

Kode 80%

Kelompok 15 , M.GHYMVANDYAR EL M. 103092400087 & LAURA DJASMIN TANDIawan

RUTE H DAN CPP

The screenshot shows a code editor window with multiple tabs at the top: R.cpp, R.h, Rute.h (which is the active tab), Rute.cpp, and Halte.cpp. The Rute.h tab is highlighted with a blue border. The code itself is a header file for a route list. It includes standard headers like iostream and string, defines a namespace std, and uses the R.h header. The file contains two main structures: Rute and ListRute. The Rute structure is a linked list node with fields for idRute, namaRute, a pointer to the firstRelasi node, and a next pointer to the next node. The ListRute structure is a pointer to the first node of the list. The file also defines several functions: createListRute, insertRute, findRute, deleteRute, showParentNoChild, and showAllRoute. The code concludes with an #endif directive to close the preprocessor macro definition.

```
1 ifndef RUTE_H_INCLUDED
2 define RUTE_H_INCLUDED
3
4 include <iostream>
5 include <string>
6 include "R.h"
7 using namespace std;
8
9 struct Relasi;
10
11 struct Rute;
12 typedef Rute* adrRute;
13 typedef Relasi* adrRelasi;
14
15 struct Rute{
16     int idRute;
17     string namaRute;
18     adrRelasi firstRelasi;
19     adrRute next;
20 };
21
22 struct ListRute{
23     adrRute first;
24 };
25
26
27
28
29 void createListRute(ListRute &Lr);
30 adrRute newRute(int id, string nama);
31 void insertRute(ListRute &Lr, adrRute r);
32 adrRute findRute(ListRute Lr, int id);
33 void deleteRute(ListRute &Lr, int id);
34
35
36 //tambahan responsi80
37 void showParentNoChild(ListRute Lr);
38 void showAllRute(ListRute Lr); // showAllParent
39
40
41
42
43
44
45 endif // RUTE_H_INCLUDED
46
```

