**BANKERS ALGORITHM**

**EXPT NO: 5 DATE: 4/11/22**

**AIM**

To implement Bankers Algorithm

**THEORY**

The Banker’s algorithm is a resource allocation and deadlock avoidance algorithm developed by Edsger Dijkstra that tests for safety by simulating the allocation of pre-determined maximum possible amounts of all resources, and then makes a ”safe-state” check to test for possible deadlock conditions for all other pending activities, before deciding whether allocation should be allowed to continue. The Banker’s algorithm is run by the operating system whenever a process requests resources. The algorithm prevents deadlock by denying or postponing the request if it determines that accepting the request could put the system in an unsafe state (one where deadlock could occur).

The following data structures are needed:

* n represents the number of processes and m represents the number of resource types.
* Available

1. A vector (array) of available resources of each type
2. If available[j] = k, then k instances of Rj are available.

* Max

1. An n by m matrix
2. Defines maximum demand for each process
3. Max[i][j] = k, then process Pi may request at most k instances of resource Rj.

* Allocation

1. An n by m matrix
2. Defines number of resources of each type currently allocated to each process
3. Allocation[i][j]=k, then process Pi is currently allocated k instances of Rj.

* Need

1. An n by m matrix
2. Indicates remaining resource need of each process
3. If need[i][j] = k, then process Pi needs k more instances of Rj.
4. Need[i][j]= Max[i][j]-allocation[i][j]

The safety algorithm is used to determine if a system is in a safe state. It works as follows:

1. Work and finish: vectors of length m and n respectively Initialize work = available Finish[i] = false for I = 0,1,9,n-1
2. Find an index I such that both a. Finish[i] = false b. Needi ≤work If no such i exists, go to step 4.
3. Work = work + allocationi Finish[i] = true. Go to step 2.
4. If all finish[i] = true, then the system is in a safe state.

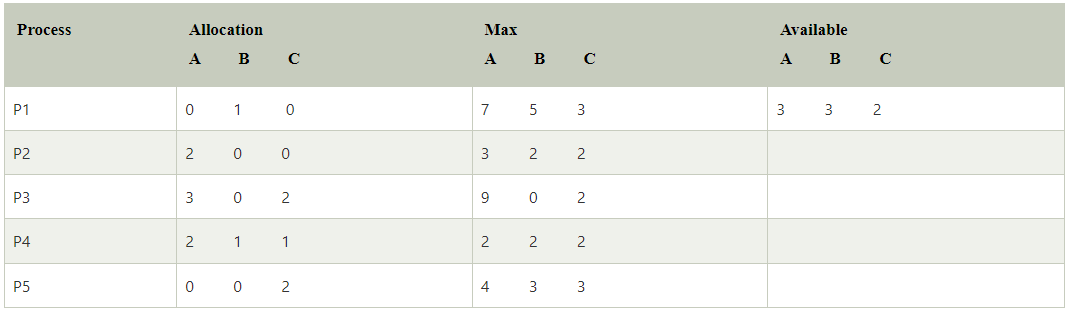
**Merits:**

Following are the essential characteristics of the Banker's algorithm:

1. It contains various resources that meet the requirements of each process.
2. Each process should provide information to the operating system for upcoming resource requests, the number of resources, and how long the resources will be held.
3. It helps the operating system manage and control process requests for each type of resource in the computer system.
4. The algorithm has a Max resource attribute that represents indicates each process can hold the maximum number of resources in a system.

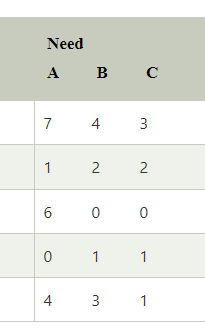
**Demerits:**

1. It requires a fixed number of processes, and no additional processes can be started in the system while executing the process.
2. The algorithm does no longer allows the processes to exchange its maximum needs while processing its tasks.
3. Each process has to know and state their maximum resource requirement in advance for the system.
4. The number of resource requests can be granted in a finite time, but the time limit for allocating the resources is one year.



Now we check if each type of resource request is available for each process.

**Step 1:** For Process P1:

Need <= Available

7, 4, 3 <= 3, 3, 2 condition is **false**.

