// Program to check whether given system is in safe state or not using Banker’s deadlock avoidance algorithm )

#include <stdio.h>

#include <stdbool.h>

#define MAX\_PROCESSES 10

#define MAX\_RESOURCES 10

int processes, resources;

int available[MAX\_RESOURCES];

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

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

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

void calculateNeed() {

for (int i = 0; i < processes; i++) {

for (int j = 0; j < resources; j++) {

need[i][j] = max[i][j] - allocation[i][j];

}

}

}

bool isSafeState() {

int work[MAX\_RESOURCES];

bool finish[MAX\_PROCESSES] = {false};

int safeSequence[MAX\_PROCESSES];

int count = 0;

// Initialize work as a copy of available resources

for (int i = 0; i < resources; i++) {

work[i] = available[i];

}

while (count < processes) {

bool found = false;

for (int i = 0; i < processes; i++) {

if (!finish[i]) {

bool canAllocate = true;

for (int j = 0; j < resources; j++) {

if (need[i][j] > work[j]) {

canAllocate = false;

break;

}

}

if (canAllocate) {

for (int k = 0; k < resources; k++) {

work[k] += allocation[i][k];

}

safeSequence[count++] = i;

finish[i] = true;

found = true;

}

}

}

if (!found) {

printf("System is not in a safe state.\n");

return false;

}

}

printf("System is in a safe state.\nSafe sequence is: ");

for (int i = 0; i < processes; i++) {

printf("P%d ", safeSequence[i]);

}

printf("\n");

return true;

}

int main() {

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

scanf("%d", &processes);

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

scanf("%d", &resources);

printf("Enter the available resources:\n");

for (int i = 0; i < resources; i++) {

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

}

printf("Enter the maximum resource matrix:\n");

for (int i = 0; i < processes; i++) {

for (int j = 0; j < resources; j++) {

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

}

}

printf("Enter the allocation matrix:\n");

for (int i = 0; i < processes; i++) {

for (int j = 0; j < resources; j++) {

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

}

}

calculateNeed();

isSafeState();

return 0;

}