3)]
}
aStarAlgo('A', 'J')
```

Slip 7

Q.2 Write the simulation program using FCFS. The arrival time and first CPU bursts of different

jobs should be input to the system. Assume the

fixed I/O waiting time (2 units). The next CPU

burst should be generated using random function.

The output should give the Gantt chart,

Turnaround Time and Waiting time for each process

and average times

OR

Q.2 Write the simulation program to implement

demand paging and show the page schedulingand total number of page faults for the following

given page reference string. Give input n as thenumber of memory frames.

Reference String: 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2

i. Implement LRU

```
#include <stdio.h>

int main()

{

    int q[20], p[] = {7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2},



c = 0, c1, d, f, i, j, k = 0, n = 15, r, t,

b[20], c2[20];



    printf("Enter no of frames:");

    scanf("%d", &f);



    q[k] = p[k];

    printf("\n\t%d\n", q[k]);

    c++;

    k++;

    for (i = 1; i < n; i++)

    {

        c1 = 0;

        for (j = 0; j < f; j++)

        {
```

```
    if (p[i] != q[j])
        c1++;
}
if (c1 == f)
{
    c++;
    if (k < f)
    {
        q[k] = p[i];
        k++;
        for (j = 0; j < k; j++)
            printf("\t%d", q[j]);
        printf("\n");
    }
    else
    {
        for (r = 0; r < f; r++)
        {
            c2[r] = 0;
            for (j = i - 1; j < n; j--)
            {
                if (q[r] != p[j])
                    c2[r]++;
                else
                    break;
            }
        }
        for (r = 0; r < f; r++)
            b[r] = c2[r];
        for (r = 0; r < f; r++)
        {
            for (j = r; j < f; j++)
            {
                if (b[r] < b[j])
                {
                    t = b[r];
                    b[r] = b[j];
                    b[j] = t;
                }
            }
        }
        for (r = 0; r < f; r++)
        {
            if (c2[r] == b[0])
```

```
                        q[r] = p[i];

                    printf("\t%d", q[r]);

                }

                printf("\n");

            }

        }

    }

    printf("\n Total Page Faults = %d\n", c);

}
```

Slip 8

Q.2 Implement AO* algorithm in C /python for following graph and find out minimum cost solution

```
class Graph:
    def __init__(self, graph, heuristicNodeList,
startNode):

        self.graph = graph
        self.H=heuristicNodeList
        self.start=startNode
        self.parent={}
        self.status={}
        self.solutionGraph={}

    def applyAOStar(self):
        self.aoStar(self.start, False)

    def getNeighbors(self, v):

        return self.graph.get(v,'')

    def getStatus(self,v):
        return self.status.get(v,0)

    def setStatus(self,v, val):
        self.status[v]=val
    def getHeuristicNodeValue(self, n):

        return self.H.get(n,0)


    def setHeuristicNodeValue(self, n, value):

        self.H[n]=value


    def printSolution(self):


print("FOR GRAPH SOLUTION, TRAVERSE THE

GRAPH FROM THE START NODE:",self.start)

        print("----------------------------------------------------------")

        print(self.solutionGraph)

        print("----------------------------------------------------------")


    def computeMinimumCostChildNodes(self, v):

minimumCost=0

        costToChildNodeListDict={}

        costToChildNodeListDict[minimumCost]=[]

        flag=True

        for nodeInfoTupleList in self.getNeighbors(v):

            cost=0
```

```
      nodeList=[]

      for c, weight in nodeInfoTupleList:

        cost=cost+self.getHeuristicNodeValue(c)+weight

        nodeList.append(c)

      if flag==True:

        minimumCost=cost

        costToChildNodeListDict[minimumCost]=nodeList

        flag=False

      else:

        if minimumCost>cost:

          minimumCost=cost

          costToChildNodeListDict[minimumCost]=nodeList


    return minimumCost,

costToChildNodeListDict[minimumCost]


  def aoStar(self, v, backTracking):

    print("HEURISTIC VALUES :", self.H)

    print("SOLUTION GRAPH :", self.solutionGraph)