**So, we examine another process, P2.**

**Step 2:** For Process P2:

Need <= Available

1, 2, 2 <= 3, 3, 2 condition **true**

New available = available + Allocation

(3, 3, 2) + (2, 0, 0) => 5, 3, 2

**Similarly, we examine another process P3, P4, P5.**

**SOURCE CODE**

##include<iostream>

#define MAX\_PROCESSES 20

#define MAX\_RESOURCES 5

using namespace std;

int AVAILABLE[MAX\_PROCESSES][MAX\_RESOURCES];

int REMAINING\_NEED[MAX\_PROCESSES][MAX\_RESOURCES];

int ALLOCATION[MAX\_PROCESSES][MAX\_RESOURCES];

int MAX\_NEED[MAX\_PROCESSES][MAX\_RESOURCES];

int TOTAL\_RESOURCES[MAX\_RESOURCES];

int INITIAL\_TOTAL[MAX\_RESOURCES],REQUEST[MAX\_RESOURCES],R\_index;

bool FLAG[MAX\_PROCESSES];

int N,RESOURCES;

int NotInSafeSequence=false;

int sep=0,InfiniteLoop;

int SAFE\_SEQ[MAX\_PROCESSES];

void BANKERS\_INPUT()

{

int i,j;

cout<<"ENTER NUMBER OF PROCESSES: ";

cin>>N;

cout<<"ENTER NUMBER OF RESOURCES: ";

cin>>RESOURCES;

cout<<"ENTER TOTAL RESOURCE VALUES RESPECTIVELY: ";

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

{

cin>>TOTAL\_RESOURCES[i];

}

cout<<"ENTER ALLOCATION MATRIX:\n";

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

{

cout<<"P"<<i+1<<": ";

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

{

cin>>ALLOCATION[i][j];

}

}

cout<<"ENTER MAX NEED MATRIX:\n";

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

{

cout<<"P"<<i+1<<": ";

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

{

cin>>MAX\_NEED[i][j];

}

}

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

{

FLAG[i]=false;

}

}

void Calc\_INITIALLY\_AVAILABLE()

{

int sum;

for(int i=0;i<RESOURCES;i++)

{

sum=0;

for(int j=0;j<N;j++)

{

sum+=ALLOCATION[j][i];

}

INITIAL\_TOTAL[i]=sum;

}

for(int i=0;i<RESOURCES;i++)

{ AVAILABLE[0][i]=TOTAL\_RESOURCES[i]-INITIAL\_TOTAL[i];}

}

void Calc\_REMAINING\_NEED()

{

int i,j;

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

{

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

{

REMAINING\_NEED[i][j]=MAX\_NEED[i][j]-ALLOCATION[i][j];

}

}

}

int CHECK\_if\_all\_true()

{

InfiniteLoop++;

for(int i=0;i<N;i++)

{

if(FLAG[i]==false)

return 0;

}

return 1;

}

void BANKERS\_ALGO()

{

int fullfilled;InfiniteLoop=0;int s=0;

Calc\_REMAINING\_NEED();

Calc\_INITIALLY\_AVAILABLE();

for(int i=0;i<N;i++)

{

FLAG[i]=false;

}

sep=0;

while(1){

for(int i=0;i<N;i++)

{

fullfilled=0;

if(FLAG[i]==false)

{

for(int j=0;j<RESOURCES;j++)

{

if(REMAINING\_NEED[i][j]<=AVAILABLE[sep][j])

{fullfilled++;}

}

if(fullfilled==RESOURCES)

{

FLAG[i]=true;

SAFE\_SEQ[s++]=i;

// cout<<"P"<<i<<" ";

for(int j=0;j<RESOURCES;j++)

{

AVAILABLE[sep+1][j]=AVAILABLE[sep][j]+ALLOCATION[i][j];

}

sep++;

}

}

}

if(InfiniteLoop>N+40)

{

NotInSafeSequence=true;

break;

}

if(CHECK\_if\_all\_true())

break;

}

}

void PRINT\_SAFEsequence()

{

cout<<"\nSAFE SEQUENECE: ";

for(int i=0;i<N;i++)

{

cout<<"P"<<SAFE\_SEQ[i]<<" ";

}

}

void REQUEST\_HANDLER()

{

for(int i=0;i<RESOURCES;i++)

{

if(REQUEST[i]>REMAINING\_NEED[R\_index][i])

{

cout<<"PROCESS NEED CROSSES MAXIMUM CLAIM FOR RESOURCES\n";

exit(1);

}

if(REQUEST[i]>AVAILABLE[0][i])

{

cout<<"PROCESS MUST WAIT, RESOURCES NOT AVAILABLE\n";

exit(1);

}

}

for(int i=0;i<RESOURCES;i++)

{

AVAILABLE[0][i]-=REQUEST[i];

ALLOCATION[R\_index][i]+=REQUEST[i];

REMAINING\_NEED[R\_index][i]-=REQUEST[i];

}

}

void RESTORE\_SAFE()

{

for(int i=0;i<RESOURCES;i++)

{

AVAILABLE[0][i]+=REQUEST[i];

ALLOCATION[R\_index][i]-=REQUEST[i];

REMAINING\_NEED[R\_index][i]+=REQUEST[i];

}

}

void PRINT\_matrices()

{

cout<<"\n\nPROCESS NUMBER "<<"ALLOCATION "<<"AVAILABLE "<<"NEED "<<endl;

int i;

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

{

cout.setf(ios::left,ios::adjustfield);

cout.width(19);

cout<<i;

for(int j=0;j<RESOURCES;j++)

