


 MelvinLecoy / gitcode Private[Code](#) [Issues](#) [Pull requests](#) [Actions](#) [Projects](#) [Security](#) [Insights](#) [Settings](#) master ▾

...

[gitcode](#) / [p4-graph](#) / graph.h

Kwan Ting Lau gitcode

 History 0 contributors

380 lines (339 sloc) | 9.48 KB

...

```
1 // Project Identifier: 9B734EC0C043C5A836EA0EBE4BEFEA164490B2C7
2
3 #include "float.h"
4 #include "getopt.h"
5 #include "string.h"
6 #include <algorithm>
7 #include <cfloat>
8 #include <cmath>
9 #include <cstdlib>
10 #include <deque>
11 #include <iomanip>
12 #include <iostream>
13 #include <limits>
14 #include <queue>
15 #include <stack>
16 #include <string>
17 #include <vector>
18 using namespace std;
19
20 #define EPS 1e-6
21
22 struct node { // check
23     int lhs;
24     int rhs;
25     char room;
26     node(int lhs, int rhs) : lhs(lhs), rhs(rhs) {
27
28         if (lhs < 0 && rhs < 0) {
29             room = 'L';
30         } else if (lhs == 0 && rhs <= 0) {
31             room = 'D';
32         } else if (lhs <= 0 && rhs == 0) {
33             room = 'D';
34
35         } else {
```

```
36     room = '0';
37 }
38 }
39 };
40
41 struct three_v { // check
42     bool kv;
43     double dv;
44     int pv;
45     three_v() : kv(0), dv(numeric_limits<double>::infinity()), pv(-1) {}
46     three_v(bool k, double d, int p) : kv(k), dv(d), pv(p) {}
47 };
48
49 class graph {
50 public:
51     int mode;
52     double total_distance;
53     double upper;
54     const double INFNTY = numeric_limits<double>::infinity();
55
56     vector<vector<double>> distance;
57     vector<int> opt_index;
58     vector<node> vertex;
59     vector<three_v> Prims;
60
61     double get_distance(node &x, node &y) { // check
62         double d1 = static_cast<double>(x.lhs) - static_cast<double>(y.lhs);
63         double d2 = static_cast<double>(x.rhs) - static_cast<double>(y.rhs);
64         return sqrt(d1 * d1 + d2 * d2);
65     }
66
67     double get_mst_distance(const node &x, const node &y) {
68         if ((x.room == 'L' && y.room == '0') || (x.room == '0' && y.room == 'L')) {
69             return INFNTY;
70         } else {
71             double d1 = static_cast<double>(x.lhs) - static_cast<double>(y.lhs);
72             double d2 = static_cast<double>(x.rhs) - static_cast<double>(y.rhs);
73             return sqrt(d1 * d1 + d2 * d2);
74         }
75     }
76
77     void genPerms(vector<int> &path, size_t permLength) { // check
78         if (permLength == path.size()) {
79             total_distance += distance[0][path.back()];
80             if (total_distance <= upper) {
81                 upper = total_distance;
82                 opt_index = path;
83             }
84             total_distance -= distance[0][path.back()];
85             return;
86         }
87         if (!promising(path, permLength)) {
88             return;
89         }
```

```
90     for (size_t i = permLength; i < path.size(); ++i) {
91         swap(path[permLength], path[i]);
92         total_distance += distance[path[permLength]][path[permLength - 1]];
93         genPerms(path, permLength + 1);
94         total_distance -= distance[path[permLength]][path[permLength - 1]];
95         swap(path[permLength], path[i]);
96     }
97 }
98
99 bool promising(vector<int> &path, size_t permLength) { // check
100     int num = path.size() - permLength;
101     if (num < 2) {
102         return total_distance + distance[path[0]][path[permLength]] +
103             distance[path[permLength - 1]][path[permLength]] <=
104             upper + EPS;
105     } else {
106         vector<int> min_index(num, 0);
107         vector<double> min_distance(num, DBL_MAX);
108         min_index[0] = -1;
109         min_distance[0] = 0;
110
111         int previuos = 0;
112         int count = 0;
113         int tmp_index;
114         double tmp_distance;
115         double tmp;
116         double lower = 0;
117
118         while (count < num - 1) {
119             tmp_index = -1;
120             tmp_distance = DBL_MAX;
