

```

1  #include <iostream>
2  #include <vector>
3  #include <string>
4  #include <algorithm>
5  #include <stack>
6  #include <queue>
7  #include <climits>
8  using namespace std;
9
10 const int NO_PATH = -1; // Use -1 to indicate no path
11
12 // Incident structure for the first program
13 struct Inc
14 {
15     int id; // Incident ID
16     string patName; // Patient's name
17     string desc; // Description of the incident
18     int priority; // Priority of the incident
19 };
20
21 // Patient structure for the second program
22 struct Pat
23 {
24     int id; // Patient ID
25     string name; // Patient name
26     int priority; // Priority (lower number = higher priority)
27     string phone; // Patient phone number
28     string ailment; // Patient ailment
29     int aptTime; // Time of appointment (24-hour format, e.g., 900 for 9:00 AM)
30 };
31
32 // Inventory Item structure for the third program
33 struct Item
34 {
35     string name; // Name of the item
36     int quantity; // Quantity of the item
37     int priority; // Priority (lower = more critical)
38 };
39
40 // Node structure to hold hospital details
41 struct N
42 { // N: Node
43     string n; // n: name
44     vector<pair<string, int>> r; // r: resources
45 };
46
47 // Class for Union-Find data structure
48 class UF
49 { // UF: UnionFind
50 public:
51     vector<int> p, rk; // p: parent, rk: rank
52
53     UF(int sz)
54     { // sz: size
55         p.resize(sz);
56         rk.resize(sz, 0);
57         for (int i = 0; i < sz; ++i)
58         {
59             p[i] = i;
60         }
61     }
62
63     int f(int x)
64     { // f: find
65         if (p[x] != x)
66         {
67             p[x] = f(p[x]);
68         }
69         return p[x];
70     }
71
72     void u(int x, int y)
73     { // u: unite
74         int rx = f(x), ry = f(y);
75         if (rx != ry)
76         {
77             if (rk[rx] > rk[ry])
78             {

```

```

79         p[ry] = rx;
80     }
81     else if (rk[rx] < rk[ry])
82     {
83         p[rx] = ry;
84     }
85     else
86     {
87         p[ry] = rx;
88         rk[rx]++;
89     }
90 }
91 }
92 };
93
94 // Function declarations for the first program
95 vector<int> rabinSearch(const string& text, const string& pattern);
96 void searchIncByDesc(vector<Inc>& incidents);
97 void inputIncidents(vector<Inc>& incidents);
98 int partitionIncidents(vector<Inc>& incidents, int low, int high);
99 void quickSortIncidents(vector<Inc>& incidents, int low, int high);
100 void displayAllIncidents(const vector<Inc>& incidents);
101 void displayIncByPriority(const vector<Inc>& incidents);
102 void editIncDetails(vector<Inc>& incidents);
103 void removeIncFromList(vector<Inc>& incidents);
104 void showIncidentMenu();
105 void handleIncidentUserInput(vector<Inc>& incidents); // Fixed function name
106
107
108 // Function declarations for the second program
109 struct PatComp
110 {
111     bool operator() (Pat a, Pat b)
112     {
113         return a.priority > b.priority; // Min-Heap (higher priority first)
114     }
115 };
116
117 void addUrgPat(priority_queue<Pat, vector<Pat>, PatComp>& urgQueue);
118 void viewUrgPats(const priority_queue<Pat, vector<Pat>, PatComp>& urgQueue);
119 void addRegPat(queue<Pat>& regQueue);
120 void viewregPats(queue<Pat>& regQueue);
121 void serveNextUrgPat(priority_queue<Pat, vector<Pat>, PatComp>& urgQueue);
122 void serveNextRegPat(queue<Pat>& regQueue);
123 void manageAvailableSlots(vector<int>& slots);
124 void sortRegPats(queue<Pat>& regQueue);
125 void updatePatData(priority_queue<Pat, vector<Pat>, PatComp>& urgQueue);
126 void removePatAppointment(priority_queue<Pat, vector<Pat>, PatComp>& urgQueue, queue<Pat>& regQueue);
127 void displayPatientMenu(priority_queue<Pat, vector<Pat>, PatComp>& urgQueue, queue<Pat>& regQueue,
vector<int>& slots);
128
129
130 // Function declarations for the third program
131 int hashItem(const string& itemSearchName, int hashSize);
132 void addInvItem(vector<vector<Item>>& invHashTable, int hashSize);
133 void findInvItem(const vector<vector<Item>>& invHashTable, int hashSize);
134 void editInvItem(vector<vector<Item>>& invHashTable, int hashSize);
135 void removeInvItem(vector<vector<Item>>& invHashTable, int hashSize);
136 void showInvItems(const vector<vector<Item>>& invHashTable);
137 struct BSTNode
138 {
139     Item item;
140     BSTNode* left;
141     BSTNode* right;
142     BSTNode(Item it) : item(it), left(nullptr), right(nullptr) {}
143 };
144 BSTNode* addBSTNode(BSTNode* root, Item item);
145 void findBSTNode(BSTNode* root, const string& itemSearchName);
146 void listBST(BSTNode* root);
147 BSTNode* removeBSTNode(BSTNode* root, const string& itemName);
148 void clearBST(BSTNode* root);
149 void showInventoryMenu(const int hashSize, vector<vector<Item>>& invHashTable, BSTNode* bst);
150
151
152 // Function declarations for the fourth program
153 int vInt(string prompt);
154 bool vNode(int idx, int n);
155 void dijkstra(const vector<vector<int>>& g, int s, int n, vector<int>& d, vector<int>& p);