HALTE H DAN CPP

R.cpp X R.h X Rute.h X *Rute.cpp X Halte.cpp X Halte.h X main.cpp X

D:\Desktop\vandy bahasa c\TubesStrukdat02\TubesStrukdat\R.cpp

Project: TubesStrukdat.

```
4 #include <iostream>
5 #include <string>
6 using namespace std;
7
8
9 // struct deklarasi
10 struct Halte;
11 typedef Halte* adrHalte;
12
13
14 struct Halte{
15     string namaHalte;
16     adrHalte next;
17
18 };
19
20 struct ListHalte {
21     adrHalte first;
22
23 };
24
25
26
27 void createListHalte(ListHalte &Lh); // list kosong
28 adrHalte newHalte(string nama); // node halte baru
29 void insertHalte(ListHalte &Lh, adrHalte p); //
30 adrHalte findHalte(ListHalte &Lh, string nama); // serachir halte(child) be
31 void deleteHalte(ListHalte &Lh, ListRute &Lr, string nama); //hapus (by sea
32
33 //tambahan responsi80
34 void showAllHalte(ListHalte &Lh); //showAll halte
35
36
37
38
39
40
41
42 #endif // HALTE_H_INCLUDED
43
```

```

R.cpp X R.h X Rute.h

1 #include <iostream>
2 #include <vector>
3 #include <stack>
4 #include <queue>
5 #include <limits.h>
6
7 using namespace std;
8
9
10 void dijkstra(int s, int t) {
11     int n = max(s, t);
12     vector<int> dist(n, INT_MAX);
13     vector<int> pred(n, -1);
14     vector<int> vis(n, 0);
15
16     dist[s] = 0;
17     pred[s] = s;
18
19     priority_queue<pair<int, int>, vector<pair<int, int>>, greater<pair<int, int>> pq;
20     pq.push({0, s});
21
22     while (!pq.empty()) {
23         auto [d, u] = pq.top();
24         pq.pop();
25
26         if (vis[u]) continue;
27
28         vis[u] = 1;
29
30         for (auto v : adj[u]) {
31             if (dist[v] == INT_MAX) continue;
32
33             if (dist[v] >= dist[u] + weight[u][v]) {
34                 dist[v] = dist[u] + weight[u][v];
35                 pred[v] = u;
36
37                 if (v == t) break;
38             }
39         }
40     }
41
42     cout << "Distances: " << endl;
43     for (int i = 0; i < n; ++i) cout << i << ": " << dist[i] << endl;
44
45     cout << "Path: " << endl;
46     vector<int> path;
47     int cur = t;
48
49     while (pred[cur] != -1) {
50         path.push_back(cur);
51         cur = pred[cur];
52     }
53
54     reverse(path.begin(), path.end());
55     for (int v : path) cout << v << " ";
56
57     cout << endl;
58 }
59
60
61 void dijkstra(int s, int t, vector<int> &dist) {
62     int n = max(s, t);
63     vector<int> pred(n, -1);
64
65     dist[s] = 0;
66     pred[s] = s;
67
68     priority_queue<pair<int, int>, vector<pair<int, int>>, greater<pair<int, int>> pq;
69     pq.push({0, s});
70
71     while (!pq.empty()) {
72         auto [d, u] = pq.top();
73         pq.pop();
74
75         if (dist[u] < d) continue;
76
77         for (auto v : adj[u]) {
78             if (dist[v] == INT_MAX) continue;
79
80             if (dist[v] >= dist[u] + weight[u][v]) {
81                 dist[v] = dist[u] + weight[u][v];
82                 pred[v] = u;
83
84                 if (v == t) break;
85             }
86         }
87     }
88
89     cout << "Distances: " << endl;
90     for (int i = 0; i < n; ++i) cout << i << ": " << dist[i] << endl;
91
92     cout << "Path: " << endl;
93     vector<int> path;
94     int cur = t;
95
96     while (pred[cur] != -1) {
97         path.push_back(cur);
98         cur = pred[cur];
99     }
100
101    reverse(path.begin(), path.end());
102    for (int v : path) cout << v << " ";
103
104    cout << endl;
105 }
106
107
108 void dijkstra(int s, int t, vector<int> &dist) {
109     int n = max(s, t);
110     vector<int> pred(n, -1);
111
112     dist[s] = 0;
113     pred[s] = s;
114
115     priority_queue<pair<int, int>, vector<pair<int, int>>, greater<pair<int, int>> pq;
116     pq.push({0, s});
117
118     while (!pq.empty()) {
119         auto [d, u] = pq.top();
120         pq.pop();
121
122         if (dist[u] < d) continue;
123
124         for (auto v : adj[u]) {
125             if (dist[v] == INT_MAX) continue;
126
127             if (dist[v] >= dist[u] + weight[u][v]) {
128                 dist[v] = dist[u] + weight[u][v];
129                 pred[v] = u;
130
131                 if (v == t) break;
132             }
133         }
134     }
135
136     cout << "Distances: " << endl;
137     for (int i = 0; i < n; ++i) cout << i << ": " << dist[i] << endl;
138
139     cout << "Path: " << endl;
140     vector<int> path;
141     int cur = t;
142
143     while (pred[cur] != -1) {
144         path.push_back(cur);
145         cur = pred[cur];
146     }
147
148    reverse(path.begin(), path.end());
149    for (int v : path) cout << v << " ";
150
151    cout << endl;
152 }
153
154
155 void dijkstra(int s, int t, vector<int> &dist) {
156     int n = max(s, t);
157     vector<int> pred(n, -1);
158
159     dist[s] = 0;
160     pred[s] = s;
161
162     priority_queue<pair<int, int>, vector<pair<int, int>>, greater<pair<int, int>> pq;
163     pq.push({0, s});
164
165     while (!pq.empty()) {
166         auto [d, u] = pq.top();
167         pq.pop();
168
169         if (dist[u] < d) continue;
170
171         for (auto v : adj[u]) {
172             if (dist[v] == INT_MAX) continue;
173
174             if (dist[v] >= dist[u] + weight[u][v]) {
175                 dist[v] = dist[u] + weight[u][v];
176                 pred[v] = u;
177
178                 if (v == t) break;
179             }
180         }
181     }
182
183     cout << "Distances: " << endl;
184     for (int i = 0; i < n; ++i) cout << i << ": " << dist[i] << endl;
185
186     cout << "Path: " << endl;
187     vector<int> path;
188     int cur = t;
189
190     while (pred[cur] != -1) {
