CS225 Final Project

Open Flights

Yirong Chen, Leo Luo, Yanzhen Shen

Overview

- Graph Structure
- DFS
- Strongly Connected Components
- PageRank
- Demo
- Leading Question
    - What is the shortest air-path given two airports
    - Which is the most important airport

Development: Graph Structure

- Data Acquisition and Correction
  - Invalid data due to military bases
  - Quotation marks and extra commas
  - Parsed in character by character
- Graph Structure
  - bijection between airport and IATA code

```cpp
class Data {
public:
    /**
     * @brief store a simple directed graph
     *
     * @param airport_is
     * @param airline_is
     */
    Data(std::istream& airport_is, std::istream& airline_is);
    /**
     * @brief Filter data
     *
     * @param data_ori original data object
     * @param allowed_idx all idx in this vector must be valid (in the range; no repeat elements)
     */
    Data(const Data& data_ori, const std::vector<size_t>& allowed_idx);
    const AdjList& GetAdjList() const;
    const AdjMatrix& GetAdjMatrix() const;
    long double ToRadiant(const long double degree);
    unsigned Distance(long double lat1, long double long1, long double lat2, long double long2);
    const Node& GetNode(size_t idx) const;
    size_t GetIdx(const std::string& code) const;
private:
    void ReadAirport(std::istream& airport_is);
    void ReadAirline(std::istream& airline_is);

    std::vector<Node> idx_to_node_;// map index to node
    std::unordered_map<std::string, size_t> code_to_idx_;// map code to idx
    AdjList adj_list_;
    AdjMatrix adj_matrix_;
};
```

Development: DFS

- Function Objects
    - look_next_origin
    - op_before_component
    - op_start_visit
    - op_after_visit

```
 * @param graph adjacency list
 * @param look_next_origin function object that
 * return the next node in the origin considering sequence;
 * return a number greater than or equal to the graph size to stop DFS
 * @param op_before_component function object that
 * op_before_component(origin_idx) is called before the start of component traversal;
 * return component_handle (size_t) that will be passed into op_after_pop
 * @param op_start_visit function object that
 * op_start_visit(curr_node_idx, component_handle) is called after the node is poped from the stack
 * @param op_after_visit function object that
 * op_after_visit(curr_node_idx, component_handle) is called after the node is finished from visiting
 */
template <typename LookNextOrigin, typename OpBeforeComponent, typename OpBeforeVisit, typename OpAfterVisit>
void DFS(const AdjList& graph, LookNextOrigin look_next_origin,
        OpBeforeComponent op_before_component, OpBeforeVisit op_start_visit, OpAfterVisit op_after_visit) {
    size_t n = graph.size(), origin;
    std::vector<bool> visited(graph.size(), false);
    std::function<void(size_t, size_t)> dfs_visit;
    dfs_visit = [&](size_t u, size_t component_handle) {
        op_start_visit(u, component_handle);
        for (size_t v : graph[u]) {
            if (!visited[v]) {
                visited[v] = true;
                dfs_visit(v, component_handle);
            }
        }
        op_after_visit(u, component_handle);
    };
    while ((origin = look_next_origin()) < n) {
        if (visited[origin]) { continue; }
        visited[origin] = true;
        size_t component_handle = op_before_component(origin);
        dfs_visit(origin, component_handle);
    }
}
```

Development: DFS

```cpp
size_t i = 0;
auto look_next_origin = [n, &i, v]() {
    if (i == n) { return n; } ++i;
    return (v + i - 1) % n;
};

auto op_before_component = [&os, &data](size_t origin_idx) {
    os << "New Component where the origin is " << data.GetNode(origin_idx).iata_code << '\n';
    return origin_idx;
};

auto op_start_visit = [&os, &data](size_t curr_node_idx, size_t) {
    const Node& node = data.GetNode(curr_node_idx);
    os << "Start visiting " << node.iata_code << " (" << node.city << ")" << '\n';
};

auto op_after_visit = [&os, &data](size_t curr_node_idx, size_t) {
    const Node& node = data.GetNode(curr_node_idx);
    os << "Finish visiting " << node.iata_code << " (" << node.city << ")" << '\n';
};

DFS(data.GetAdjList(), look_next_origin, op_before_component, op_start_visit, op_after_visit);
```

```
New Component where the origin is CMI
Start visiting CMI (Champaign)
Start visiting ORD (Chicago)
Start visiting YXE (Saskatoon)
Start visiting MSP (Minneapolis)
Start visiting PUJ (Punta Cana)
Start visiting LED (St. Petersburg)
Start visiting PED (Pardubice)
Start visiting DME (Moscow)
Start visiting UKS (Sevastopol)
Start visiting KBP (Kiev)
Start visiting ATH (Athens)
Start visiting KRR (Krasnodar)
Start visiting KJA (Krasnoyarsk)
Start visiting UUD (Ulan-ude)
Start visiting SVO (Moscow)
Start visiting SSH (Sharm El Sheikh)
Start visiting SVX (Yekaterinburg)
Start visiting NYM (Nadym)
Start visiting UFA (Ufa)
Start visiting NOJ (Noyabrsk)
Start visiting TJM (Tyumen)
Start visiting SLY (Salekhard)
Start visiting TQL (Tarko-Sale)
Finish visiting TQL (Tarko-Sale)
```