```

```

156 void printPath(const vector<int> &p, int s, int d, const vector<int>& dist);
157 void inputGraph(int &n, vector<vector<int>> &g, vector<N> &nodes);
158 void checkResources(int n, const vector<N> &nodes);
159 void modifyResources(vector<N> &nodes);
160 void addNewNode(int &n, vector<vector<int>> &g, vector<N> &nodes);
161 void findHospitalsWithResource(vector<N> &nodes, int n);
162 void displayAdjacencyMatrix(const vector<vector<int>> &g, int n);
163 void menufour(vector<vector<int>> &g, vector<N> &nodes, int n);
164
165
166 // Main menu function
167 void showMainMenu()
168 {
169     cout << "\n--- Main Menu ---\n";
170     cout << "1. Emergency Medical Incident Categorization\n";
171     cout << "2. Patient Appointment Scheduling System\n";
172     cout << "3. Medical Supply Inventory Management\n";
173     cout << "4. Healthcare Resource Allocation System\n";
174     cout << "5. Exit\n";
175     cout << "Enter your choice: ";
176 }
177
178 int main()
179 {
180     vector<Inc> incidents;
181     priority_queue<Pat, vector<Pat>, PatComp> urgQueue; // Urgent appointments (Min-Heap)
182     queue<Pat> regQueue; // Regular appointments (FIFO Queue)
183     vector<int> slots; // Available time slots for appointments
184     const int hashSize = 10; // Size of the hash table
185     vector<vector<Item>> invHashTable(hashSize); // Hash Table
186     BSTNode* bst = nullptr; // Root of Binary Search Tree
187     vector<vector<int>> g; // g: graph
188     vector<N> nodes; // N: Node
189     int n;
190
191     int choice;
192     do
193     {
194         showMainMenu();
195         cin >> choice;
196
197         switch (choice)
198         {
199             case 1:
200                 handleIncidentUserInput(incidents);
201                 break;
202             case 2:
203                 displayPatientMenu(urgQueue, regQueue, slots);
204                 break;
205             case 3:
206                 showInventoryMenu(hashSize, invHashTable, bst);
207                 break;
208             case 4:
209                 menufour(g, nodes, n);
210                 break;
211             case 5:
212                 cout << "Exiting program.\n";
213                 break;
214             default:
215                 cout << "Invalid choice! Please try again.\n";
216         }
217     }
218     while (choice != 5);
219
220     return 0;
221 }
222
223 // Implementations for the first program functions
224 vector<int> rabinSearch(const string& text, const string& pattern)
225 {
226     vector<int> positions;
227     int textLength = text.length();
228     int patternLength = pattern.length();
229
230     if (patternLength > textLength)
231     {
232         return positions;
233     }

```

```

234
235     const int d = 256;
236     const int q = 101;
237
238     int h = 1;
239     for (int i = 0; i < patternLength - 1; ++i)
240     {
241         h = (h * d) % q;
242     }
243
244     int patternHash = 0;
245     int textHash = 0;
246
247     for (int i = 0; i < patternLength; ++i)
248     {
249         patternHash = (d * patternHash + pattern[i]) % q;
250         textHash = (d * textHash + text[i]) % q;
251     }
252
253     for (int i = 0; i <= textLength - patternLength; ++i)
254     {
255         if (patternHash == textHash)
256         {
257             if (text.substr(i, patternLength) == pattern)
258             {
259                 positions.push_back(i);
260             }
261         }
262
263         if (i < textLength - patternLength)
264         {
265             textHash = (d * (textHash - text[i] * h) + text[i + patternLength]) % q;
266             if (textHash < 0)
267             {
268                 textHash = (textHash + q);
269             }
270         }
271     }
272
273     return positions;
274 }
275
276 void searchIncByDesc(vector<Inc>& incidents)
277 {
278     string query;
279     cout << "Enter part of the description to search for: ";
280     cin.ignore();
281     getline(cin, query);
282
283     bool found = false;
284
285     for (const auto& inc : incidents)
286     {
287         vector<int> positions = rabinSearch(inc.desc, query);
288         if (!positions.empty())
289         {
290             cout<< "Found in incident with priority ";
291             cout<< inc.priority;
292             cout<< ": ";
293             cout<< inc.desc << endl;
294             found = true;
295         }
296     }
297
298     if (!found)
299     {
300         cout << "No incidents found with the description containing \"";
301         cout << query;
302         cout << "\"\n";
303     }
304 }
305
306 void inputIncidents(vector<Inc>& incidents)
307 {
308     int numInc;
309     cout << "Enter number of incidents: ";
310     cin >> numInc;
311     cin.ignore();

```

```

312
313     for (int i = 0; i < numInc; ++i)
314     {
315         Inc newInc;
316         newInc.id = i + 1;
317
318         cout << "Enter patient name for incident ";
319         cout << i + 1;
320         cout << ": ";
321         getline(cin, newInc.patName);
322
323         cout << "Enter priority (number) for incident ";
324         cout << i + 1;
325         cout << ": ";
326         cin >> newInc.priority;
327         cin.ignore();
328
329         cout << "Enter description for incident ";
330         cout << i + 1;
331         cout << ": ";
332         getline(cin, newInc.desc);
333
334         incidents.push_back(newInc);
335     }
336     cout << "Incidents added successfully!\n";
337 }
338
339 int partitionIncidents(vector<Inc>& incidents, int low, int high)
340 {
341     int pivot = incidents[high].priority;
342     int i = low - 1;
343
344     for (int j = low; j < high; ++j)
345     {
346         if (incidents[j].priority <= pivot)
347         {
348             i++;
349             swap(incidents[i], incidents[j]);
350         }
351     }
352     swap(incidents[i + 1], incidents[high]);
353     return i + 1;
354 }
355
356 void quickSortIncidents(vector<Inc>& incidents, int low, int high)
357 {
358     if (low < high)
359     {
360         int pi = partitionIncidents(incidents, low, high);
361         quickSortIncidents(incidents, low, pi - 1);
362         quickSortIncidents(incidents, pi + 1, high);
363     }
364 }
365
366 void displayAllIncidents(const vector<Inc>& incidents)
367 {
368     if (incidents.empty())
369     {
370         cout << "No incidents available.\n";
371         return;
372     }
373     cout << "\n--- All Incidents ---\n";
374     cout << "Incident ID | Patient Name | Priority | Description\n";
375     for (const auto& inc : incidents)
376     {
377         cout << inc.id;
378         cout << " | ";
379         cout << inc.patName;
380         cout << " | ";
381         cout << inc.priority;
382         cout << " | ";
383         cout << inc.desc << endl;
384     }
385 }
386
387 void displayIncByPriority(const vector<Inc>& incidents)
388 {
389     int priority;