191         path.push_back(cur);
192         cur = pred[cur];
193     }
194
195    reverse(path.begin(), path.end());
196    for (int v : path) cout << v << " ";
197
198    cout << endl;
199 }
200
201
202 void dijkstra(int s, int t, vector<int> &dist) {
203     int n = max(s, t);
204     vector<int> pred(n, -1);
205
206     dist[s] = 0;
207     pred[s] = s;
208
209     priority_queue<pair<int, int>, vector<pair<int, int>>, greater<pair<int, int>> pq;
210     pq.push({0, s});
211
212     while (!pq.empty()) {
213         auto [d, u] = pq.top();
214         pq.pop();
215
216         if (dist[u] < d) continue;
217
218         for (auto v : adj[u]) {
219             if (dist[v] == INT_MAX) continue;
220
221             if (dist[v] >= dist[u] + weight[u][v]) {
222                 dist[v] = dist[u] + weight[u][v];
223                 pred[v] = u;
224
225                 if (v == t) break;
226             }
227         }
228     }
229
230     cout << "Distances: " << endl;
231     for (int i = 0; i < n; ++i) cout << i << ": " << dist[i] << endl;
232
233     cout << "Path: " << endl;
234     vector<int> path;
235     int cur = t;
236
237     while (pred[cur] != -1) {
238         path.push_back(cur);
239         cur = pred[cur];
240     }
241
242    reverse(path.begin(), path.end());
243    for (int v : path) cout << v << " ";
244
245    cout << endl;
246 }
247
248
249 void dijkstra(int s, int t, vector<int> &dist) {
250     int n = max(s, t);
251     vector<int> pred(n, -1);
252
253     dist[s] = 0;
254     pred[s] = s;
255
256     priority_queue<pair<int, int>, vector<pair<int, int>>, greater<pair<int, int>> pq;
257     pq.push({0, s});
258
259     while (!pq.empty()) {
260         auto [d, u] = pq.top();
261         pq.pop();
262
263         if (dist[u] < d) continue;
264
265         for (auto v : adj[u]) {
266             if (dist[v] == INT_MAX) continue;
267
268             if (dist[v] >= dist[u] + weight[u][v]) {
269                 dist[v] = dist[u] + weight[u][v];
270                 pred[v] = u;
271
272                 if (v == t) break;
273             }
274         }
275     }
276
277     cout << "Distances: " << endl;
278     for (int i = 0; i < n; ++i) cout << i << ": " << dist[i] << endl;
279
280     cout << "Path: " << endl;
281     vector<int> path;
282     int cur = t;
283
284     while (pred[cur] != -1) {
285         path.push_back(cur);
286         cur = pred[cur];
287     }
288
289    reverse(path.begin(), path.end());
290    for (int v : path) cout << v << " ";
291
292    cout << endl;
293 }
294
295
296 void dijkstra(int s, int t, vector<int> &dist) {
297     int n = max(s, t);
298     vector<int> pred(n, -1);
299
300     dist[s] = 0;
301     pred[s] = s;
302
303     priority_queue<pair<int, int>, vector<pair<int, int>>, greater<pair<int, int>> pq;
304     pq.push({0, s});
305
306     while (!pq.empty()) {
307         auto [d, u] = pq.top();
308         pq.pop();
309
310         if (dist[u] < d) continue;
311
312         for (auto v : adj[u]) {
313             if (dist[v] == INT_MAX) continue;
314
315             if (dist[v] >= dist[u] + weight[u][v]) {
316                 dist[v] = dist[u] + weight[u][v];
317                 pred[v] = u;
318
319                 if (v == t) break;
320             }
321         }
322     }
323
324     cout << "Distances: " << endl;
325     for (int i = 0; i < n; ++i) cout << i << ": " << dist[i] << endl;
326
327     cout << "Path: " << endl;
328     vector<int> path;
329     int cur = t;
330
331     while (pred[cur] != -1) {
332         path.push_back(cur);
333         cur = pred[cur];
334     }
335
336    reverse(path.begin(), path.end());
337    for (int v : path) cout << v << " ";
338
339    cout << endl;
340 }
341
342
343 void dijkstra(int s, int t, vector<int> &dist) {
344     int n = max(s, t);
345     vector<int> pred(n, -1);
346
347     dist[s] = 0;
348     pred[s] = s;
349
350     priority_queue<pair<int, int>, vector<pair<int, int>>, greater<pair<int, int>> pq;
351     pq.push({0, s});
352
353     while (!pq.empty()) {
354         auto [d, u] = pq.top();
355         pq.pop();
356
357         if (dist[u] < d) continue;
358
359         for (auto v : adj[u]) {
360             if (dist[v] == INT_MAX) continue;
361
362             if (dist[v] >= dist[u] + weight[u][v]) {
363                 dist[v] = dist[u] + weight[u][v];
364                 pred[v] = u;
365
366                 if (v == t) break;
367             }
368         }
369     }
370
371     cout << "Distances: " << endl;
372     for (int i = 0; i < n; ++i) cout << i << ": " << dist[i] << endl;
373
374     cout << "Path: " << endl;
375     vector<int> path;
376     int cur = t;
377
378     while (pred[cur] != -1) {
379         path.push_back(cur);
380         cur = pred[cur];
381     }
382
383    reverse(path.begin(), path.end());
384    for (int v : path) cout << v << " ";
385
386    cout << endl;
387 }
388
389
390 void dijkstra(int s, int t, vector<int> &dist) {
391     int n = max(s, t);
392     vector<int> pred(n, -1);
393
394     dist[s] = 0;
395     pred[s] = s;
396
397     priority_queue<pair<int, int>, vector<pair<int, int>>, greater<pair<int, int>> pq;
398     pq.push({0, s});
399
400     while (!pq.empty()) {
401         auto [d, u] = pq.top();
402         pq.pop();
403
404         if (dist[u] < d) continue;
405
406         for (auto v : adj[u]) {
407             if (dist[v] == INT_MAX) continue;
408
409             if (dist[v] >= dist[u] + weight[u][v]) {
410                 dist[v] = dist[u] + weight[u][v];
411                 pred[v] = u;
412
413                 if (v == t) break;