    print("PROCESSING NODE :", v)

    print("----------------------------------------------------------------------------------------")

    if self.getStatus(v) >= 0:
        minimumCost, childNodeList =
self.computeMinimumCostChildNodes(v)
        print(minimumCost, childNodeList)
        self.setHeuristicNodeValue(v, minimumCost)
self.setStatus(v,len(childNodeList))
        solved=True
        for childNode in childNodeList:
           self.parent[childNode]=v
           if self.getStatus(childNode)!=-1:
              solved=solved & False
        if solved==True:
           self.setStatus(v,-1)
           self.solutionGraph[v]=childNodeList
        if v!=self.start:
           self.aoStar(self.parent[v], True)
        if backTracking==False:
           for childNode in childNodeList:
              self.setStatus(childNode,0)
              self.aoStar(childNode, False)


print ("Graph - 1")
h1 = {'A': 8, 'B': 1, 'C': 2, 'D': 8, 'E': 1, 'F': 0}
```

```
graph1 ={

     'A':[[('B',4), ('C',5)],[('D',5)]],

     'B':[[('C',2)]],

     'C':[[('E',2)]],

     'D':[[('E',2),('F',4)]],

     'E':[[('F',3)]]

}

G1= Graph(graph1, h1, 'A')

G1.applyAOStar()

G1.printSolution()
```

OR

Q.2. Write the simulation program to implement demand paging and show the page schedulingand total number of page faults for the following given page reference string. Give input n =3 as the number of memory frames.

ReferenceString12,15,12,18,6,8,11,12,19,12,6,8,12,11

19,8

Implement OPT

```
#include <stdio.h>
int main()
{
  int no_of_frames, no_of_pages = 16, frames[10],
pages[] = {12, 15, 12, 18, 6, 8, 11, 12, 19, 12, 6, 8, 12,
15, 19, 8},
temp[10], flag1, flag2, flag3, i, j, k, pos, max,
faults = 0;
  printf("Enter number of frames: ");
  scanf("%d", &no_of_frames);

  for (i = 0; i < no_of_frames; ++i)
  {
    frames[i] = -1;
```

```
}

for (i = 0; i < no_of_pages; ++i)
{
 flag1 = flag2 = 0;

 for (j = 0; j < no_of_frames; ++j)
 {
  if (frames[j] == pages[i])
  {
   flag1 = flag2 = 1;
   break;
  }
 }

 if (flag1 == 0)
 {
  for (j = 0; j < no_of_frames; ++j)
  {
   if (frames[j] == -1)
   {
    faults++;
```

```
     frames[j] = pages[i];
     flag2 = 1;
     break;
    }
   }
  }
if (flag2 == 0)
  {
   flag3 = 0;

   for (j = 0; j < no_of_frames; ++j)
   {
    temp[j] = -1;

    for (k = i + 1; k < no_of_pages; ++k)
    {
     if (frames[j] == pages[k])
     {
      temp[j] = k;
      break;
     }
    }
```

```
}

for (j = 0; j < no_of_frames; ++j)
{
  if (temp[j] == -1)
  {
    pos = j;
    flag3 = 1;
    break;
  }
}

if (flag3 == 0)
{
  max = temp[0];
  pos = 0;

  for (j = 1; j < no_of_frames; ++j)
  {
    if (temp[j] > max)
    {
      max = temp[j];
      pos = j;
     }
    }
   }
   frames[pos] = pages[i];
   faults++;
  }

  printf("\n");

  for (j = 0; j < no_of_frames; ++j)
  {
   printf("%d\t", frames[j]);
  }
 }

 printf("\n\nTotal Page Faults = %d\n\n", faults);