```

```

390     cout << "Enter the priority of the incident to search: ";
391     cin >> priority;
392
393     bool found = false;
394     for (const auto& inc : incidents)
395     {
396         if (inc.priority == priority)
397         {
398             cout << "Found incident: ";
399             cout << inc.id;
400             cout << " | ";
401             cout << inc.patName;
402             cout << " | ";
403             cout << inc.desc << endl;
404             found = true;
405         }
406     }
407     if (!found)
408     {
409         cout << "No incidents found with priority ";
410         cout << priority << endl;
411     }
412 }
413
414 void editIncDetails(vector<Inc>& incidents)
415 {
416     int index;
417     cout << "Enter the incident number to edit (1 to ";
418     cout << incidents.size();
419     cout << "): ";
420     cin >> index;
421     if (index < 1 || index > incidents.size())
422     {
423         cout << "Invalid incident number!" << endl;
424         return;
425     }
426     index--;
427
428     int newPriority;
429     string newDescription;
430     cout << "Enter new priority for incident ";
431     cout << index + 1;
432     cout << ": ";
433     cin >> newPriority;
434     cin.ignore();
435     cout << "Enter new description for incident ";
436     cout << index + 1;
437     cout << ": ";
438     getline(cin, newDescription);
439
440     incidents[index].priority = newPriority;
441     incidents[index].desc = newDescription;
442     cout << "Incident ";
443     cout << index + 1;
444     cout << " updated successfully!\n";
445 }
446
447 void removeIncFromList(vector<Inc>& incidents)
448 {
449     int index;
450     cout << "Enter the incident number to remove (1 to ";
451     cout << incidents.size();
452     cout << "): ";
453     cin >> index;
454     if (index < 1 || index > incidents.size())
455     {
456         cout << "Invalid incident number!" << endl;
457         return;
458     }
459     index--;
460
461     incidents.erase(incidents.begin() + index);
462     cout << "Incident ";
463     cout << index + 1;
464     cout << " removed successfully!\n";
465 }
466
467 void showIncidentMenu()

```

```

468 {
469     cout << "\n--- Emergency Medical Incident Categorization ---\n";
470     cout << "1. Input Incidents\n";
471     cout << "2. Prioritize Incidents (Quick Sort)\n";
472     cout << "3. Display All Incidents\n";
473     cout << "4. Display Specific Incident\n";
474     cout << "5. Edit Incident\n";
475     cout << "6. Remove Incident\n";
476     cout << "7. Search Incidents by Description\n";
477     cout << "8. Return to Main Menu\n";
478     cout << "Enter your choice: ";
479 }
480
481 void handleIncidentUserInput(vector<Inc>& incidents)
482 {
483     int choice;
484     do
485     {
486         showIncidentMenu();
487         cin >> choice;
488
489         switch (choice)
490         {
491             case 1:
492                 inputIncidents(incidents);
493                 break;
494             case 2:
495                 if (incidents.empty())
496                 {
497                     cout << "No incidents available to prioritize.\n";
498                 }
499                 else
500                 {
501                     quickSortIncidents(incidents, 0, incidents.size() - 1);
502                     cout << "Incidents prioritized:\n";
503                     displayAllIncidents(incidents);
504                 }
505                 break;
506             case 3:
507                 displayAllIncidents(incidents);
508                 break;
509             case 4:
510                 displayIncByPriority(incidents);
511                 break;
512             case 5:
513                 editIncDetails(incidents);
514                 break;
515             case 6:
516                 removeIncFromList(incidents);
517                 break;
518             case 7:
519                 searchIncByDesc(incidents);
520                 break;
521             case 8:
522                 cout << "Returning to main menu.\n";
523                 break;
524             default:
525                 cout << "Invalid choice! Please try again.\n";
526         }
527     }
528     while (choice != 8);
529 }
530
531 // Implementations for the second program functions
532 void addUrgPat(priority_queue<Pat, vector<Pat>, PatComp>& urgQueue)
533 {
534     Pat newPat;
535     cout << "Enter patient ID: ";
536     cin >> newPat.id;
537     cout << "Name: ";
538     cin >> newPat.name;
539     cout << "Priority: ";
540     cin >> newPat.priority;
541     cout << "Phone number: ";
542     cin >> newPat.phone;
543     cout << "Ailment: ";
544     cin >> newPat.ailment;
545     cout << "Appointment time(in 24-hour format, e.g., 900 for 9:00 AM): ";

```

```

546     cin >> newPat.aptTime;
547     urgQueue.push(newPat);
548     cout << "Patient added to urgent appointments.\n";
549 }
550
551 void viewUrgPats(const priority_queue<Pat, vector<Pat>, PatComp>& urgQueue)
552 {
553     if (urgQueue.empty())
554     {
555         cout << "No urgent appointments to display.\n";
556         return;
557     }
558
559     priority_queue<Pat, vector<Pat>, PatComp> tempQueue = urgQueue; // Create a temporary copy to
display all patients
560     cout << "\n--- Urgent Appointments ---\n";
561     while (!tempQueue.empty())
562     {
563         Pat p = tempQueue.top();
564         tempQueue.pop();
565         cout << "ID: ";
566         cout << p.id;
567         cout << ", Name: ";
568         cout << p.name;
569         cout << ", Priority: ";
570         cout << p.priority;
571         cout << ", Phone: ";
572         cout << p.phone;
573         cout << ", Ailment: ";
574         cout << p.ailment;
575         cout << ", Time: ";
576         cout << p.aptTime << endl;
577     }
578 }
579 void viewregPats(queue<Pat>& regQueue)
580 {
581     if (regQueue.empty())
582     {
583         cout << "No regular appointments to display.\n";
584         return;
585     }
586
587     queue<Pat> tempQueue = regQueue; // Create a temporary copy to display all patients
588     cout << "\n--- Regular Appointments ---\n";
589     while (!tempQueue.empty())
590     {
591         Pat p = tempQueue.front();
592         tempQueue.pop();
593         cout << "ID: ";
594         cout << p.id;
595         cout << ", Name: ";
596         cout << p.name;
597         cout << ", Time: ";
598         cout << p.aptTime << endl;
599     }
600 }
601
602 void addRegPat(queue<Pat>& regQueue)
603 {
604     Pat newPat;
605     cout << "Enter patient ID: ";
606     cin >> newPat.id;
607     cout << "Name: ";
608     cin >> newPat.name;
609     cout << "Appointment time (in 24-hour format, e.g., 900 for 9:00 AM): ";
610     cin >> newPat.aptTime;
611     newPat.priority = INT_MAX; // Default priority for regular appointments
612     regQueue.push(newPat);
613     cout << "Regular appointment added.\n";
614 }
615
616 void serveNextUrgPat(priority_queue<Pat, vector<Pat>, PatComp>& urgQueue)
617 {
618     if (urgQueue.empty())
619     {
620         cout << "No urgent appointments.\n";
621         return;
622     }