121             for (int i = 0; i < num; ++i) {
122                 if (min_index[i] >= 0) {
123                     tmp = distance[path[permLength + previuos]][path[permLength + i]];
124
125                     if (tmp < min_distance[i]) {
126                         min_distance[i] = tmp;
127                         min_index[i] = previuos;
128                     }
129                     if (min_distance[i] < tmp_distance) {
130                         tmp_distance = min_distance[i];
131                         tmp_index = i;
132                     }
133                 }
134             }
135             lower += tmp_distance;
136             min_index[tmp_index] = -1;
137             previuos = tmp_index;
138             ++count;
139         }
140         double a = DBL_MAX;
141         double b = DBL_MAX;
142
143         for (int i = 0; i < num; ++i) {
```

```
144     tmp = distance[path[0]][path[permLength + i]];
145     if (tmp < a) {
146         a = tmp;
147     }
148     tmp = distance[path[permLength - 1]][path[permLength + i]];
149     if (tmp < b) {
150         b = tmp;
151     }
152 }
153 lower += total_distance + a + b;
154 return lower <= upper + EPS;
155 }
156 }
157
158
159
160 void mst() {
161     int num;
162     int first;
163     int second;
164     double total_distance;
165
166     cin >> num;
167     vertex.reserve(num);
168     for (int i = 0; i < num; ++i) {
169         cin >> first >> second;
170         vertex.emplace_back(first, second);
171     }
172
173     total_distance = 0;
174     Prims.resize(num);
175     Prims[0].dv = 0;
176     for (int i = 0; i < num; ++i) {
177         double D = INFNTY;
178         int V = -1;
179         for (uint32_t j = 0; j < vertex.size(); ++j) {
180             if (Prims[j].kv == false && Prims[j].dv < D) {
181                 D = Prims[j].dv;
182                 V = static_cast<int>(j);
183             }
184         }
185         Prims[static_cast<uint32_t>(V)].kv = true;
186         total_distance += D;
187         for (uint32_t k = 0; k < vertex.size(); ++k) {
188             if (Prims[k].kv == false &&
189                 get_mst_distance(vertex[static_cast<uint32_t>(V)], vertex[k]) <
190                 Prims[k].dv) {
191                 Prims[static_cast<uint32_t>(k)].dv =
192                     get_mst_distance(vertex[static_cast<uint32_t>(V)], vertex[k]);
193                 Prims[static_cast<uint32_t>(k)].pv = V;
194             }
195         }
196     }
197     cout << total_distance << "\n";
```

```
198     for (int i = 0; i < num; ++i) {
199         if (Prims[i].pv != -1) {
200             if (Prims[i].pv < i) {
201                 cout << Prims[i].pv << " " << i << "\n";
202             } else {
203                 cout << i << " " << Prims[i].pv << "\n";
204             }
205         }
206     }
207 }
208
209 void fast() { // check
210
211     int num;
212     int first;
213     int second;
214     double tmp;
215     double tmp_distance;
216     int tmp_index;
217     vector<int> path = {0, 1, 2};
218
219     cin >> num;
220     vertex.reserve(num);
221     for (int i = 0; i < num; ++i) {
222         cin >> first >> second;
223         vertex.emplace_back(first, second);
224     }
225
226     path.reserve(num);
227     double total_distance = get_distance(vertex[0], vertex[1]) +
228                             get_distance(vertex[1], vertex[2]) +
229                             get_distance(vertex[2], vertex[0]);
230
231     for (int i = 3; i < num; ++i) {
232         tmp_distance = get_distance(vertex[i], vertex[0]) +
233                       get_distance(vertex[i], vertex[path[i - 1]]) -
234                       get_distance(vertex[0], vertex[path[i - 1]]);
235         tmp_index = 0;
236         for (int j = 1; j < i; ++j) {
237             tmp = get_distance(vertex[i], vertex[path[j]]) +
238                  get_distance(vertex[i], vertex[path[j - 1]]) -
239                  get_distance(vertex[path[j]], vertex[path[j - 1]]);