414             }
415         }
416     }
417
418     cout << "Distances: " << endl;
419     for (int i = 0; i < n; ++i) cout << i << ": " << dist[i] << endl;
420
421     cout << "Path: " << endl;
422     vector<int> path;
423     int cur = t;
424
425     while (pred[cur] != -1) {
426         path.push_back(cur);
427         cur = pred[cur];
428     }
429
430    reverse(path.begin(), path.end());
431    for (int v : path) cout << v << " ";
432
433    cout << endl;
434 }
435
436
437 void dijkstra(int s, int t, vector<int> &dist) {
438     int n = max(s, t);
439     vector<int> pred(n, -1);
440
441     dist[s] = 0;
442     pred[s] = s;
443
444     priority_queue<pair<int, int>, vector<pair<int, int>>, greater<pair<int, int>> pq;
445     pq.push({0, s});
446
447     while (!pq.empty()) {
448         auto [d, u] = pq.top();
449         pq.pop();
450
451         if (dist[u] < d) continue;
452
453         for (auto v : adj[u]) {
454             if (dist[v] == INT_MAX) continue;
455
456             if (dist[v] >= dist[u] + weight[u][v]) {
457                 dist[v] = dist[u] + weight[u][v];
458                 pred[v] = u;
459
460                 if (v == t) break;
461             }
462         }
463     }
464
465     cout << "Distances: " << endl;
466     for (int i = 0; i < n; ++i) cout << i << ": " << dist[i] << endl;
467
468     cout << "Path: " << endl;
469     vector<int> path;
470     int cur = t;
471
472     while (pred[cur] != -1) {
473         path.push_back(cur);
474         cur = pred[cur];
475     }
476
477    reverse(path.begin(), path.end());
478    for (int v : path) cout << v << " ";
479
480    cout << endl;
481 }
482
483
484 void dijkstra(int s, int t, vector<int> &dist) {
485     int n = max(s, t);
486     vector<int> pred(n, -1);
487
488     dist[s] = 0;
489     pred[s] = s;
490
491     priority_queue<pair<int, int>, vector<pair<int, int>>, greater<pair<int, int>> pq;
492     pq.push({0, s});
493
494     while (!pq.empty()) {
495         auto [d, u] = pq.top();
496         pq.pop();
497
498         if (dist[u] < d) continue;
499
500         for (auto v : adj[u]) {
501             if (dist[v] == INT_MAX) continue;
502
503             if (dist[v] >= dist[u] + weight[u][v]) {
504                 dist[v] = dist[u] + weight[u][v];
505                 pred[v] = u;
506
507                 if (v == t) break;
508             }
509         }
510     }
511
512     cout << "Distances: " << endl;
513     for (int i = 0; i < n; ++i) cout << i << ": " << dist[i] << endl;
514
515     cout << "Path: " << endl;
516     vector<int> path;
517     int cur = t;
518
519     while (pred[cur] != -1) {
520         path.push_back(cur);
521         cur = pred[cur];
522     }
523
524    reverse(path.begin(), path.end());
525    for (int v : path) cout << v << " ";
526
527    cout << endl;
528 }
529
530
531 void dijkstra(int s, int t, vector<int> &dist) {
532     int n = max(s, t);
533     vector<int> pred(n, -1);
534
535     dist[s] = 0;
536     pred[s] = s;
537
538     priority_queue<pair<int, int>, vector<pair<int, int>>, greater<pair<int, int>> pq;
539     pq.push({0, s});
540
541     while (!pq.empty()) {
542         auto [d, u] = pq.top();
543         pq.pop();
544
545         if (dist[u] < d) continue;
546
547         for (auto v : adj[u]) {
548             if (dist[v] == INT_MAX) continue;
549
550             if (dist[v] >= dist[u] + weight[u][v]) {
551                 dist[v] = dist[u] + weight[u][v];
552                 pred[v] = u;
553
554                 if (v == t) break;
555             }
556         }
557     }
558
559     cout << "Distances: " << endl;
560     for (int i = 0; i < n; ++i) cout << i << ": " << dist[i] << endl;
561
562     cout << "Path: " << endl;
563     vector<int> path;
564     int cur = t;
565
566     while (pred[cur] != -1) {
567         path.push_back(cur);
568         cur = pred[cur];
569     }
570
571    reverse(path.begin(), path.end());
572    for (int v : path) cout << v << " ";
573
574    cout << endl;
575 }
576
577
578 void dijkstra(int s, int t, vector<int> &dist) {
579     int n = max(s, t);
580     vector<int> pred(n, -1);
581
582     dist[s] = 0;
583     pred[s] = s;
584
585     priority_queue<pair<int, int>, vector<pair<int, int>>, greater<pair<int, int>> pq;
586     pq.push({0, s});
587
588     while (!pq.empty()) {
589         auto [d, u] = pq.top();
590         pq.pop();
591
592         if (dist[u] < d) continue;
593
594         for (auto v : adj[u]) {
595             if (dist[v] == INT_MAX) continue;
596
597             if (dist[v] >= dist[u] + weight[u][v]) {
598                 dist[v] = dist[u] + weight[u][v];
599                 pred[v] = u;
600
601                 if (v == t) break;
602             }
603         }
604     }
605
606     cout << "Distances: " << endl;
607     for (int i = 0; i < n; ++i) cout << i << ": " << dist[i] << endl;
608
609     cout << "Path: " << endl;
610     vector<int> path;
611     int cur = t;
612
613     while (pred[cur] != -1) {
614         path.push_back(cur);
615         cur = pred[cur];
616     }
617
618    reverse(path.begin(), path.end());
619    for (int v : path) cout << v << " ";
620
621    cout << endl;
622 }
623
624
625 void dijkstra(int s, int t, vector<int> &dist) {
626     int n = max(s, t);
627     vector<int> pred(n, -1);
628
629     dist[s] = 0;
630     pred[s] = s;
631
632     priority_queue<pair<int, int>, vector<pair<int, int>>, greater<pair<int, int>> pq;
633     pq.push({0, s});
634
635     while (!pq.empty()) {
636         auto [d, u] = pq.top();
637         pq.pop();
638
