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

```cpp
// Project identifier: B99292359FFD910ED13A7E6C7F9705B8742F0D79
#include <array>
#include <cstdlib>
#include <deque>
#include <getopt.h>
#include <iostream>
#include <limits>
#include <string>
#include <vector>
using namespace std;

struct castle {
  char input_mode;
  char output_mode;
  uint32_t R;
  uint32_t N;
  uint32_t pipe_num;
  char pipe_char{};

  vector<vector<vector<char>>> map_1;
  vector<vector<vector<char>>> map_2;

  array<uint32_t, 3> coordinate;
  array<uint32_t, 3> current;

  deque<array<uint32_t, 3>> container;

  array<uint32_t, 3> n;
  array<uint32_t, 3> e;
  array<uint32_t, 3> w;
  array<uint32_t, 3> s;
  array<uint32_t, 3> p;
  array<uint32_t, 3> c;
  bool dfs = false;
  int tiles_discovered = 1;

  castle() {
    cin >> input_mode >> R >> N;
    map_1.resize(R, vector<vector<char>>(N, vector<char>(N, '.')));
    map_2.resize(R, vector<vector<char>>(N, vector<char>(N, '\0')));
  }

  bool valid(char c) {
    return c == 'S' || c == 'C' || c == '.' || c == '!' || c == '#' ||
           (c >= '0' && c <= '9');
  }

  void get_inputfile_M() { // ok
    char read;
    for (uint32_t room = 0; room < R; room++) {
      for (uint32_t row = 0; row < N; row++) {
        cin >> read;
        while (read == '/') {
          cin.ignore(numeric_limits<streamsize>::max(), '\n');
          cin >> read;
        }
        for (uint32_t col = 0; col < N - 1; col++) {
          if (!valid(read))
            exit(1);
          map_1[room][row][col] = read;
          if (read == 'S') {
            coordinate[0] = room;
            coordinate[1] = row;
            coordinate[2] = col;
            container.push_back(coordinate);
          }
          cin >> read;
        }
        if (!valid(read)) {
          cerr << "Unknown map character" << endl;
          exit(1);
        }
        map_1[room][row][N - 1] = read;
        if (read == 'S') {
          coordinate[0] = room;
          coordinate[1] = row;
          coordinate[2] = N - 1;
          container.push_back(coordinate);
        }
      }
    }
  }

  void get_outputfile_M() {
    cout << "Start in room " << coordinate[0] << ","
         << " row " << coordinate[1] << ","
         << " column " << coordinate[2] << '\n';
    // backtrack
    for (uint32_t room = 0; room < R; room++) {
      cout << "//castle room " << room << '\n';
      for (uint32_t row = 0; row < N; row++) {
        for (uint32_t col = 0; col < N; col++) {
          cout << map_1[room][row][col];
        }
        cout << '\n';
      }
    }
  }

  void get_inputfile_L() { // ok
    char char_1;
    char mark;

    uint32_t room_1;
    uint32_t row_1;
    uint32_t col_1;
    while (cin >> char_1) {
      if (char_1 == '/') {
        cin.ignore(numeric_limits<streamsize>::max(), '\n');
        continue;
      }
      cin >> room_1 >> char_1 >> row_1 >> char_1 >> col_1 >> char_1 >> mark >>
          char_1;
      if (room_1 >= R) {
        cerr << "Invalid room number" << endl;
        exit(1);
      }
      if (row_1 >= N) {
        cerr << "Invalid row number" << endl;
        exit(1);
      }
      if (col_1 >= N) {
        cerr << "Invalid column number" << endl;
        exit(1);
      }
      if (!valid(mark)) {
        cerr << "Unknown map character" << endl;
        exit(1);
      }
      map_1[room_1][row_1][col_1] = mark;
      if (mark == 'S') {
        coordinate[0] = room_1;
        coordinate[1] = row_1;
        coordinate[2] = col_1;
        container.push_back(coordinate);
      }
    }
  }

  void get_outputfile_L() {
    cout << "Path taken:\n";
    back();
  }
```

```

bool isInvalid(char s) {
  if (s >= '0' && s <= '9') {
    return true;
  } else if (s == 'C' || s == '.') {
    return true;
  }
  return false;
}

bool whther_pipe(char s) {
  if (s >= '0' && s <= '9') {
    return true;
  }
  return false;
}

void search_add() {
  while (!container.empty()) {
    if (dfs) {
      current = container.back();
      container.pop_back();
    } else {
      current = container.front();
      container.pop_front();
    }
    uint32_t room = current[0];
    uint32_t row = current[1];
    uint32_t column = current[2];

    char place = map_1[room][row][column];

    if (whther_pipe(place)) {
      p[0] = static_cast<uint32_t>(place - '0');
      if (p[0] >= R)
        continue;
      if (isInvalid(map_1[p[0]][row][column]) &&
          map_2[static_cast<uint32_t>(place - '0')][row][column] == 0) {
        map_2[p[0]][row][column] = static_cast<char>(room + '0');
        p[1] = current[1];
        p[2] = current[2];
        ++tiles_discovered;
        container.push_back(p);
        if (map_1[p[0]][row][column] == 'C')
          return;
      }
      continue;
    }

    if (current[1] >= 1) {
      if (isInvalid(map_1[room][row - 1][column]) &&
          map_2[room][row - 1][column] == 0) {
        s[0] = current[0];
        s[1] = current[1] - 1;
        s[2] = current[2];
        container.push_back(s);
        ++tiles_discovered;