 return 0;
}
```

Slip 9

Q.2 Partially implement the Menu driven Banker's

algorithm for accepting Allocation, Max from

user.

a) Accept Available

b) Display Allocation, Max

c) Find Need and display It,

d) Display Available Consider the system with 3

resources types A,B, and C with 7,2,6 instances

respectively.

Consider the following snapshot:

OR

Q.2 Write the program to simulate Round Robin (RR)

scheduling. The arrival time and first CPUburst for

different n number of processes should be input to

the algorithm. Also give the time

quantum as input. Assume the fixed IO waiting

time(2 units). The next CPU-burst should begenerated randomly. The output should give Gantt

chart, turnaround time and waiting time for each

process. Also find the average waiting

timeandturnaround time.

Slip 10

Q.2 Write the simulation program to

implementdemand paging and show the page scheduling and

total number of page faults for the following given

page reference string. Give input n=3 as thenumber of memory frames.

Reference String :

12,15,12,18,6,8,11,12,19,12,6,8,12,15,19,8

Implement OPT

OR

Q.2 Write the simulation program using FCFS. The

arrival time and first CPU bursts of different

jobs should be input to the system. Assume the

fixed I/O waiting time (2 units). The next CPU

burst should be generated using random function.

The output should give the Gantt chart,

Turnaround Time and Waiting time for each process and average times.

Slip 11

Q.2 Write the simulation program to implement

demand paging and show the page scheduling

and total number of page faults for the following

given page reference string. Give input n as thenumber of memory frames.

Reference String: 0, 2, 1, 6, 4, 0, 1, 0, 3, 1, 2, 1

Implement FIFO

```
#include <stdio.h>

int main()

{

  int rstring[] = {0, 2, 1, 6, 4, 0, 1, 0, 3, 1, 2, 1};

  int pf = 0, frames, m, n, s, pages = 12;

  printf("\nFIFO Page Replacement Algorithm\n\n");
  printf("Enter no of frames: ");

  scanf("%d", &frames);


  int temp[frames];


  for (m = 0; m < frames; m++)

  {

    temp[m] = -1;

  }

  for (m = 0; m < pages; m++)

  {

    s = 0;

    for (n = 0; n < frames; n++)

    {

      if (rstring[m] == temp[n])

      {

        s++;

        pf--;

      }

    }

    pf++;
```

```
    if ((pf <= frames) && (s == 0))
    {
      temp[m] = rstring[m];
    }
    else if (s == 0)
    {
      temp[(pf - 1) % frames] = rstring[m];
    }
    printf("\n");
    // printf("%d\t", rstring[m]);
    for (n = 0; n < frames; n++)
    {
      if (temp[n] != -1)
        printf("%d\t", temp[n]);
      else
        printf("-\t");
    }
  }