```



```

623     Pat p = urgQueue.top();
624     urgQueue.pop();
625     cout << "Serving urgent appointment:\n";
626     cout << "Patient ID ";
627     cout << p.id;
628     cout << ", Name: ";
629     cout << p.name;
630     cout << ", Priority: ";
631     cout << p.priority << endl;
632 }
633
634 void serveNextRegPat(queue<Pat>& regQueue)
635 {
636     if (regQueue.empty())
637     {
638         cout << "No regular appointments.\n";
639         return;
640     }
641     Pat p = regQueue.front();
642     regQueue.pop();
643     cout << "Serving regular appointment:\n";
644     cout << "Patient ID ";
645     cout << p.id;
646     cout << ", Name: ";
647     cout << p.name;
648     cout << ", Time: ";
649     cout << p.apptTime << endl;
650 }
651
652 void manageAvailableSlots(vector<int>& slots)
653 {
654     int n;
655     cout << "Enter the number of available time slots: ";
656     cin >> n;
657
658     slots.resize(n);
659     cout << "Enter the time slots (in 24-hour format, e.g., 900 for 9:00 AM): ";
660     for (int i = 0; i < n; ++i)
661     {
662         cin >> slots[i];
663     }
664
665     cout << "Available time slots: ";
666     for (int s : slots)
667     {
668         cout << s;
669         cout << " ";
670     }
671     cout << endl;
672 }
673
674 void sortRegPats(queue<Pat>& regQueue)
675 {
676     vector<Pat> tempPats;
677
678     while (!regQueue.empty())
679     {
680         tempPats.push_back(regQueue.front());
681         regQueue.pop();
682     }
683
684     sort(tempPats.begin(), tempPats.end(), [](const Pat& a, const Pat& b)
685     {
686         return a.apptTime < b.apptTime;
687     });
688
689     for (const Pat& p : tempPats)
690     {
691         regQueue.push(p);
692     }
693
694     cout << "Regular appointments sorted by time.\n";
695 }
696
697 void updatePatData(priority_queue<Pat, vector<Pat>, PatComp>& urgQueue)
698 {
699     int id;
700     cout << "Enter patient ID to change details: ";

```

```

701     cin >> id;
702
703     priority_queue<Pat, vector<Pat>, PatComp> tempQueue;
704     bool found = false;
705
706     while (!urgQueue.empty())
707     {
708         Pat p = urgQueue.top();
709         urgQueue.pop();
710         if (p.id == id)
711         {
712             cout << "Enter new details for patient ID " << id << "\n";
713             cout << "New name: ";
714             cin >> p.name;
715             cout << "New priority: ";
716             cin >> p.priority;
717             cout << "New phone number: ";
718             cin >> p.phone;
719             cout << "New ailment: ";
720             cin >> p.ailment;
721             found = true;
722         }
723         tempQueue.push(p);
724     }
725
726     urgQueue = tempQueue;
727
728     if (found)
729     {
730         cout << "Patient details updated.\n";
731     }
732     else
733     {
734         cout << "Patient ID not found.\n";
735     }
736 }
737
738 void removePatAppointment(priority_queue<Pat, vector<Pat>, PatComp>& urgQueue, queue<Pat>& regQueue)
739 {
740     int id;
741     char ap;
742     cout << "Urgent (U) or Regular (R): ";
743     cin >> ap;
744     cout << "Enter patient ID to remove appointment: ";
745     cin >> id;
746
747     bool found = false;
748     priority_queue<Pat, vector<Pat>, PatComp> tempQueue;
749     if (ap == 'U' || ap == 'u')
750     {
751         while (!urgQueue.empty())
752         {
753             Pat p = urgQueue.top();
754             urgQueue.pop();
755             if (p.id == id)
756             {
757                 found = true;
758                 cout << "Appointment removed: ";
759                 cout << p.name;
760                 cout << " with ID ";
761                 cout << p.id << endl;
762             }
763             else
764             {
765                 tempQueue.push(p);
766             }
767         }
768
769         urgQueue = tempQueue;
770     }
771     if (ap == 'R' || ap == 'r')
772     {
773         queue<Pat> tempRegQueue;
774         while (!regQueue.empty())
775         {
776             Pat p = regQueue.front();
777             regQueue.pop();
778             if (p.id == id)

```

```

779         {
780             found = true;
781             cout << "Regular appointment removed: ";
782             cout << p.name;
783             cout << " with ID ";
784             cout << p.id << endl;
785         }
786         else
787         {
788             tempRegQueue.push(p);
789         }
790     }
791     regQueue = tempRegQueue;
792 }
793
794 if (!found)
795 {
796     cout << "Patient ID not found in any appointment.\n";
797 }
798 }
799
800 void displayPatientMenu(priority_queue<Pat, vector<Pat>, PatComp>& urgQueue, queue<Pat>& regQueue,
vector<int>& slots)
801 {
802     int choice;
803     do
804     {
805         cout << "\n--- Patient Appointment Scheduling System ---\n";
806         cout << "1. Add Urgent Appointment\n";
807         cout << "2. Add Regular Appointment\n";
808         cout << "3. Serve Urgent Appointment\n";
809         cout << "4. Serve Regular Appointment\n";
810         cout << "5. View Urgent Appointments\n";
811         cout << "6. View Regular Appointments\n";
812         cout << "7. Manage Time Slots\n";
813         cout << "8. Sort Regular Appointments by Time\n";
814         cout << "9. Change Patient Details\n";
815         cout << "10. Remove Appointment\n";
816         cout << "11. Return to Main Menu\n";
817         cout << "Enter your choice: ";
818         cin >> choice;
819
820         switch (choice)
821         {
822             case 1:
823                 addUrgPat(urgQueue);
824                 break;
825             case 2:
826                 addRegPat(regQueue);
827                 break;
828             case 3:
829                 serveNextUrgPat(urgQueue);
830                 break;
831             case 4:
832                 serveNextRegPat(regQueue);
833                 break;
834             case 5:
835                 viewUrgPats(urgQueue);
836                 break;
837             case 6:
838                 viewregPats(regQueue);
839                 break;
840             case 7:
841                 manageAvailableSlots(slots);
842                 break;
843             case 8:
844                 sortRegPats(regQueue);
845                 break;
846             case 9:
847                 updatePatData(urgQueue);
848                 break;
849             case 10:
850                 removePatAppointment(urgQueue, regQueue);
851                 break;
852             case 11:
853                 cout << "Returning to main menu.\n";
854                 break;
855             default:

```

```

856         cout << "Invalid choice. Please try again.\n";
857     }
858 }
859 while (choice != 11);
860 }
861
862 // Implementations for the third program functions
863 int hashItem(const string& itemSearchName, int hashSize)
864 {
865     int hashValue = 0;
866     for (char c : itemSearchName)
867     {
868         hashValue = (hashValue * 31 + c) % hashSize;
869     }
870     return hashValue;
871 }
872
873 void addInvItem(vector<vector<Item>>& invHashTable, int hashSize)
874 {
875     Item newInvItem;
876     cout << "Enter item name: ";
877     cin >> newInvItem.name;
878     cout<< "Quantity: ";
879     cin >> newInvItem.quantity;
880     cout<<"Priority: ";
881     cin >> newInvItem.priority;
882
883     int hashValue = hashItem(newInvItem.name, hashSize);
884     invHashTable[hashValue].push_back(newInvItem);
885     cout << "Item '";
886     cout << newInvItem.name;
887     cout << "' added to inventory.\n";
888 }
889
890 void findInvItem(const vector<vector<Item>>& invHashTable, int hashSize)
891 {
892     string itemSearchName;
893     cout << "Enter item name to search: ";
894     cin >> itemSearchName;
895
896     int hashValue = hashItem(itemSearchName, hashSize);
897     for (const Item& item : invHashTable[hashValue])
898     {
899         if (item.name == itemSearchName)
900         {
901             cout << "Item found: Name = " << item.name
902                 << ", Quantity = " << item.quantity
903                 << ", Priority = " << item.priority << endl;
904             return;
905         }
906     }
907     cout << "Item '";
908     cout << itemSearchName;
909     cout << "' not found in inventory 1.\n";
910 }
911
912 void editInvItem(vector<vector<Item>>& invHashTable, int hashSize)
913 {
914     string itemSearchName;
915     cout << "Enter item name to update: ";
916     cin >> itemSearchName;
917
918     int hashValue = hashItem(itemSearchName, hashSize);
919     for (Item& item : invHashTable[hashValue])
920     {
921         if (item.name == itemSearchName)
922         {
923             cout << "Enter new quantity: ";
924             cin >> item.quantity;
925             cout<<"New priority: ";
926             cin >> item.priority;
927             cout<< "Item '";
928             cout<< item.name;
929             cout << "' updated.\n";
930             return;
931         }
932     }
933     cout << "Item '";

```

```

934     cout << itemSearchName;
935     cout << "' not found in inventory 1.\n";
936 }
937
938 void removeInvItem(vector<vector<Item>>& invHashTable, int hashSize)
939 {
940     string itemSearchName;
941     cout << "Enter item name to delete: ";
942     cin >> itemSearchName;
943
944     int hashValue = hashItem(itemSearchName, hashSize);
945     auto& bucket = invHashTable[hashValue];
946     for (auto it = bucket.begin(); it != bucket.end(); ++it)
947     {
948         if (it->name == itemSearchName)
949         {
950             bucket.erase(it);
951             cout << "Item ";
952             cout << itemSearchName;
953             cout << " deleted from inventory 1.\n";
954             return;
955         }
956     }
957     cout << "Item ";
958     cout << itemSearchName;
959     cout << "' not found in inventory 1.\n";
960 }
961
962 void showInvItems(const vector<vector<Item>>& invHashTable)
963 {
964     vector<Item> collectedItems;
965
966     for (const auto& bucket : invHashTable)
967     {
968         for (const Item& item : bucket)
969         {
970             collectedItems.push_back(item);
971         }
972     }
973     if (collectedItems.empty())
974     {
975         cout << "No items in inventory 1 to display.\n";
976         return;
977     }
978     cout << "Items sorted by restocking priority:\n";
979     for (const Item& item : collectedItems)
980     {
981         cout << "Name: ";
982         cout << item.name;
983         cout << ", Quantity: ";
984         cout << item.quantity;
985         cout << ", Priority: ";
986         cout << item.priority << endl;
987     }
988 }
989
990 BSTNode* addBSTNode(BSTNode* root, Item item)
991 {
992     if (!root)
993     {
994         return new BSTNode(item);
995     }
996     if (item.name < root->item.name)
997     {
998         root->left = addBSTNode(root->left, item);
999     }
1000     else if (item.name > root->item.name)
1001     {
1002         root->right = addBSTNode(root->right, item);
1003     }
1004     return root;
1005 }
1006
1007 void findBSTNode(BSTNode* root, const string& itemSearchName)
1008 {
1009     if (!root)
1010     {
1011         cout << "Item ";

```

```

1012     cout << itemSearchName;
1013     cout << "' not found in inventory 2.\n";
1014     return;
1015 }
1016
1017 if (itemSearchName < root->item.name)
1018 {
1019     findBSTNode(root->left, itemSearchName);
1020 }
1021 else if (itemSearchName > root->item.name)
1022 {
1023     findBSTNode(root->right, itemSearchName);
1024 }
1025 else
1026 {
1027     cout<< "Item found: Name = " << root->item.name
1028         << ", Quantity = " << root->item.quantity
1029         << ", Priority = " << root->item.priority << endl;
1030 }
1031 }
1032
1033 void listBST(BSTNode* root)
1034 {
1035     if (!root)
1036     {
1037         return;
1038     }
1039     listBST(root->left);
1040     cout << "Name: " << root->item.name
1041         << ", Quantity: " << root->item.quantity
1042         << ", Priority: " << root->item.priority << endl;
1043     listBST(root->right);
1044 }
1045
1046 // Delete Item from BST
1047 BSTNode* removeBSTNode(BSTNode* root, const string& itemName)
1048 {
1049     if (!root)
1050     {
1051         cout << "Item ";
1052         cout << itemName;
1053         cout << "' not found in inventory 2.\n";
1054         return root; // Item not found
1055     }
1056
1057     // Traverse the tree
1058     if (itemName < root->item.name)
1059     {
1060         root->left = removeBSTNode(root->left, itemName); // Go left
1061     }
1062     else if (itemName > root->item.name)
1063     {
1064         root->right = removeBSTNode(root->right, itemName); // Go right
1065     }
1066     else
1067     {
1068         // Node with only one child or no child
1069         if (!root->left)
1070         {
1071             BSTNode* temp = root->right;
1072             delete root; // Free memory
1073             return temp; // Return the right child
1074         }
1075         else if (!root->right)
1076         {
1077             BSTNode* temp = root->left;
1078             delete root; // Free memory
1079             return temp; // Return the left child
1080         }
1081
1082         // Node with two children: Get the inorder successor (smallest in the right subtree)
1083         BSTNode* temp = root->right;
1084         while (temp && temp->left)
1085         {
1086             temp = temp->left; // Find the leftmost node
1087         }
1088
1089         // Copy the inorder successor's content to this node