639         if (dist[u] < d) continue;
640
641         for (auto v : adj[u]) {
642             if (dist[v] == INT_MAX) continue;
643
644             if (dist[v] >= dist[u] + weight[u][v]) {
645                 dist[v] = dist[u] + weight[u][v];
646                 pred[v] = u;
647
648                 if (v == t) break;
649             }
650         }
651     }
652
653     cout << "Distances: " << endl;
654     for (int i = 0; i < n; ++i) cout << i << ": " << dist[i] << endl;
655
656     cout << "Path: " << endl;
657     vector<int> path;
658     int cur = t;
659
660     while (pred[cur] != -1) {
661         path.push_back(cur);
662         cur = pred[cur];
663     }
664
665    reverse(path.begin(), path.end());
666    for (int v : path) cout << v << " ";
667
668    cout << endl;
669 }
670
671
672 void dijkstra(int s, int t, vector<int> &dist) {
673     int n = max(s, t);
674     vector<int> pred(n, -1);
675
676     dist[s] = 0;
677     pred[s] = s;
678
679     priority_queue<pair<int, int>, vector<pair<int, int>>, greater<pair<int, int>> pq;
680     pq.push({0, s});
681
682     while (!pq.empty()) {
683         auto [d, u] = pq.top();
684         pq.pop();
685
686         if (dist[u] < d) continue;
687
688         for (auto v : adj[u]) {
689             if (dist[v] == INT_MAX) continue;
690
691             if (dist[v] >= dist[u] + weight[u][v]) {
692                 dist[v] = dist[u] + weight[u][v];
693                 pred[v] = u;
694
695                 if (v == t) break;
696             }
697         }
698     }
699
700     cout << "Distances: " << endl;
701     for (int i = 0; i < n; ++i) cout << i << ": " << dist[i] << endl;
702
703     cout << "Path: " << endl;
704     vector<int> path;
705     int cur = t;
706
707     while (pred[cur] != -1) {
708         path.push_back(cur);
709         cur = pred[cur];
710     }
711
712    reverse(path.begin(), path.end());
713    for (int v : path) cout << v << " ";
714
715    cout << endl;
716 }
717
718
719 void dijkstra(int s, int t, vector<int> &dist) {
720     int n = max(s, t);
721     vector<int> pred(n, -1);
722
723     dist[s] = 0;
724     pred[s] = s;
725
726     priority_queue<pair<int, int>, vector<pair<int, int>>, greater<pair<int, int>> pq;
727     pq.push({0, s});
728
729     while (!pq.empty()) {
730         auto [d, u] = pq.top();
731         pq.pop();
732
733         if (dist[u] < d) continue;
734
735         for (auto v : adj[u]) {
736             if (dist[v] == INT_MAX) continue;
737
738             if (dist[v] >= dist[u] + weight[u][v]) {
739                 dist[v] = dist[u] + weight[u][v];
740                 pred[v] = u;
741
742                 if (v == t) break;
743             }
744         }
745     }
746
747     cout << "Distances: " << endl;
748     for (int i = 0; i < n; ++i) cout << i << ": " << dist[i] << endl;
749
750     cout << "Path: " << endl;
751     vector<int> path;
752     int cur = t;
753
754     while (pred[cur] != -1) {
755         path.push_back(cur);
756         cur = pred[cur];
757     }
758
759    reverse(path.begin(), path.end());
760    for (int v : path) cout << v << " ";
761
762    cout << endl;
763 }
764
765
766 void dijkstra(int s, int t, vector<int> &dist) {
767     int n = max(s, t);
768     vector<int> pred(n, -1);
769
770     dist[s] = 0;
771     pred[s] = s;
772
773     priority_queue<pair<int, int>, vector<pair<int, int>>, greater<pair<int, int>> pq;
774     pq.push({0, s});
775
776     while (!pq.empty()) {
777         auto [d, u] = pq.top();
778         pq.pop();
779
780         if (dist[u] < d) continue;
781
782         for (auto v : adj[u]) {
783             if (dist[v] == INT_MAX) continue;
784
785             if (dist[v] >= dist[u] + weight[u][v]) {
786                 dist[v] = dist[u] + weight[u][v];
787                 pred[v] = u;
788
789                 if (v == t) break;
790             }
791         }
792     }
793
794     cout << "Distances: " << endl;
795     for (int i = 0; i < n; ++i) cout << i << ": " << dist[i] << endl;
796
797     cout << "Path: " << endl;
798     vector<int> path;
799     int cur = t;
800
801     while (pred[cur] != -1) {
802         path.push_back(cur);
803         cur = pred[cur];
804     }
805
806    reverse(path.begin(), path.end());
807    for (int v : path) cout << v << " ";
808
809    cout << endl;
810 }
811
812
813 void dijkstra(int s, int t, vector<int> &dist) {
814     int n = max(s, t);
815     vector<int> pred(n, -1);
816
817     dist[s] = 0;
818     pred[s] = s;
819
820     priority_queue<pair<int, int>, vector<pair<int, int>>, greater<pair<int, int>> pq;
821     pq.push({0, s});
822
823     while (!pq.empty()) {
824         auto [d, u] = pq.top();
825         pq.pop();
826
827         if (dist[u] < d) continue;
828
829         for (auto v : adj[u]) {
830             if (dist[v] == INT_MAX) continue;
831
832             if (dist[v] >= dist[u] + weight[u][v]) {
833                 dist[v] = dist[u] + weight[u][v];
834                 pred[v] = u;
835
836                 if (v == t) break;
837             }
838         }
839     }
840
841     cout << "Distances: " << endl;
842     for (int i = 0; i < n; ++i) cout << i << ": " << dist[i] << endl;
843
844     cout << "Path: " << endl;
845     vector<int> path;
846     int cur = t;
847
848     while (pred[cur] != -1) {
849         path.push_back(cur);
850         cur = pred[cur];
851     }
852
853    reverse(path.begin(), path.end());
854    for (int v : path) cout << v << " ";
855
856    cout << endl;
857 }
858
859
860 void dijkstra(int s, int t, vector<int> &dist) {
861     int n = max(s, t);
