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

```
// Project Identifier: 9B734EC0C043C5A836EA0EBE4BEFEA164490B2C7

#include "float.h"
#include "getopt.h"
#include "string.h"
#include <algorithm>
#include <cfloat>
#include <cmath>
#include <cstdlib>
#include <deque>
#include <iomanip>
#include <iostream>
#include <limits>
#include <queue>
#include <stack>
#include <string>
#include <vector>
using namespace std;

#define EPS 1e-6

struct node { // check
  int lhs;
  int rhs;
  char room;
  node(int lhs, int rhs) : lhs(lhs), rhs(rhs) {

    if (lhs < 0 && rhs < 0) {
      room = 'L';
    } else if (lhs == 0 && rhs <= 0) {
      room = 'D';
    } else if (lhs <= 0 && rhs == 0) {
      room = 'D';

    } else {
```

```
      room = 'O';
    }
  }
};

struct three_v { // check
  bool kv;
  double dv;
  int pv;
  three_v() : kv(0), dv(numeric_limits<double>::infinity()), pv(-1) {}
  three_v(bool k, double d, int p) : kv(k), dv(d), pv(p) {}
};

class graph {
public:
  int mode;
  double total_distance;
  double upper;
  const double INFNTY = numeric_limits<double>::infinity();

  vector<vector<double>> distance;
  vector<int> opt_index;
  vector<node> vertex;
  vector<three_v> Prims;

  double get_distance(node &x, node &y) { // check
    double d1 = static_cast<double>(x.lhs) - static_cast<double>(y.lhs);
    double d2 = static_cast<double>(x.rhs) - static_cast<double>(y.rhs);
    return sqrt(d1 * d1 + d2 * d2);
  }

  double get_mst_distance(const node &x, const node &y) {
    if ((x.room == 'L' && y.room == 'O') || (x.room == 'O' && y.room == 'L')) {
      return INFNTY;
    } else {
      double d1 = static_cast<double>(x.lhs) - static_cast<double>(y.lhs);
      double d2 = static_cast<double>(x.rhs) - static_cast<double>(y.rhs);
      return sqrt(d1 * d1 + d2 * d2);
    }
  }

  void genPerms(vector<int> &path, size_t permLength) { // check
    if (permLength == path.size()) {
      total_distance += distance[0][path.back()];
      if (total_distance <= upper) {
        upper = total_distance;
        opt_index = path;
      }
      total_distance -= distance[0][path.back()];
      return;
    }
    if (!promising(path, permLength)) {
      return;
    }
```

```
    for (size_t i = permLength; i < path.size(); ++i) {
      swap(path[permLength], path[i]);
      total_distance += distance[path[permLength]][path[permLength - 1]];
      genPerms(path, permLength + 1);
      total_distance -= distance[path[permLength]][path[permLength - 1]];
      swap(path[permLength], path[i]);
    }
  }

  bool promising(vector<int> &path, size_t permLength) { // check
    int num = path.size() - permLength;
    if (num < 2) {
      return total_distance + distance[path[0]][path[permLength]] +
                 distance[path[permLength - 1]][path[permLength]] <=
             upper + EPS;
    } else {
      vector<int> min_index(num, 0);
      vector<double> min_distance(num, DBL_MAX);
      min_index[0] = -1;
      min_distance[0] = 0;

      int previuos = 0;
      int count = 0;
      int tmp_index;
      double tmp_distance;
      double tmp;
      double lower = 0;

      while (count < num - 1) {
        tmp_index = -1;
        tmp_distance = DBL_MAX;
        for (int i = 0; i < num; ++i) {
          if (min_index[i] >= 0) {
            tmp = distance[path[permLength + previuos]][path[permLength + i]];

            if (tmp < min_distance[i]) {
              min_distance[i] = tmp;
              min_index[i] = previuos;
            }
            if (min_distance[i] < tmp_distance) {
              tmp_distance = min_distance[i];
              tmp_index = i;
            }
          }
        }
        lower += tmp_distance;
        min_index[tmp_index] = -1;
        previuos = tmp_index;
        ++count;
      }
      double a = DBL_MAX;
      double b = DBL_MAX;

      for (int i = 0; i < num; ++i) {
```

```
        tmp = distance[path[0]][path[permLength + i]];
        if (tmp < a) {
          a = tmp;
        }
        tmp = distance[path[permLength - 1]][path[permLength + i]];
        if (tmp < b) {
          b = tmp;
        }
      }
      lower += total_distance + a + b;
      return lower <= upper + EPS;
    }
  }



  void mst() {
    int num;
    int first;
    int second;
    double total_distance;

    cin >> num;
    vertex.reserve(num);
    for (int i = 0; i < num; ++i) {
      cin >> first >> second;
      vertex.emplace_back(first, second);
    }

    total_distance = 0;
    Prims.resize(num);
    Prims[0].dv = 0;
    for (int i = 0; i < num; ++i) {
      double D = INFNTY;
      int V = -1;
      for (uint32_t j = 0; j < vertex.size(); ++j) {
        if (Prims[j].kv == false && Prims[j].dv < D) {
          D = Prims[j].dv;
          V = static_cast<int>(j);
        }
      }
      Prims[static_cast<uint32_t>(V)].kv = true;
      total_distance += D;
      for (uint32_t k = 0; k < vertex.size(); ++k) {
        if (Prims[k].kv == false &&
