#include <stdio.h>

#define A 0

#define B 1

#define C 2

#define MAX\_NODES 1000

#define O 1000000000

int n;

int e;

int capacity[MAX\_NODES][MAX\_NODES];

int flow[MAX\_NODES][MAX\_NODES];

int color[MAX\_NODES];

int pred[MAX\_NODES];

int min(int x, int y) {

return x < y ? x : y;

}

int head, tail;

int q[MAX\_NODES + 2];

void enqueue(int x) {

q[tail] = x;

tail++;

color[x] = B;

}

int dequeue() {

int x = q[head];

head++;

color[x] = C;

return x;

}

// Using BFS as a searching algorithm

int bfs(int start, int target) {

int u, v;

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

color[u] = A;

}

head = tail = 0;

enqueue(start);

pred[start] = -1;

while (head != tail) {

u = dequeue();

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

if (color[v] == A && capacity[u][v] - flow[u][v] > 0) {

enqueue(v);

pred[v] = u;

}

}

}

return color[target] == C;

}

// Applying fordfulkerson algorithm

int fordFulkerson(int source, int sink) {

int i, j, u;

int max\_flow = 0;

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

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

flow[i][j] = 0;

}

}

// Updating the residual values of edges

while (bfs(source, sink)) {

int increment = O;

for (u = n - 1; pred[u] >= 0; u = pred[u]) {

increment = min(increment, capacity[pred[u]][u] - flow[pred[u]][u]);

}

for (u = n - 1; pred[u] >= 0; u = pred[u]) {

flow[pred[u]][u] += increment;

flow[u][pred[u]] -= increment;

}

// Adding the path flows

max\_flow += increment;

}

return max\_flow;

}

int main() {

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

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

capacity[i][j] = 0;

}

}

n = 6;

e = 7;

capacity[0][1] = 8;

capacity[0][4] = 3;

capacity[1][2] = 9;

capacity[2][4] = 7;

capacity[2][5] = 2;

capacity[3][5] = 5;

capacity[4][2] = 7;

capacity[4][3] = 4;

int s = 0, t = 5;

printf("Max Flow: %d\n", fordFulkerson(s, t));

}