```

```

1090         root->item = temp->item;
1091
1092         // Delete the inorder successor
1093         root->right = removeBSTNode(root->right, temp->item.name);
1094     }
1095     return root; // Return the (potentially unchanged) node pointer
1096 }
1097
1098 void clearBST(BSTNode* root)
1099 {
1100     if (!root) return;
1101     clearBST(root->left);
1102     clearBST(root->right);
1103     delete root;
1104 }
1105
1106 void showInventoryMenu(const int hashSize, vector<vector<Item>>& invHashTable, BSTNode* bst)
1107 {
1108     int menuChoice;
1109     do
1110     {
1111         cout << "\n--- Medical Supply Inventory Management ---\n";
1112         cout << "1. Add Item to Inventory 1\n";
1113         cout << "2. Search Item in Inventory 1\n";
1114         cout << "3. Update Item in Inventory 1\n";
1115         cout << "4. Delete Item from Inventory 1\n";
1116         cout << "5. Display Restock Priority \n";
1117         cout << "6. Add Item to Inventory 2\n";
1118         cout << "7. Search Item in Inventory 2\n";
1119         cout << "8. delete Item from Inventory 2\n";
1120         cout << "9. Display Inventory 2\n";
1121         cout << "10. Return to Main Menu\n";
1122
1123         cout << "Enter your choice: ";
1124         cin >> menuChoice;
1125
1126         switch (menuChoice) {
1127             case 1:
1128                 addInvItem(invHashTable, hashSize);
1129                 break;
1130             case 2:
1131                 findInvItem(invHashTable, hashSize);
1132                 break;
1133             case 3:
1134                 editInvItem(invHashTable, hashSize);
1135                 break;
1136             case 4:
1137                 removeInvItem(invHashTable, hashSize);
1138                 break;
1139             case 5:
1140                 showInvItems(invHashTable);
1141                 break;
1142             case 6:
1143                 {
1144                     Item newInvItem;
1145                     cout << "Enter item name: ";
1146                     cin >> newInvItem.name;
1147                     cout << "Quantity: ";
1148                     cin >> newInvItem.quantity;
1149                     cout << "Priority: ";
1150                     cin >> newInvItem.priority;
1151                     bst = addBSTNode(bst, newInvItem);
1152                     cout << "Item ";
1153                     cout << newInvItem.name;
1154                     cout << " added to inventory 2.\n";
1155                     break;
1156                 }
1157             case 7:
1158                 {
1159                     string itemSearchName;
1160                     cout << "Enter item name to search in inventory 2: ";
1161                     cin >> itemSearchName;
1162                     findBSTNode(bst, itemSearchName);
1163                     break;
1164                 }
1165             case 8:
1166                 {
1167                     string itemSearchName;

```

```

1168         cout << "Enter item name to delete from BST: ";
1169         cin >> itemSearchName;
1170         bst = removeBSTNode(bst, itemSearchName); // Delete from BST
1171         break;
1172     }
1173     case 9:
1174         cout << "Inventory 2(sorted by name):\n";
1175         listBST(bst);
1176         break;
1177     case 10:
1178         cout << "Returning to main menu.\n";
1179         break;
1180     default:
1181         cout << "Invalid choice! Please try again.\n";
1182     }
1183 }
1184 while (menuChoice != 10);
1185 }
1186
1187 // Function to handle invalid integer input
1188 int vInt(string prompt)
1189 { // vInt: validInt
1190     int val;
1191     while (true)
1192     {
1193         cout << prompt;
1194         if (cin >> val)
1195         {
1196             cin.ignore(numeric_limits<streamsize>::max(), '\n'); // Clear input buffer
1197             return val;
1198         }
1199         else
1200         {
1201             cout << "Invalid input. Please enter a valid integer.\n";
1202             cin.clear(); // Clear error flag
1203             cin.ignore(numeric_limits<streamsize>::max(), '\n'); // Clear input buffer
1204         }
1205     }
1206 }
1207
1208 // Check if node index is valid
1209 bool vNode(int idx, int n)
1210 { // vNode: validNode
1211     return (idx >= 0 && idx < n);
1212 }
1213
1214 // Dijkstra's algorithm for shortest path
1215 void dijkstra(const vector<vector<int>> &g, int s, int n, vector<int> &d, vector<int> &p)
1216 { // g: graph, s: source, d: dist, p: prev
1217     d.assign(n, INT_MAX); // Initialize distances to INT_MAX
1218     p.assign(n, -1);
1219     priority_queue<pair<int, int>, vector<pair<int, int>>, greater<>> pq;
1220     d[s] = 0;
1221     pq.push({0, s});
1222
1223     while (!pq.empty())
1224     {
1225         int u = pq.top().second;
1226         int dist = pq.top().first;
1227         pq.pop();
1228
1229         if (dist > d[u])
1230         {
1231             continue;
1232         }
1233
1234         for (int v = 0; v < n; ++v)
1235         {
1236             if (g[u][v] > 0 && g[u][v] != INT_MAX)
1237             { // Check for valid path
1238                 int newDist = d[u] + g[u][v];
1239                 if (newDist < d[v])
1240                 {
1241                     d[v] = newDist;
1242                     p[v] = u;
1243                     pq.push({newDist, v});
1244                 }
1245             }
1246         }
1247     }
1248 }