  printf("\n\n Total Page Faults = %d\n\n", pf);
}
```

OR

Consider the following graphThe numbers written on edges represent the

distance between the nodes.

The numbers written on nodes represent the

heuristic value.

Implement A* algorithm in C/Python for above graph

and find out most cost-effective path

from A to J.

Slip 12

Q.2 Write the simulation program to implement

demand paging and show the page

scheduling and total number of page faults for the

following given page reference string.

Give input n as the number of memory frames.

Reference String :

12,15,12,18,6,8,11,12,19,12,6,8,12,15,19,8

Implement OPT

OR

Q2. Write the simulation program using FCFS. The

arrival time and first CPU bursts of different

jobs should be input to the system. Assume the fixed I/O waiting time (2 units). The next CPU burst

should be generated using random function. The

output should give the Gantt chart,

Turnaround Time and Waiting time for each process

and average times.

Slip 13

Q.2 Write a C program to simulate Banker's

algorithm for the purpose of deadlockavoidance. Consider the following snapshot of

system, A, B, C and D are the resource type.

a) Calculate and display the content of need matrix?

b) Is the system in safe state? If display the safe

sequence.

c) If a request from process P arrives for (0, 4, 2, 0)

can it be granted immediately by keeping

the system in safe state. Print a message.

OR

Write the simulation program using SJF(non-preemptive). The arrival time and first CPU bursts of different jobsshould be input to the system. The Assume the fixed I/O waiting time (2 units).Thenext CPU burst should be generated using random function. The output should give the Gantt chart, Turnaround Time and Waiting time for each process and average times.

Slip 14

Q.2 Write the simulation program to implement

demand paging and show the page schedulingand total number of page faults for the

following given page reference string.

Give input n =3 as the number of memory frames.

Reference String : 0, 2, 1, 6, 4, 0, 1, 0, 3, 1, 2, 1

Implement FIFO

OR

Write the simulation program using SJF(non-

preemptive). The arrival time and first CPU bursts of

different jobs should be input to the system. The

Assume the fixed I/O waiting time (2 units).Thenext CPU burst should be generated using random

function. The output should give the Gantt chart,

Turnaround Time and Waiting time for each process and average times.

Slip 15

Q.2 Write the simulation program to implement

demand paging and show the page schedulingand total number of page faults for the following

given page reference string. Give input n as the

number of memory frames.

Reference String :7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2

Implement LRU

OR

Write the program to simulate Preemptive Shortest Job First (SJF) -scheduling. The arrival time and first CPU-burst for different n number of processes should be input to the algorithm. Assumethe fixed IO waiting time (2 units). The nextCPU-burstshould be generated randomly. The outputshould give Gantt chart, turnaround time and waiting time for each process. Also find the averagewaiting time and turnaround time.

Slip 16

Q.2 Write the simulation program to implement

demand paging and show the page scheduling

and total number of page faults for the following

given page reference string. Give input n =3 as

the number of memory frames.

Reference String :

12,15,12,18,6,8,11,12,19,12,6,8,12,15,19,8

Implement OPT

OR

Given an initial state of a 8-puzzle problem and final state to be reached

Find the most cost-effective path to reach the final

state from initial state using A* Algorithm

in C/Python.

Consider g(n) = Depth of node and h(n) = Number of

misplaced tiles.

Slip 17

Q.2 Write the simulation program to implement

demand paging and show the page scheduling

and total number of page faults for the following

given page reference string. Give input n=3 as

the number of memory frames.

Reference String :

12,15,12,18,6,8,11,12,19,12,6,8,12,15,19,8

Implement OPT

OR

Write the simulation program using FCFS. The arrival

time and first CPU bursts of differentjobs should be input to the system. Assume the

fixed I/O waiting time (2 units). The next

CPU burst should be generated using random

function. The output should give the Ganttchart, Turnaround Time and Waiting time for each

process and average

Slip 18

Q.2 Write the simulation program to implement

demand paging and show the page scheduling

and total number of page faults for the following

given page reference string. Give input n as

the number of memory frames.Reference String :

12,15,12,18,6,8,11,12,19,12,6,8,12,15,19,8

Implement OPT

OR

Write the simulation program using SJF (non-preemptive). The arrival time and first CPU bursts of

different jobs should be input to the system. The

Assume the fixed I/O waiting time (2 units).Thenext

CPU burst should be generated using random

function. The output should give the Gantt chart,

Turnaround Time and Waiting time for each process

and average times.

Slip 19

Q.2 Write the program to simulate Non-preemptive

Priority scheduling. The arrival time and

first CPU burst and priority for different n number of

processes should be input to the algorithm.

Assume the fixed IO waiting time (2 units). The next

CPU-burst should be generated randomly.The output should give Gantt chart, turnaround time

and waiting time for each process. Also find the average waiting time and turnaround time

OR

Write a C program to simulate Banker's algorithm for

the purpose of deadlock

avoidance. Consider the following snapshot of

system, A, B, C and D are the resource type.

a) Calculate and display the content of need matrix?

b) Is the system in safe state? If display the safe

sequence.

c) If a request from process P arrives for (0, 4, 2, 0)

can it be granted immediately by keeping the

system in safe state. Print a message

Slip 20

Q.2 Write the simulation program to implement

demand paging and show the page scheduling

and total number of page faults for the following

given page reference string. Give input n=3 as

the number of memory frames.

Reference String : 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2

i. Implement LRU

OR

Write the simulation program using FCFS. The arrival

time and first CPU bursts of different jobs

should be input to the system. Assume the fixed I/O waiting time (2 units). The next CPU burst should be generated using random function. The output should give the Gantt chart, Turnaround Time and Waiting time for each process and average times