{

cout<<ALLOCATION[i][j]<<" ";

}

cout<<" ";

for(int j=0;j<RESOURCES;j++)

{

cout<<AVAILABLE[i][j]<<" ";

}

cout<<" ";

for(int j=0;j<RESOURCES;j++)

{

cout<<REMAINING\_NEED[i][j]<<" ";

}

cout<<endl;

}

cout<<" ";

for(int j=0;j<RESOURCES;j++)

{

cout<<AVAILABLE[i][j]<<" ";

}

cout<<endl;

}

int main()

{

BANKERS\_INPUT();

BANKERS\_ALGO();

if(NotInSafeSequence==true)

{

cout<<"NOT IN SAFE SEQUENCE";

exit(1);

}

else

{

cout<<"\nIN SAFE SEQUENCE\n";

PRINT\_matrices();

PRINT\_SAFEsequence();

}

cout<<"\nENTER REQUEST PROCESS NO: ";

cin>>R\_index;

cout<<"ENTER REQUEST: ";

for(int i=0;i<RESOURCES;i++)

{

cin>>REQUEST[i];

}

REQUEST\_HANDLER();

BANKERS\_ALGO();

if(NotInSafeSequence==true)

{

cout<<"\n\nREQUEST CANNOT BE GRANTED, HENCE RESTORED";

RESTORE\_SAFE();

BANKERS\_ALGO();

PRINT\_matrices();

PRINT\_SAFEsequence();

}

else

{

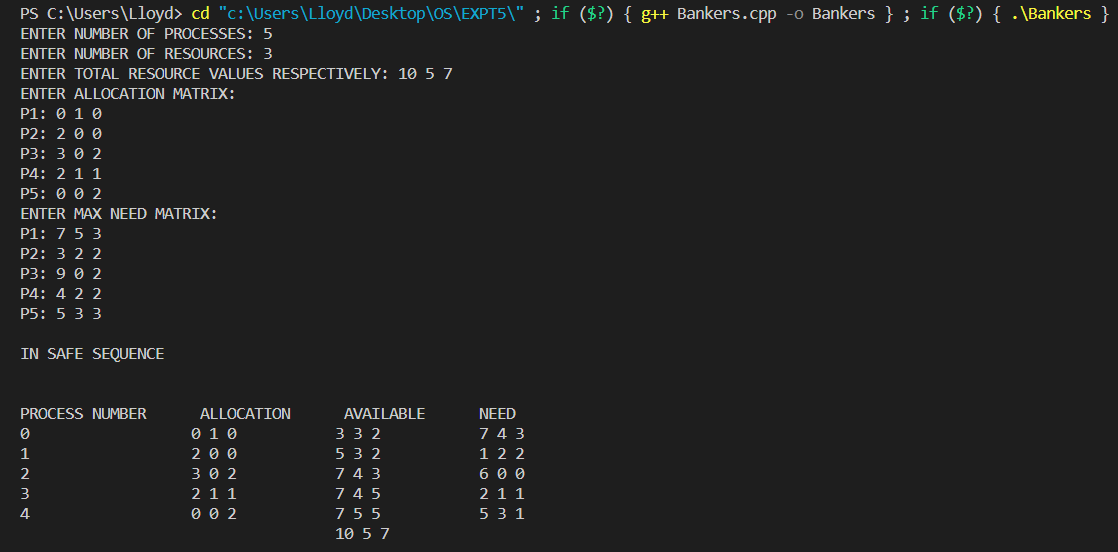
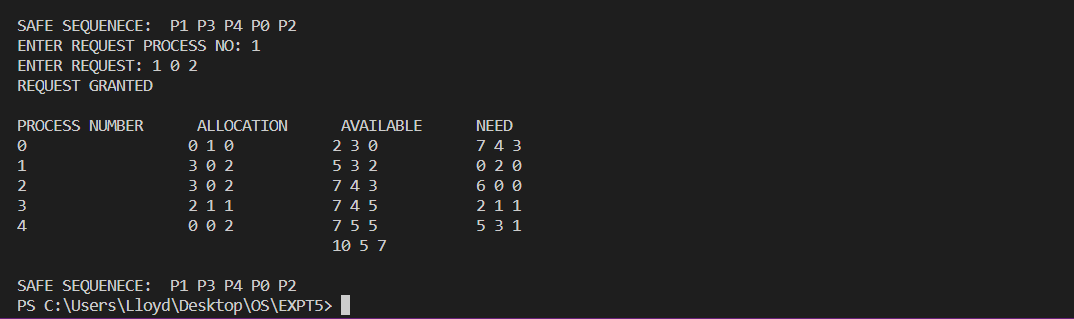
cout<<"REQUEST GRANTED";

PRINT\_matrices();

PRINT\_SAFEsequence();

}

}

**OUTPUT**

**CONCLUSION**

The Bankers algorithm was comprehended and was successfully implemented.