


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

...

[gitcode](#) / [p1-bfsdfs](#) / [project1.cpp](#)

Kwan Ting Lau gitcode

 History 0 contributors

417 lines (377 sloc) | 10.3 KB

...

```
1 // Project identifier: B99292359FFD910ED13A7E6C7F9705B8742F0D79
2 #include <array>
3 #include <cstdlib>
4 #include <deque>
5 #include <getopt.h>
6 #include <iostream>
7 #include <limits>
8 #include <string>
9 #include <vector>
10 using namespace std;
11
12 struct castle {
13     char input_mode;
14     char output_mode;
15     uint32_t R;
16     uint32_t N;
17     uint32_t pipe_num;
18     char pipe_char{};
19
20     vector<vector<vector<char>>> map_1;
21     vector<vector<vector<char>>> map_2;
22
23     array<uint32_t, 3> coordinate;
24     array<uint32_t, 3> current;
25
26     deque<array<uint32_t, 3>> container;
27
28     array<uint32_t, 3> n;
29     array<uint32_t, 3> e;
30     array<uint32_t, 3> w;
31     array<uint32_t, 3> s;
32     array<uint32_t, 3> p;
33     array<uint32_t, 3> c;
34     bool dfs = false;
35     int tiles_discovered = 1;
```

```
36
37 castle() {
38     cin >> input_mode >> R >> N;
39     map_1.resize(R, vector<vector<char>>(N, vector<char>(N, '.')));
40     map_2.resize(R, vector<vector<char>>(N, vector<char>(N, '\\0')));
41 }
42
43 bool valid(char c) {
44     return c == 'S' || c == 'C' || c == '.' || c == '!' || c == '#' ||
45         (c >= '0' && c <= '9');
46 }
47
48 void get_inputfile_M() { // ok
49     char read;
50     for (uint32_t room = 0; room < R; room++) {
51         for (uint32_t row = 0; row < N; row++) {
52             cin >> read;
53             while (read == '/') {
54                 cin.ignore(numeric_limits<streamsize>::max(), '\\n');
55                 cin >> read;
56             }
57             for (uint32_t col = 0; col < N - 1; col++) {
58                 if (!valid(read))
59                     exit(1);
60                 map_1[room][row][col] = read;
61                 if (read == 'S') {
62                     coordinate[0] = room;
63                     coordinate[1] = row;
64                     coordinate[2] = col;
65                     container.push_back(coordinate);
66                 }
67                 cin >> read;
68             }
69             if (!valid(read)) {
70                 cerr << "Unknown map character" << endl;
71                 exit(1);
72             }
73             map_1[room][row][N - 1] = read;
74             if (read == 'S') {
75                 coordinate[0] = room;
76                 coordinate[1] = row;
77                 coordinate[2] = N - 1;
78                 container.push_back(coordinate);
79             }
80         }
81     }
82 }
83
84 void get_outputfile_M() {
85     cout << "Start in room " << coordinate[0] << ", "
86         << " row " << coordinate[1] << ", "
87         << " column " << coordinate[2] << '\\n';
88     // backtrack
89     for (uint32_t room = 0; room < R; room++) {
```

```
90     cout << "//castle room " << room << '\n';
91     for (uint32_t row = 0; row < N; row++) {
92         for (uint32_t col = 0; col < N; col++) {
93             cout << map_1[room][row][col];
94         }
95         cout << '\n';
96     }
97 }
98 }
99
100 void get_inputfile_L() { // ok
101     char char_1;
102     char mark;
103
104     uint32_t room_1;
105     uint32_t row_1;
106     uint32_t col_1;
107     while (cin >> char_1) {
108         if (char_1 == '/') {
109             cin.ignore(numeric_limits<streamsize>::max(), '\n');
110             continue;
111         }
112         cin >> room_1 >> char_1 >> row_1 >> char_1 >> col_1 >> char_1 >> mark >>
113             char_1;
114         if (room_1 >= R) {
115             cerr << "Invalid room number" << endl;
116             exit(1);
117         }
118         if (row_1 >= N) {
119             cerr << "Invalid row number" << endl;
120             exit(1);
121         }
122         if (col_1 >= N) {
123             cerr << "Invalid column number" << endl;
124             exit(1);
125         }
126         if (!valid(mark)) {
127             cerr << "Unknown map character" << endl;
128             exit(1);
129         }
130         map_1[room_1][row_1][col_1] = mark;
131         if (mark == 'S') {
