1.Scalability Issue

#include <stdio.h>

#include <stdlib.h>

int main() {

int max\_users = 500000;

int num\_users;

printf("Enter the number of concurrent users (up to %d): ", max\_users);

scanf("%d", &num\_users);

if (num\_users < 0) {

printf("Invalid input! Number of users cannot be negative.\n");

return 1;

}

if (num\_users <= max\_users) {

printf("Platform is stable. Handling %d users.\n", num\_users);

return 0;

} else {

printf("ERROR: Platform CRASH! Too many users: %d (Max: %d)\n", num\_users, max\_users);

return 1;

}

}

2.Recommendation Algorithm Failure

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

int main() {

srand(time(NULL));

int failed\_recommendations = 0;

int total\_recommendations = 100;

for (int i = 0; i < total\_recommendations; i++) {

double probability = (double)rand() / RAND\_MAX;

if (probability < 0.02) {

failed\_recommendations++;

}

}

printf("Total Recommendations: %d\n", total\_recommendations);

printf("Failed Recommendations: %d\n", failed\_recommendations);

printf("Failure Rate: %.2f%%\n", (double)failed\_recommendations / total\_recommendations \* 100);

return 0;

}

3.Inventory Optimization

#include <stdio.h>

#include <stdlib.h>

#define WAREHOUSES 10

void optimize\_inventory(int capacities[], int products, int warehouse\_count) {

int dp[products + 1][warehouse\_count + 1];

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

for (int j = 0; j <= warehouse\_count; j++) {

if (i == 0 || j == 0) {

dp[i][j] = 0;

} else if (i <= capacities[j - 1]) {

dp[i][j] = i;

} else {

dp[i][j] = dp[capacities[j - 1]][j - 1];

}

}

}

printf("Optimal inventory allocation:\n");

for (int j = 1; j <= warehouse\_count; j++) {

printf("Warehouse %d: %d products\n", j, dp[products][j]);

products -= dp[products][j];

}

}

int main() {

int capacities[WAREHOUSES] = {100, 150, 200, 250, 300, 350, 400, 450, 500, 550};

int total\_products;

printf("Enter the total number of products to allocate: ");

scanf("%d", &total\_products);

optimize\_inventory(capacities, total\_products, WAREHOUSES);

return 0;

}

4.Logistic and Supply Chain Optimization

#include <stdio.h>

#include <limits.h>

#define WAREHOUSES 5

#define LOCATIONS 10

#define V (WAREHOUSES + LOCATIONS)

void dijkstra(int graph[V][V], int src) {

int dist[V];

int visited[V] = {0};

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

dist[i] = INT\_MAX;

}

dist[src] = 0;

for (int count = 0; count < V - 1; count++) {

int min\_index = -1;

int min\_value = INT\_MAX;

for (int v = 0; v < V; v++) {

if (!visited[v] && dist[v] < min\_value) {

min\_value = dist[v];

min\_index = v;

}

}

visited[min\_index] = 1;

for (int v = 0; v < V; v++) {

if (!visited[v] && graph[min\_index][v] && dist[min\_index] != INT\_MAX &&

dist[min\_index] + graph[min\_index][v] < dist[v]) {

dist[v] = dist[min\_index] + graph[min\_index][v];

}

}

}

printf("Shortest distances from Warehouse %d:\n", src + 1);

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

if (dist[i] == INT\_MAX) {

printf("To Location %d: Unreachable\n", i - WAREHOUSES + 1);

} else {

printf("To Location %d: %d\n", i - WAREHOUSES + 1, dist[i]);

}

}

printf("\n");

}

int main() {

int graph[V][V] = {

{0, 10, 15, 0, 0, 0, 0, 0, 0, 0}, // Warehouse 1

{10, 0, 0, 12, 0, 0, 0, 0, 0, 0}, // Warehouse 2

{15, 0, 0, 0, 8, 0, 0, 0, 0, 0}, // Warehouse 3

{0, 12, 0, 0, 0, 5, 0, 0, 0, 0}, // Warehouse 4

{0, 0, 8, 0, 0, 5, 0, 3, 0, 0}, // Warehouse 5

{0, 0, 0, 0, 0, 0, 0, 0, 0, 2}, // Location 1

{0, 0, 0, 0, 0, 0, 2, 0, 0, 0}, // Location 2

{0, 0, 0, 0, 0, 0, 0, 0, 1, 0}, // Location 3

{0, 0, 0, 0, 0, 0, 0, 1, 0, 0}, // Location 4

{0, 0, 0, 0, 0, 0, 0, 0, 0, 0} // Location 5

};

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

dijkstra(graph, i);

}

return 0;

}

5.Technical Debt Reduction

#include <stdio.h>

#define TOTAL\_LINES\_OF\_CODE 1000000

#define INITIAL\_TECHNICAL\_DEBT\_PER\_LINE 0.1

#define DEBT\_REDUCTION\_RATE 0.02

int main() {

double total\_technical\_debt = TOTAL\_LINES\_OF\_CODE \* INITIAL\_TECHNICAL\_DEBT\_PER\_LINE;

int time\_periods;

printf("Initial Total Technical Debt: %.2f\n", total\_technical\_debt);

printf("Enter the number of time periods for debt reduction: ");

scanf("%d", &time\_periods);

for (int i = 1; i <= time\_periods; i++) {

total\_technical\_debt -= total\_technical\_debt \* DEBT\_REDUCTION\_RATE;

printf("After period %d, Total Technical Debt: %.2f\n", i, total\_technical\_debt);

}

return 0;

}

6.Order Fulfilment Optimization

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#define STAGES 5

void simulate\_stage(const char \*stage\_name, int delay) {

printf("Starting %s...\n", stage\_name);

sleep(delay);

printf("%s completed in %d seconds.\n", stage\_name, delay);

}

int main() {

const char \*stages[STAGES] = {

"Order Receipt",

"Inventory Allocation",

"Packaging",

"Shipping",

"Delivery"

};

int delays[STAGES] = {2, 3, 1, 4, 2};

int total\_time = 0;

printf("Simulating Order Fulfillment Process...\n");

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

simulate\_stage(stages[i], delays[i]);

total\_time += delays[i];

}

printf("Total time taken for order fulfillment: %d seconds\n", total\_time);

return 0;

}