```



```

1246     }
1247 }
1248 }
1249
1250 // Print the shortest path and distance
1251 void printPath(const vector<int> &p, int s, int d, const vector<int>& dist)
1252 { // p: prev, s: source, d: destination
1253     if (d == -1 || d == s)
1254     {
1255         cout << s << " ";
1256         return;
1257     }
1258     printPath(p, s, p[d], dist);
1259     cout << d << " ";
1260 }
1261
1262 // Input graph details
1263 void inputGraph(int &n, vector<vector<int>> &g, vector<N> &nodes)
1264 { // g: graph, N: Node
1265     //cout << "Enter number of hospitals: ";
1266     n = vInt("Enter number of hospitals: "); // Ensuring valid integer input
1267
1268     g.resize(n, vector<int>(n, 0));
1269     nodes.resize(n);
1270
1271     for (int i = 0; i < n; ++i)
1272     {
1273         cout << "Enter name of hospital ";
1274         cout << i;
1275         cout << ": ";
1276         cin >> nodes[i].n; // n: name
1277
1278         int rCount = vInt("Enter number of resources for hospital " + nodes[i].n + ": "); //
rCount: resourceCount
1279         for (int j = 0; j < rCount; ++j)
1280         {
1281             string rName; // rName: resourceName
1282             int avail; // avail: availability
1283             cout << "Enter resource name : ";
1284             cin >> rName;
1285             avail = vInt("Enter availability: ");
1286             nodes[i].r.push_back({rName, avail}); // r: resources
1287         }
1288
1289         for (int j = 0; j < i; ++j)
1290         {
1291             cout << "Enter distance between ";
1292             cout << nodes[i].n;
1293             cout << " and ";
1294             cout << nodes[j].n;
1295             cout << " (enter -1 for no path): \n";
1296             int distance = vInt("Enter distance: ");
1297             if (distance == NO_PATH)
1298             {
1299                 g[i][j] = INT_MAX;
1300             }
1301             else
1302             {
1303                 g[i][j] = distance;
1304             }
1305             g[j][i] = g[i][j]; // Ensure symmetry
1306         }
1307     }
1308 }
1309
1310 // Display resources for each hospital
1311 void checkResources(int n, const vector<N> &nodes)
1312 { // N: Node
1313     for (int i = 0; i < n; ++i)
1314     {
1315         cout << "Hospital ";
1316         cout << i;
1317         cout << " - ";
1318         cout << nodes[i].n;
1319         cout << ":\n";
1320         for (const auto &res : nodes[i].r)
1321         { // r: resources
1322             cout << "Resource: ";

```

```

1323         cout << res.first;
1324         cout << ", Availability: ";
1325         cout << res.second << endl;
1326     }
1327 }
1328 }
1329
1330 // Modify resources for a hospital
1331 void modifyResources(vector<N> &nodes)
1332 { // N: Node
1333     int idx = vInt("Enter hospital index to modify resources: ");
1334     if (!vNode(idx, nodes.size()))
1335     { // vNode: validNode
1336         cout << "Invalid hospital index.\n";
1337         return;
1338     }
1339     cout << "Modifying resources for ";
1340     cout << nodes[idx].n;
1341     cout << ":\n";
1342     int choice;
1343     do
1344     {
1345         cout << "\n1. Add new resource\n";
1346         cout << "2. Update existing resource\n";
1347         cout << "3. Remove existing resource\n";
1348         cout << "4. Go back\n";
1349         choice = vInt("Enter your choice: ");
1350         switch (choice)
1351         {
1352             case 1:
1353             {
1354                 string rName; // rName: resourceName
1355                 int avail; // avail: availability
1356                 cout << "Enter resource name: ";
1357                 cin >> rName;
1358                 avail = vInt("Enter availability: ");
1359                 nodes[idx].r.push_back({rName, avail});
1360                 cout << "Resource added successfully.\n";
1361                 break;
1362             }
1363             case 2:
1364             {
1365                 string rName; // rName: resourceName
1366                 cout << "Enter resource name to update: ";
1367                 cin >> rName;
1368                 for (auto &res : nodes[idx].r)
1369                 { // r: resources
1370                     if (res.first == rName) {
1371                         cout << "Enter new availability: ";
1372                         res.second = vInt("Enter new availability: ");
1373                         cout << "Resource updated successfully.\n";
1374                         break;
1375                     }
1376                 }
1377                 break;
1378             }
1379             case 3:
1380             {
1381                 string rName; // rName: resourceName
1382                 cout << "Enter resource name to remove: ";
1383                 cin >> rName;
1384                 nodes[idx].r.erase(remove_if(nodes[idx].r.begin(), nodes[idx].r.end(),
1385                                             [rName](const pair<string, int> &res)
1386                                             {
1387                                                 return res.first == rName;
1388                                             })),
1389                                     nodes[idx].r.end());
1390                 cout << "Resource removed successfully.\n";
1391                 break;
1392             }
1393             case 4:
1394                 cout << "Going back.\n";
1395                 break;
1396             default:
1397                 cout << "Invalid choice. Please try again.\n";
1398             }
1399         }
1400     while (choice != 4);

```

```

1401     }
1402
1403     // Add a new hospital node
1404     void addNewNode(int &n, vector<vector<int>> &g, vector<N> &nodes)
1405     {
1406         N newNode; // newNode: Node
1407         cout << "Enter name of new hospital: ";
1408         cin >> newNode.n; // n: name
1409
1410         int rCount = vInt("Enter number of resources for the new hospital: "); // rCount: resourceCount
1411         for (int i = 0; i < rCount; ++i)
1412         {
1413             string rName; // rName: resourceName
1414             int avail; // avail: availability
1415             cout << "Enter resource name: ";
1416             cin >> rName;
1417             avail = vInt("Enter availability: ");
1418             newNode.r.push_back({rName, avail}); // r: resources
1419         }
1420
1421         // Resize the graph to accommodate the new node
1422         g.resize(n + 1, vector<int>(n + 1, 0)); // Resize g to (n+1) x (n+1) and initialize with 0
1423
1424         // Set distances for the new node
1425         for (int i = 0; i < n; ++i)
1426         {
1427             cout << "Enter distance between ";
1428             cout << newNode.n;
1429             cout << " and ";
1430             cout << nodes[i].n;
1431             cout << ": ";
1432             cout << "(If no path then put -1)\n";
1433             int distance = vInt("Enter distance: ");
1434             if (distance == NO_PATH)
1435             {
1436                 g[n][i] = INT_MAX;
1437             }
1438             else
1439             {
1440                 g[n][i] = distance;
1441             }
1442             g[i][n] = g[n][i]; // Ensure symmetry
1443         }
1444
1445         n++; // Increment the number of nodes
1446         nodes.push_back(newNode); // Add the new node to the list
1447     }
1448 }
1449
1450 // Find hospitals with a given resource
1451 void findHospitalsWithResource(vector<N> &nodes, int n)
1452 {
1453     string r; // r: resource
1454     cout << "Enter resource name: ";
1455     cin >> r;
1456
1457     UF uf(n); // UF: Union-Find
1458
1459     // Group hospitals with the same resource
1460     for (int i = 0; i < n; ++i)
1461     {
1462         for (const auto &res : nodes[i].r)
1463         {
1464             if (res.first == r)
1465             {
1466                 for (int j = 0; j < n; ++j)
1467                 {
1468                     if (i != j)
1469                     {
1470                         for (const auto &otherRes : nodes[j].r)
1471                         {
1472                             if (otherRes.first == r)
1473                             {
1474                                 uf.u(i, j); // unite hospitals
1475                             }
1476                         }
1477                     }
1478                 }
1479             }
1480         }
1481     }
1482 }