862     vector<int> pred(n, -1);
863
864     dist[s] = 0;
865     pred[s] = s;
866
867     priority_queue<pair<int, int>, vector<pair<int, int>>, greater<pair<int, int>> pq;
868     pq.push({0, s});
869
870     while (!pq.empty()) {
871         auto [d, u] = pq.top();
872         pq.pop();
873
874         if (dist[u] < d) continue;
875
876         for (auto v : adj[u]) {
877             if (dist[v] == INT_MAX) continue;
878
879             if (dist[v] >= dist[u] + weight[u][v]) {
880                 dist[v] = dist[u] + weight[u][v];
881                 pred[v] = u;
882
883                 if (v == t) break;
884             }
885         }
886     }
887
888     cout << "Distances: " << endl;
889     for (int i = 0; i < n; ++i) cout << i << ": " << dist[i] << endl;
890
891     cout << "Path: " << endl;
892     vector<int> path;
893     int cur = t;
894
895     while (pred[cur] != -1) {
896         path.push_back(cur);
897         cur = pred[cur];
898     }
899
900    reverse(path.begin(), path.end());
901    for (int v : path) cout << v << " ";
902
903    cout << endl;
904 }
905
906
907 void dijkstra(int s, int t, vector<int> &dist) {
908     int n = max(s, t);
909     vector<int> pred(n, -1);
910
911     dist[s] = 0;
912     pred[s] = s;
913
914     priority_queue<pair<int, int>, vector<pair<int, int>>, greater<pair<int, int>> pq;
915     pq.push({0, s});
916
917     while (!pq.empty()) {
918         auto [d, u] = pq.top();
919         pq.pop();
920
921         if (dist[u] < d) continue;
922
923         for (auto v : adj[u]) {
924             if (dist[v] == INT_MAX) continue;
925
926             if (dist[v] >= dist[u] + weight[u][v]) {
927                 dist[v] = dist[u] + weight[u][v];
928                 pred[v] = u;
929
930                 if (v == t) break;
931             }
932         }
933     }
934
935     cout << "Distances: " << endl;
936     for (int i = 0; i < n; ++i) cout << i << ": " << dist[i] << endl;
937
938     cout << "Path: " << endl;
939     vector<int> path;
940     int cur = t;
941
942     while (pred[cur] != -1) {
943         path.push_back(cur);
944         cur = pred[cur];
945     }
946
947    reverse(path.begin(), path.end());
948    for (int v : path) cout << v << " ";
949
950    cout << endl;
951 }
952
953
954 void dijkstra(int s, int t, vector<int> &dist) {
955     int n = max(s, t);
956     vector<int> pred(n, -1);
957
958     dist[s] = 0;
959     pred[s] = s;
960
961     priority_queue<pair<int, int>, vector<pair<int, int>>, greater<pair<int, int>> pq;
962     pq.push({0, s});
963
964     while (!pq.empty()) {
965         auto [d, u] = pq.top();
966         pq.pop();
967
968         if (dist[u] < d) continue;
969
970         for (auto v : adj[u]) {
971             if (dist[v] == INT_MAX) continue;
972
973             if (dist[v] >= dist[u] + weight[u][v]) {
974                 dist[v] = dist[u] + weight[u][v];
975                 pred[v] = u;
976
977                 if (v == t) break;
978             }
979         }
980     }
981
982     cout << "Distances: " << endl;
983     for (int i = 0; i < n; ++i) cout << i << ": " << dist[i] << endl;
984
985     cout << "Path: " << endl;
986     vector<int> path;
987     int cur = t;
988
989     while (pred[cur] != -1) {
990         path.push_back(cur);
991         cur = pred[cur];
992     }
993
994    reverse(path.begin(), path.end());
995    for (int v : path) cout << v << " ";
996
997    cout << endl;
998 }
999
1000
1001 void dijkstra(int s, int t, vector<int> &dist) {
1002     int n = max(s, t);
1003     vector<int> pred(n, -1);
1004
1005     dist[s] = 0;
1006     pred[s] = s;
1007
1008     priority_queue<pair<int, int>, vector<pair<int, int>>, greater<pair<int, int>> pq;
1009     pq.push({0, s});
1010
1011     while (!pq.empty()) {
1012         auto [d, u] = pq.top();
1013         pq.pop();
1014
1015         if (dist[u] < d) continue;
1016
1017         for (auto v : adj[u]) {
1018             if (dist[v] == INT_MAX) continue;
1019
1020             if (dist[v] >= dist[u] + weight[u][v]) {
1021                 dist[v] = dist[u] + weight[u][v];
1022                 pred[v] = u;
1023
1024                 if (v == t) break;
1025             }
1026         }
1027     }
1028
1029     cout << "Distances: " << endl;
1030     for (int i = 0; i < n; ++i) cout << i << ": " << dist[i] << endl;
1031
1032     cout << "Path: " << endl;
1033     vector<int> path;
1034     int cur = t;
1035
1036     while (pred[cur] != -1) {
1037         path.push_back(cur);
1038         cur = pred[cur];
1039     }
1040
1041    reverse(path.begin(), path.end());
1042    for (int v : path) cout << v << " ";
1043
1044    cout << endl;
1045 }
1046
1047
1048 void dijkstra(int s, int t, vector<int> &dist) {
1049     int n = max(s, t);
1050     vector<int> pred(n, -1);
1051
1052     dist[s] = 0;
1053     pred[s] = s;
1054
1055     priority_queue<pair<int, int>, vector<pair<int, int>>, greater<pair<int, int>> pq;
1056     pq.push({0, s});
1057
1058     while (!pq.empty()) {
1059         auto [d, u] = pq.top();
1060         pq.pop();
1061
1062         if (dist[u] < d) continue;
1063
1064         for (auto v : adj[u]) {
1065             if (dist[v] == INT_MAX) continue;
1066
1067             if (dist[v] >= dist[u] + weight[u][v]) {
1068                 dist[v] = dist[u] + weight[u][v];
1069                 pred[v] = u;
1070
1071                 if (v == t) break;
1072             }
1073         }
1074     }
1075
1076     cout << "Distances: " << endl;
1077     for (int i = 0; i < n; ++i) cout << i << ": " << dist[i] << endl;
1078