240             if (tmp < tmp_distance) {
241                 tmp_distance = tmp;
242                 tmp_index = j;
243             }
244         }
245         total_distance += tmp_distance;
246         if (tmp_index == 0) {
247             path.emplace_back(i);
248
249         } else {
250             path.insert(path.begin() + tmp_index, i);
251         }
```

```
252     }
253     cout << total_distance << "\n";
254     for (auto p : path) {
255         cout << p << " ";
256     }
257 }
258
259 void opt() { // check
260     int num;
261     int first;
262     int second;
263     double tmp;
264     vector<int> path;
265
266     cin >> num;
267     vertex.reserve(num);
268     path.reserve(num);
269     distance.resize(num, vector<double>(num, 0));
270
271     for (int i = 0; i < num; ++i) {
272         cin >> first >> second;
273         vertex.emplace_back(first, second);
274         path.emplace_back(i);
275
276         for (int j = 0; j < i; ++j) {
277             tmp = get_distance(vertex[i], vertex[j]);
278             distance[i][j] = tmp;
279             distance[j][i] = tmp;
280         }
281     }
282     total_distance = 0;
283     upper = arb_insertion();
284     upper++;
285     genPerms(path, 1);
286     cout << upper << "\n";
287     for (auto o : opt_index) {
288         cout << o << " ";
289     }
290 }
291
292 double arb_insertion() { // check
293     int num = vertex.size();
294     vector<int> path = {0, 1, 2};
295     vector<int> tail_path;
296     path.reserve(num);
297
298     double total_distance = distance[0][1] + distance[1][2] + distance[2][0];
299     double tmp_distance;
300     int tmp_index;
301     double tmp;
302
303     for (int i = 3; i < num; ++i) {
304         tmp_distance =
305             distance[i][0] + distance[i][path[i - 1]] - distance[0][path[i - 1]];
```

```
306     tmp_index = 0;
307     for (int j = 1; j < i; ++j) {
308         tmp = distance[i][path[j]] + distance[i][path[j - 1]] -
309             distance[path[j]][path[j - 1]];
310         if (tmp < tmp_distance) {
311             tmp_distance = tmp;
312             tmp_index = j;
313         }
314     }
315     total_distance += tmp_distance;
316     if (tmp_index == 0) {
317         path.emplace_back(i);
318     } else {
319         path.insert(path.begin() + tmp_index, i);
320     }
321 }
322 return total_distance;
323 }
324
325 void getMode(int argc, char *argv[]) { // check
326
327     string mode;
328     opterr = false;
329     int choice;
330     int option_index = 0;
331
332     option long_options[] = {
333         {"mode", required_argument, nullptr, 'm'},
334         {"help", no_argument, nullptr, 'h'},
335         {nullptr, 0, nullptr, '\0'},
336     };
337
338     while ((choice = getopt_long(argc, argv, "hm:", long_options,
339                                &option_index)) != -1) {
340         switch (choice) {
341             case 'h':
342                 cout << "something"
343                     << "\n";
344                 exit(1);
345                 break;
346
347             case 'm':
348                 if (!optarg) {
349                     cerr << "Error: No mode specified"
350                         << "\n";
351                     exit(1);
352                 }
353                 if (!strcmp(optarg, "MST")) {
354                     mst();
355                 } else if (!strcmp(optarg, "FASTTSP")) {
356                     fast();
357                 } else if (!strcmp(optarg, "OPTTSP")) {
358                     opt();
359                 } else {
```

```
360         cerr << "Error : Invalid mode"
361         << "\n";
362         exit(1);
363     }
364     break;
365
366     default:
367         cerr << "Unknown command line option" << '\n';
368         exit(1);
369     }
370 }
371 }
372 };
373
374 int main(int argc, char **argv) {
375     ios_base::sync_with_stdio(false);
376     cout << setprecision(2);
377     cout << fixed;
378     auto g = graph();
379     g.getMode(argc, argv);
380 }
```

[Give feedback](#)