            get_mst_distance(vertex[static_cast<uint32_t>(V)], vertex[k]) <
                Prims[k].dv) {
          Prims[static_cast<uint32_t>(k)].dv =
              get_mst_distance(vertex[static_cast<uint32_t>(V)], vertex[k]);
          Prims[static_cast<uint32_t>(k)].pv = V;
        }
      }
    }
    cout << total_distance << "\n";
```

```
    for (int i = 0; i < num; ++i) {
      if (Prims[i].pv != -1) {
        if (Prims[i].pv < i) {
          cout << Prims[i].pv << " " << i << "\n";
        } else {
          cout << i << " " << Prims[i].pv << "\n";
        }
      }
    }
  }

  void fast() { // check

    int num;
    int first;
    int second;
    double tmp;
    double tmp_distance;
    int tmp_index;
    vector<int> path = {0, 1, 2};

    cin >> num;
    vertex.reserve(num);
    for (int i = 0; i < num; ++i) {
      cin >> first >> second;
      vertex.emplace_back(first, second);
    }

    path.reserve(num);
    double total_distance = get_distance(vertex[0], vertex[1]) +
                            get_distance(vertex[1], vertex[2]) +
                            get_distance(vertex[2], vertex[0]);

    for (int i = 3; i < num; ++i) {
      tmp_distance = get_distance(vertex[i], vertex[0]) +
                     get_distance(vertex[i], vertex[path[i - 1]]) -
                     get_distance(vertex[0], vertex[path[i - 1]]);
      tmp_index = 0;
      for (int j = 1; j < i; ++j) {
        tmp = get_distance(vertex[i], vertex[path[j]]) +
              get_distance(vertex[i], vertex[path[j - 1]]) -
              get_distance(vertex[path[j]], vertex[path[j - 1]]);
        if (tmp < tmp_distance) {
          tmp_distance = tmp;
          tmp_index = j;
        }
      }
      total_distance += tmp_distance;
      if (tmp_index == 0) {
        path.emplace_back(i);

      } else {
        path.insert(path.begin() + tmp_index, i);
      }
```

```
    }
    cout << total_distance << "\n";
    for (auto p : path) {
      cout << p << " ";
    }
  }

  void opt() { // check
    int num;
    int first;
    int second;
    double tmp;
    vector<int> path;

    cin >> num;
    vertex.reserve(num);
    path.reserve(num);
    distance.resize(num, vector<double>(num, 0));

    for (int i = 0; i < num; ++i) {
      cin >> first >> second;
      vertex.emplace_back(first, second);
      path.emplace_back(i);

      for (int j = 0; j < i; ++j) {
        tmp = get_distance(vertex[i], vertex[j]);
        distance[i][j] = tmp;
        distance[j][i] = tmp;
      }
    }
    total_distance = 0;
    upper = arb_insertion();
    upper++;
    genPerms(path, 1);
    cout << upper << "\n";
    for (auto o : opt_index) {
      cout << o << " ";
    }
  }

  double arb_insertion() { // check
    int num = vertex.size();
    vector<int> path = {0, 1, 2};
    vector<int> tail_path;
    path.reserve(num);

    double total_distance = distance[0][1] + distance[1][2] + distance[2][0];
    double tmp_distance;
    int tmp_index;
    double tmp;

    for (int i = 3; i < num; ++i) {
      tmp_distance =
          distance[i][0] + distance[i][path[i - 1]] - distance[0][path[i - 1]];
```

```
      tmp_index = 0;
      for (int j = 1; j < i; ++j) {
        tmp = distance[i][path[j]] + distance[i][path[j - 1]] -
              distance[path[j]][path[j - 1]];
        if (tmp < tmp_distance) {
          tmp_distance = tmp;
          tmp_index = j;
        }
      }
      total_distance += tmp_distance;
      if (tmp_index == 0) {
        path.emplace_back(i);
      } else {
        path.insert(path.begin() + tmp_index, i);
      }
    }
    return total_distance;
  }

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

    string mode;
    opterr = false;
    int choice;
    int option_index = 0;

    option long_options[] = {
        {"mode", required_argument, nullptr, 'm'},
        {"help", no_argument, nullptr, 'h'},
        {nullptr, 0, nullptr, '\0'},
    };

    while ((choice = getopt_long(argc, argv, "hm:", long_options,
                                 &option_index)) != -1) {
      switch (choice) {
      case 'h':
        cout << "somthing"
             << "\n";
        exit(1);
        break;

      case 'm':
        if (!optarg) {
          cerr << "Error: No mode specified"
               << "\n";
          exit(1);
        }
        if (!strcmp(optarg, "MST")) {
          mst();
        } else if (!strcmp(optarg, "FASTTSP")) {
          fast();
        } else if (!strcmp(optarg, "OPTTSP")) {
          opt();
        } else {
```

```
          cerr << "Error : Invalid mode"
               << "\n";
          exit(1);
        }
        break;

      default:
        cerr << "Unknown command line option" << '\n';
        exit(1);
      }
    }
  }
};

int main(int argc, char **argv) {
  ios_base::sync_with_stdio(false);
  cout << setprecision(2);
  cout << fixed;
  auto g = graph();
  g.getMode(argc, argv);
}
```

Give feedback