132             coordinate[0] = room_1;
133             coordinate[1] = row_1;
134             coordinate[2] = col_1;
135             container.push_back(coordinate);
136         }
137     }
138 }
139
140 void get_outputfile_L() {
141     cout << "Path taken:\n";
142     back();
143 }
```

```
144
145 bool isValid(char s) {
146     if (s >= '0' && s <= '9') {
147         return true;
148     } else if (s == 'C' || s == '.') {
149         return true;
150     }
151     return false;
152 }
153
154 bool whther_pipe(char s) {
155     if (s >= '0' && s <= '9') {
156         return true;
157     }
158     return false;
159 }
160
161 void search_add() {
162     while (!container.empty()) {
163         if (dfs) {
164             current = container.back();
165             container.pop_back();
166         } else {
167             current = container.front();
168             container.pop_front();
169         }
170         uint32_t room = current[0];
171         uint32_t row = current[1];
172         uint32_t column = current[2];
173
174         char place = map_1[room][row][column];
175
176         if (whther_pipe(place)) {
177             p[0] = static_cast<uint32_t>(place - '0');
178             if (p[0] >= R)
179                 continue;
180             if (isValid(map_1[p[0]][row][column]) &&
181                 map_2[static_cast<uint32_t>(place - '0')][row][column] == 0) {
182                 map_2[p[0]][row][column] = static_cast<char>(room + '0');
183                 p[1] = current[1];
184                 p[2] = current[2];
185                 ++tiles_discovered;
186                 container.push_back(p);
187                 if (map_1[p[0]][row][column] == 'C')
188                     return;
189             }
190             continue;
191         }
192
193         if (current[1] >= 1) {
194             if (isValid(map_1[room][row - 1][column]) &&
195                 map_2[room][row - 1][column] == 0) {
196                 s[0] = current[0];
197                 s[1] = current[1] - 1;
```

```
198     s[2] = current[2];
199     container.push_back(s);
200     ++tiles_discovered;
201     map_2[room][row - 1][column] = 'n';
202     if (map_1[room][row - 1][column] == 'C') {
203         return;
204     }
205 }
206 }
207
208 if (current[2] + 1 < N) {
209     if (isInvalid(map_1[room][row][column + 1]) &&
210         map_2[room][row][column + 1] == 0) {
211         e[0] = current[0];
212         e[1] = current[1];
213         e[2] = current[2] + 1;
214         container.push_back(e);
215         ++tiles_discovered;
216         map_2[room][row][column + 1] = 'e';
217         if (map_1[room][row][column + 1] == 'C') {
218             return;
219         }
220     }
221 }
222
223 if (current[1] + 1 < N) {
224     if (isInvalid(map_1[room][row + 1][column]) &&
225         map_2[room][row + 1][column] == 0) {
226         n[0] = current[0];
227         n[1] = current[1] + 1;
228         n[2] = current[2];
229         container.push_back(n);
230         ++tiles_discovered;
231         map_2[room][row + 1][column] = 's';
232         if (map_1[room][row + 1][column] == 'C') {
233             return;
234         }
235     }
236 }
237
238 if (current[2] >= 1) {
239     if (isInvalid(map_1[room][row][column - 1]) &&
240         map_2[room][row][column - 1] == 0) {
241         w[0] = current[0];
242         w[1] = current[1];
243         w[2] = current[2] - 1;
244         container.push_back(w);
245         ++tiles_discovered;
246         map_2[room][row][column - 1] = 'w';
247         if (map_1[room][row][column - 1] == 'C') {
248             return;
249         }
250     }
251 }
```

```
252     }
253 }
254
255 void direction() {
256     auto a = container.back();
257     while (a != coordinate) { // a != S
258         auto room_2 = a[0], row_2 = a[1], col_2 = a[2];
259         auto ch = map_2[room_2][row_2][col_2];
260         // change the point of
261         if (pipe_char != 0) {
262             map_2[room_2][row_2][col_2] = pipe_char;
263             pipe_char = 0;
264         }
265         if (ch == 'n') {
266             ++a[1];
267             map_1[room_2][a[1]][col_2] = 'n';
268         }
269         if (ch == 'e') {
270             --a[2];