        map_2[room][row - 1][column] = 'n';
        if (map_1[room][row - 1][column] == 'C') {
          return;
        }
      }
    }

    if (current[2] + 1 < N) {
      if (isInvalid(map_1[room][row][column + 1]) &&
          map_2[room][row][column + 1] == 0) {
        e[0] = current[0];
        e[1] = current[1];
        e[2] = current[2] + 1;
        container.push_back(e);
        ++tiles_discovered;
        map_2[room][row][column + 1] = 'e';
        if (map_1[room][row][column + 1] == 'C') {
          return;
        }
      }
    }

    if (current[1] + 1 < N) {
      if (isInvalid(map_1[room][row + 1][column]) &&
          map_2[room][row + 1][column] == 0) {
        n[0] = current[0];
        n[1] = current[1] + 1;
        n[2] = current[2];
        container.push_back(n);
        ++tiles_discovered;
        map_2[room][row + 1][column] = 's';
        if (map_1[room][row + 1][column] == 'C') {
          return;
        }
      }
    }

    if (current[2] >= 1) {
      if (isInvalid(map_1[room][row][column - 1]) &&
          map_2[room][row][column - 1] == 0) {
        w[0] = current[0];
        w[1] = current[1];
        w[2] = current[2] - 1;
        container.push_back(w);
        ++tiles_discovered;
        map_2[room][row][column - 1] = 'w';
        if (map_1[room][row][column - 1] == 'C') {
          return;
        }
      }
    }
  }
}

void direction() {
  auto a = container.back();
  while (a != coordinate) { // a != S
    auto room_2 = a[0], row_2 = a[1], col_2 = a[2];
    auto ch = map_2[room_2][row_2][col_2];
    // change the point of
    if (pipe_char != 0) {
      map_2[room_2][row_2][col_2] = pipe_char;
      pipe_char = 0;
    }
    if (ch == 'n') {
      ++a[1];
      map_1[room_2][a[1]][col_2] = 'n';
    }
    if (ch == 'e') {
      --a[2];
      map_1[room_2][row_2][a[2]] = 'e';
    }
    if (ch == 's') {
      --a[1];
      map_1[room_2][a[1]][col_2] = 's';
    }
    if (ch == 'w') {
      ++a[2];
      map_1[room_2][row_2][a[2]] = 'w';
    }
    if (ch >= '0' && ch <= '9') {
      pipe_num = static_cast<uint32_t>(ch - '0');
      pipe_char = static_cast<char>(room_2 + '0');
      map_1[pipe_num][row_2][col_2] = 'p';
      a[0] = pipe_num;
    }
  }
}

void back() {
  auto a = coordinate;
  while (a != container.back()) {
    uint32_t room_2 = a[0];
    uint32_t row_2 = a[1];
    uint32_t col_2 = a[2];
    cout << '(' << room_2 << ',' << row_2 << ',' << col_2 << ','
         << map_1[room_2][row_2][col_2] << ")\n";
    if (map_1[room_2][row_2][col_2] == 's') {
      ++a[1];
    }
    if (map_1[room_2][row_2][col_2] == 'w') {
      --a[2];
    }
    if (map_1[room_2][row_2][col_2] == 'n') {
      --a[1];
    }
    if (map_1[room_2][row_2][col_2] == 'e') {
      ++a[2];
    }
    if (map_1[room_2][row_2][col_2] == 'p') {
      a[0] = static_cast<uint32_t>(map_2[room_2][row_2][col_2] - '0');
    }
  }
}

void getMode(int argc, char *argv[]) {

  string mode;

  opterr = false;
  int choice;
  int option_index = 0;
  option long_options[] = {



                           {"stack", no_argument, nullptr, 's'},
                           {"queue", no_argument, nullptr, 'q'},
                           {"output", required_argument, nullptr, 'o'},
                           {"help", no_argument, nullptr, 'h'},
                           {nullptr, 0, nullptr, '\0'}};
  bool yes_p = false;


  bool op = false;

  while ((choice = getopt_long(argc, argv, "sqo:h", long_options,
                               &option_index)) != -1) {
    switch (choice) {
    case 'h':
      exit(0);

    case 's':
      dfs = true;
      if (yes_p) {
        cerr << "Stack or queue can only be specified once" << endl;
        exit(1);
      } else {
        yes_p = true;
      }
      break;

    case 'q':
      dfs = false;
      if (yes_p) {
        cerr << "Stack or queue can only be specified once" << endl;
        exit(1);
      } else {
        yes_p = true;
      }
      break;

    case 'o':
      op = true;
      mode = optarg;
      if (mode != "M" && mode != "L") {
        cerr << "Unknown command line option" << '\n';
        exit(1);
      }
      output_mode = mode[0];
      break;

    default:
      cerr << "Unknown command line option" << '\n';
      exit(1);
    }
  }
  if (!yes_p) {
    cerr << "Stack or queue must be specified" << endl;
    exit(1);
  }
  if (!op) {
    output_mode = 'M';
  }
}
};

int main(int argc, char **argv) {
ios_base::sync_with_stdio(false);
castle c;
c.getMode(argc, argv);
char input = c.input_mode;
if (input == 'M') {
  c.get_inputfile_M();
} else if (input == 'L') {
  c.get_inputfile_L();
}

c.map_2[c.coordinate[0]][c.coordinate[1]][c.coordinate[2]] = 'S';
c.search_add();
if (c.container.empty()) {
  // No solution, N tiles discovered.
  cout << "No solution, " << c.tiles_discovered << " tiles discovered."
       << endl;
  return 0;
}
c.direction();

char output = c.output_mode;
if (output == 'L') {
  c.get_outputfile_L();
}

if (output == 'M') {
  c.get_outputfile_M();
}
}
```

Give feedback