```

RELASI H DAN CPP

[TubesStrukdat.] - Code::Blocks 25.03

Edit View Search Project Build Debug Fortran wxSmith To
ocks Settings Help

Projects x
ects Files ►
cspace
TubesStrukdat.
Sources
Headers

```
R.cpp x R.h x Rute.h x *Rute.cpp x Halte.cpp x
1 #ifndef _A_A__HALTE_H_
2 #define _A_A__HALTE_H_
3
4 #include <iostream>
5 #include <string>
6
7 using namespace std;
8
9
10 #include "halte.h"
11 #include "rute.h"
12
13
14 struct halte;
15 typedef halte* adminalasi;
16
17
18 struct halte{
19     adminalasi child;//p ka node halte
20     adminalasi next; // p ka node relasi selan
21 };
22
23
24 adminalasi newhalte(adminalasi child);
25 void inserthalte(adminalasi parent, adminalasi n);
26 void delcalasai(adminalasi parent, string namahalte);
27 adminalasi findhalte(adminalasi parent, string namahalte);
28
29 //laporann:
30
31 void showChildrenParent(adminalasi parent); //menampilkan child dari parent tertentu
32 void showAllParentWithChild(niexmu ir); //menampil seciap parent beserta childnya
33 void showParentOfChild(niexmu ir, string namahalte); //parent yang berelasi dengan child tertentu
34
35 //counting
36 int countHalteParent(adminalasi parent);
37 int countHalteChild(niexmu ir, string namahalte);
38 int countChildInHalte(niexmu nh, niexmu ir);
39
40
41 void editHalte(adminalasi parent, niexmu ih, string oldhalte, string newhalte);
42
43
44 //tambahan responsito
45 void showChildrenParent(niexmu ir, niexmu ch);
46 void showAllData(niexmu ir, niexmu ch);
47
48
49 //tambah yang memungkinkan
50 void delcalasaiInHalte(niexmu ar, string namahalte);
51
52
53
54
55
56 #endif // _A_A__HALTE_H_
```