271             map_1[room_2][row_2][a[2]] = 'e';
272         }
273         if (ch == 's') {
274             --a[1];
275             map_1[room_2][a[1]][col_2] = 's';
276         }
277         if (ch == 'w') {
278             ++a[2];
279             map_1[room_2][row_2][a[2]] = 'w';
280         }
281         if (ch >= '0' && ch <= '9') {
282             pipe_num = static_cast<uint32_t>(ch - '0');
283             pipe_char = static_cast<char>(room_2 + '0');
284             map_1[pipe_num][row_2][col_2] = 'p';
285             a[0] = pipe_num;
286         }
287     }
288 }
289
290 void back() {
291     auto a = coordinate;
292     while (a != container.back()) {
293         uint32_t room_2 = a[0];
294         uint32_t row_2 = a[1];
295         uint32_t col_2 = a[2];
296         cout << '(' << room_2 << ',' << row_2 << ',' << col_2 << ','
297             << map_1[room_2][row_2][col_2] << ")\n";
298         if (map_1[room_2][row_2][col_2] == 's') {
299             ++a[1];
300         }
301         if (map_1[room_2][row_2][col_2] == 'w') {
302             --a[2];
303         }
304         if (map_1[room_2][row_2][col_2] == 'n') {
305             --a[1];
```

```
306     }
307     if (map_1[room_2][row_2][col_2] == 'e') {
308         ++a[2];
309     }
310     if (map_1[room_2][row_2][col_2] == 'p') {
311         a[0] = static_cast<uint32_t>(map_2[room_2][row_2][col_2] - '0');
312     }
313 }
314 }
315
316 void getMode(int argc, char *argv[]) {
317
318     string mode;
319
320     opterr = false;
321     int choice;
322     int option_index = 0;
323     option long_options[] = {
324
325
326
327         {"stack", no_argument, nullptr, 's'},
328         {"queue", no_argument, nullptr, 'q'},
329         {"output", required_argument, nullptr, 'o'},
330         {"help", no_argument, nullptr, 'h'},
331         {nullptr, 0, nullptr, '\0'}};
332     bool yes_p = false;
333
334
335     bool op = false;
336
337     while ((choice = getopt_long(argc, argv, "sqo:h", long_options,
338                                &option_index)) != -1) {
339         switch (choice) {
340             case 'h':
341                 exit(0);
342
343             case 's':
344                 dfs = true;
345                 if (yes_p) {
346                     cerr << "Stack or queue can only be specified once" << endl;
347                     exit(1);
348                 } else {
349                     yes_p = true;
350                 }
351                 break;
352
353             case 'q':
354                 dfs = false;
355                 if (yes_p) {
356                     cerr << "Stack or queue can only be specified once" << endl;
357                     exit(1);
358                 } else {
359                     yes_p = true;
```

```
360     }
361     break;
362
363     case 'o':
364         op = true;
365         mode = optarg;
366         if (mode != "M" && mode != "L") {
367             cerr << "Unknown command line option" << '\n';
368             exit(1);
369         }
370         output_mode = mode[0];
371         break;
372
373     default:
374         cerr << "Unknown command line option" << '\n';
375         exit(1);
376     }
377 }
378 if (!yes_p) {
379     cerr << "Stack or queue must be specified" << endl;
380     exit(1);
381 }
382 if (!op) {
383     output_mode = 'M';
384 }
385 }
386 };
387
388 int main(int argc, char **argv) {
389     ios_base::sync_with_stdio(false);
390     castle c;
391     c.getMode(argc, argv);
392     char input = c.input_mode;
393     if (input == 'M') {
394         c.get_inputfile_M();
395     } else if (input == 'L') {
396         c.get_inputfile_L();
397     }
398
399     c.map_2[c.coordinate[0]][c.coordinate[1]][c.coordinate[2]] = 'S';
400     c.search_add();
401     if (c.container.empty()) {
402         // No solution, N tiles discovered.
403         cout << "No solution, " << c.tiles_discovered << " tiles discovered."
404             << endl;
405         return 0;
406     }
407     c.direction();
408
409     char output = c.output_mode;
410     if (output == 'L') {
411         c.get_outputfile_L();
412     }
413 }
```



```
414     if (output == 'M') {  
415         c.get_outputfile_M();  
416     }  
417 }
```

[Give feedback](#)