#include <stdio.h>

int max[100][100];

int alloc[100][100];

int need[100][100];

int avail[100];

int n, r;

void input();

void show();

void calculate();

int main() {

printf("\*\*\*\*\* Banker's Algorithm \*\*\*\*\*\n");

input();

show();

calculate();

return 0;

}

void input() {

int i, j;

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

scanf("%d", &n);

printf("Enter the number of resource instances: ");

scanf("%d", &r);

printf("Enter the Max Matrix:\n");

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

for (j = 0; j < r; j++) {

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

}

}

printf("Enter Allocation Matrix:\n");

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

for (j = 0; j < r; j++) {

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

}

}

printf("Enter Available Resources:\n");

for (j = 0; j < r; j++) {

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

}

// Calculate the Need Matrix

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

for (j = 0; j < r; j++) {

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

}

}

}

void show() {

int i, j;

printf("\nProcess\t Allocation\t Max\t\t Need\t\t Available\n");

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

printf("P%d\t", i + 1);

// Allocation Matrix

for (j = 0; j < r; j++) {

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

}

printf("\t\t");

// Max Matrix

for (j = 0; j < r; j++) {

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

}

printf("\t\t");

// Need Matrix

for (j = 0; j < r; j++) {

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

}

// Available resources (only for the first process)

if (i == 0) {

printf("\t\t");

for (j = 0; j < r; j++) {

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

}

}

printf("\n");

}

}

void calculate() {

int finish[100] = {0}; // Finish array to mark completion of processes

int safe[100]; // Safe sequence array

int work[100]; // Work array to track available resources

int count = 0; // Count for the number of processes in the safe sequence

int i, j, process;

// Initialize work with available resources

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

work[i] = avail[i];

}

printf("\nSafe sequence is: ");

while (count < n) {

int found = 0;

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

if (finish[i] == 0) {

int can\_allocate = 1;

for (j = 0; j < r; j++) {

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

can\_allocate = 0;

break;

}

}

// If resources can be allocated to process i

if (can\_allocate) {

for (j = 0; j < r; j++) {

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

}

safe[count++] = i;

finish[i] = 1;

found = 1;

printf("P%d ", i + 1);

}

}

}

// If no process could be allocated in this iteration, system is in unsafe state

if (!found) {

printf("\nThe System is in an unsafe state\n");

return;

}

}

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

}