```

```

1479     }
1480 }
1481 }
1482
1483 // Create a vector to store hospitals by their group
1484 vector<vector<string>> groups(n); // Each index represents a group
1485
1486 // Populate the groups with hospital names
1487 for (int i = 0; i < n; ++i)
1488 {
1489     int grp = uf.f(i); // Find the group of the hospital
1490     groups[grp].push_back(nodes[i].n); // n: name
1491 }
1492
1493 // Collect hospitals that have the specified resource
1494 vector<string> hospitalsWithResource;
1495
1496 // Check each group for hospitals with the specified resource
1497 for (const auto &group : groups)
1498 {
1499     for (const auto &hospitalName : group)
1500     {
1501         // Find the index of the hospital in nodes to check its resources
1502         for (int i = 0; i < n; ++i)
1503         {
1504             if (nodes[i].n == hospitalName)
1505             {
1506                 // Check if this hospital has the resource
1507                 for (const auto &res : nodes[i].r)
1508                 {
1509                     if (res.first == r)
1510                     {
1511                         hospitalsWithResource.push_back(hospitalName);
1512                         break; // No need to check other resources for this hospital
1513                     }
1514                 }
1515                 break; // Break after finding the hospital
1516             }
1517         }
1518     }
1519 }
1520
1521 // Display hospitals with the specified resource
1522 cout << "\nHospitals with resource ";
1523 cout << r;
1524 cout << ":\n";
1525 for (const auto &hospitalName : hospitalsWithResource)
1526 {
1527     cout << hospitalName;
1528     cout << "\n";
1529 }
1530 }
1531
1532 void displayAdjacencyMatrix(const vector<vector<int>> &g, int n)
1533 {
1534     cout << "Adjacency Matrix:\n";
1535
1536     // Print the header row with indices
1537     cout << " "; // Initial space for row header
1538     for (int i = 0; i < n; ++i)
1539     {
1540         cout << i;
1541         cout << " "; // Print each hospital index
1542     }
1543     cout << endl; // Move to the next line after the header
1544
1545     // Print each row of the adjacency matrix
1546     for (int i = 0; i < n; ++i)
1547     {
1548         cout << i;
1549         cout << " "; // Print the hospital index for the row
1550         for (int j = 0; j < n; ++j)
1551         {
1552             if (g[i][j] == INT_MAX)
1553             {
1554                 cout << "INF "; // Use "INF" to represent no path
1555             }
1556             else

```

```

1557         {
1558             cout << g[i][j];
1559             cout << " "; // Print the distance
1560         }
1561     }
1562     cout << endl; // Move to the next line after each row
1563 }
1564 }
1565
1566 void menufour(vector<vector<int>> &g, vector<N> &nodes, int n)
1567 {
1568     inputGraph(n, g, nodes); // g: graph, N: Node
1569
1570     int choice;
1571     do
1572     {
1573         cout << "\nHospital System Menu:\n";
1574         cout << "1. View all hospitals and resources\n";
1575         cout << "2. Modify resources for a hospital\n";
1576         cout << "3. Add a new hospital\n";
1577         cout << "4. Find hospitals with a specific resource\n";
1578         cout << "5. Perform shortest path search\n";
1579         cout << "6. Display adjacency matrix\n";
1580         cout << "7. Exit\n";
1581         choice = vInt("Enter your choice: "); // vInt: validInt
1582
1583         switch (choice)
1584         {
1585             case 1:
1586                 checkResources(n, nodes); // n: number of hospitals, N: Node
1587                 break;
1588             case 2:
1589                 modifyResources(nodes); // N: Node
1590                 break;
1591             case 3:
1592                 addNewNode(n, g, nodes); // g: graph, N: Node
1593                 break;
1594             case 4:
1595                 findHospitalsWithResource(nodes, n); // N: Node
1596                 break;
1597             case 5:
1598             {
1599                 int src = vInt("Enter the source hospital index for the shortest path: "); // src: source
1600                 int dest = vInt("Enter the destination hospital index for the shortest path: "); //
1601                 dest: destination
1602
1603                 if (!vNode(src, n) || !vNode(dest, n)) { // vNode: validNode
1604                     cout << "Invalid hospital indices.\n";
1605                     break;
1606                 }
1607
1608                 vector<int> d, p; // d: dist, p: prev
1609                 dijkstra(g, src, n, d, p); // g: graph, src: source, d: dist, p: prev
1610                 cout << "Shortest path from ";
1611                 cout << nodes[src].n;
1612                 cout << " to ";
1613                 cout << nodes[dest].n;
1614                 cout << ":\n"; // n: name
1615                 printPath(p, src, dest, d); // p: prev, src: source, dest: destination, d: dist
1616                 cout << "\nDistance: ";
1617                 cout << d[dest];
1618                 cout << endl; // d: dist
1619                 break;
1620             }
1621             case 6:
1622                 displayAdjacencyMatrix(g, n);
1623                 break;
1624             case 7:
1625                 cout << "Exiting program.\n";
1626                 break;
1627             default:
1628                 cout << "Invalid choice. Please try again.\n";
1629         }
1630     }
1631     while (choice != 7);
1632 }

```