R.cpp X R.h X Rute.h X *Rute.cpp

```
1  #include<iostream>
2  #include "R.h"
3  using namespace std;
4
5
6  Admalaasi newmalasi(Admalaasi child){
7      Admalaasi n = new malasi;
8      n->child = child;
9      n->next = NULL;
10     return n;
11 }
12
13
14 void insertmalasi(Admalaasi parent, Admalaasi n){
15     if(parent->firstmalasi == NULL){
16         parent->firstmalasi = n;
17     }else{
18         Admalaasi p = parent->firstmalasi;
19         while(p->next != NULL){
20             p = p->next;
21         }
22         p->next = n;
23     }
24 }
25
26
27
28 Admalaasi findmalasi(Admalaasi parent, string namamalasi){
29     Admalaasi p = parent->firstmalasi;
30     while(p != NULL){
31         if(p->child->namamalasi == namamalasi){
32             return p;
33         }
34         p = p->next;
35     }
36     return NULL;
37 }
38
39
40
41 void deletemalasi(Admalaasi parent, string namamalasi){
42     if(parent->firstmalasi == NULL) return;
43
44     Admalaasi cur = parent->firstmalasi;
45     Admalaasi prev = NULL;
46
47     while(cur != NULL && cur->child->namamalasi != namamalasi){
48         prev = cur;
49         cur = cur->next;
50     }
51
52     if(cur == NULL) return;
53
54     if(prev == NULL){
55         parent->firstmalasi = cur->next;
56     } else {
57         prev->next = cur->next;
58     }
59
60     delete cur;
61 }
62
```

```
void showChildrenParent(adrNode parent){
    //jika ruta tidak ada
    if(parent == NULL){
        cout << "Rute tidak ditemukan\n";
        return;
    }

    cout << "Rute " << parent->namarute << " memiliki halte:\n";
    adrNode p = parent->firstchild;

    //jika ruta tidak punya relasi
    if(p == NULL){
        cout << "[tidak ada halte]\n";
        return;
    }

    while(p != NULL){
        cout << "-" << p->child->namahalte << endl;
        p = p->next;
    }
}

void showAllParentWithChild(nieNode rr){
    adrNode r = rr.first;
    while(r != NULL){
        showChildrenParent(r);
        r = r->next;
    }
}

void showParentOfChild(nieNode rr, string namahalte){
    adrNode r = rr.first;
    bool found = false;

    while(r != NULL){
        if(findHalte(r, namahalte) != NULL){
            cout << "Rute: " << r->namarute << endl;
            found = true;
        }
        r = r->next;
    }

    if(!found){
        cout << "Relasi tidak memiliki ruta (parent)\n";
    }
}

//count
int countChildrenParent(adrNode parent){
    int count = 0;
    adrNode p = parent->firstchild;
    while(p != NULL){
        count++;
        p = p->next;
    }

    return count;
}
```

```
131
132
133     int countRelasiChild(ListRute Lr, string namaHalte){
134         int count = 0;
135         adrRute r = Lr.first;
136
137         while(r != NULL){
138             if(findRelasi(r, namaHalte) != NULL){
139                 count++;
140             }
141             r = r->next;
142         }
143         return count;
144     }
145
146
147     int countChildNoRelasi(ListHalte Lh, ListRute Lr){
148         int count = 0;
149         adrHalte h = Lh.first;
150
151         while(h != NULL){
152             if(countRelasiChild(Lr, h->namaHalte) == 0){
153                 count++;
154             }
155             h = h->next;
156         }
157         return count;
158     }
159
160
161     void editRelasi(adrRute parent, ListHalte Lh, string oldHalte, string newHalte){
162         if(parent == NULL) return;
163
164         //cari relasi lama
165         adrRelasi R = findRelasi(parent, oldHalte);
166         if( R == NULL){
167             cout << " Relasi lama tidak di temukan \n";
168             return;
169         }
170
171         //cari halte baru
172         adrHalte hNew = findHalte(Lh, newHalte);
173         if(hNew == NULL){
174             cout << " Halte baru tidak ditemukan\n";
175             return;
176         }
177
178         //cakdup
179         if(findRelasi(parent, newHalte) != NULL){
180             cout << " Halte sudah ada di Rute ini\n";
181             return;
182         }
183
184         //gantiR
185         R->child = hNew;
186
187         cout << " Relasi berhasil diubah\n";
188
189     }
190
191 }
```

```

193 void showAllData(ListRute Lr, ListHalte Lh) {
194
195     cout << "----- Data Rute & Halte -----\\n\\n";
196
197     // 1. Parent + Child
198     if(Lr.first == NULL){
199         cout << "[TIDAK ADA RUTE]\\n\\n";
200     } else {
201         adrRute r = Lr.first;
202         while(r != NULL){
203             cout << "Rute: " << r->namaRute << endl;
204
205             if(r->firstRelasi == NULL){
206                 cout << " [TIDAK MEMILIKI HALTE]\\n";
207             } else {
208                 adrRelasi rel = r->firstRelasi;
209                 while(rel != NULL){
210                     if(rel->child != NULL){
211                         cout << " - " << rel->child->namaHalte << endl;
212                     } else {
213                         cout << " - [RELASI RUSAK]\\n";
214                     }
215                     rel = rel->next;
216                 }
217             }
218
219             cout << endl;
220             r = r->next;
221         }
222     }
223
224     // 2. Child tanpa Parent
225     cout << "Halte tanpa Rute:\\n";
226
227     if(Lh.first == NULL){
228         cout << "[TIDAK ADA HALTE]\\n";
229     } else {
230         adrHalte h = Lh.first;
231         bool ada = false;
232
233         while(h != NULL){
234             if(countRelasiChild(Lr, h->namaHalte) == 0){
235                 cout << " - " << h->namaHalte << endl;
236                 ada = true;
237             }
238             h = h->next;
239         }
240
241         if(!ada){
242             cout << "[SEMUA HALTE SUDAH TERHUBUNG]\\n";
243         }
244     }
245
246     cout << "\\n-----\\n";
247 }
248
249
250 // tambahan
251
252 void deleteRelasiByHalte(ListRute *Lr, string namaHalte){
253     adrRute r = Lr->first;
254     while(r != NULL){
255         deleteRelasi(r, namaHalte);
256         r = r->next;
257     }
258 }
259
260
261

```