Development: Strongly Connected Component

STRONGLY-CONNECTED-COMPONENTS(G)

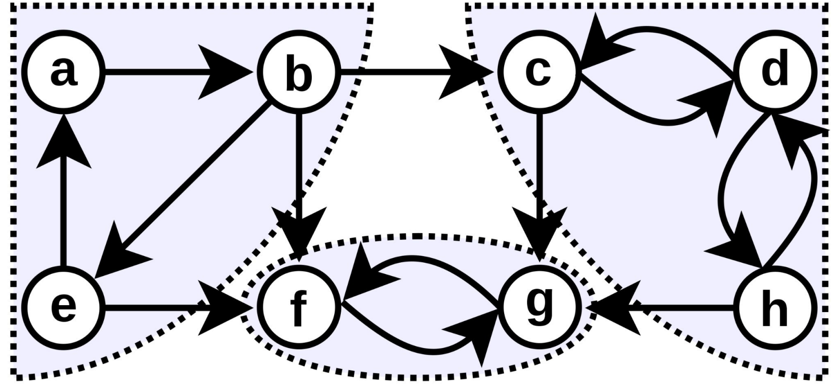
```
1  call DFS(G) to compute finishing times u.f for each vertex u
2  compute G^T
3  call DFS(G^T), but in the main loop of DFS, consider the vertices
       in order of decreasing u.f (as computed in line 1)
4  output the vertices of each tree in the depth-first forest formed in line 3 as a
       separate strongly connected component
```

(Reference: CLRS - Introduction to Algorithm P. 617)

Development: Strongly Connected Component

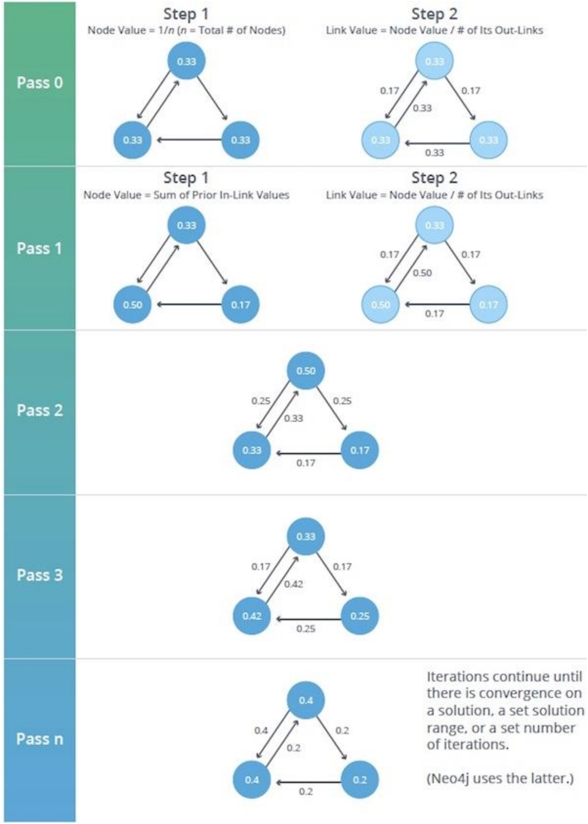
- Testcase

```
SECTION("all connected but not strongly connected 1") {
    AdjList graph {
        { 1 },// 0
        { 4, 5 },// 1
        { 3, 6 },// 2
        { 7, 2 },// 3
        { 0, 5 },// 4
        { 6 },// 5
        { 5 },// 6
        { 6, 3 },// 7
    };
    std::list<std::list<size_t> > result = StronglyConnectedComponents(graph);
    std::list<std::list<size_t> > expect = { { 0, 1, 4 }, { 2, 3, 7 }, { 5, 6 } };
    CheckPartition(result.cbegin(), result.cend(), expect.cbegin(), expect.cend());
}
```



Development: PageRank

- Probability matrix and vector to represent airports' importance
- Implementation
  - Iterative Simulation
  - Solve by eigenvector
    - LU Decomposition
    - Gaussian Elimination



Development: PageRank

- Fixed number of iteration
- O(V+E)

```
it5000:
real    0m4.308s
user    0m4.260s
sys     0m0.025s
```

```
void PageRank(const AdjList& graph, const std::vector<double>& curr_importance,
    std::vector<double>& next_importance) {
    next_importance.clear();
    next_importance.resize(graph.size(), 0);
    for (size_t u = 0; u < graph.size(); ++u) {
        double out_importance = curr_importance[u] / graph[u].size();
        for (size_t v : graph[u]) {
            next_importance[v] += out_importance;
        }
    }
}

std::vector<double> ImportanceIteration(const AdjList& graph, unsigned iteration_times) {
    double init_importance = static_cast<double>(1) / graph.size();
    std::vector<double> importance_1(graph.size(), init_importance), importance_2;
    unsigned iteration_times_half = iteration_times >> 1;
    for (unsigned i = 0; i < iteration_times_half; ++i) {
        PageRank(graph, importance_1, importance_2);
        PageRank(graph, importance_2, importance_1);
    }
    if (iteration_times & 1) {
        PageRank(graph, importance_1, importance_2);
        return importance_2;
    }
    return importance_1;
}
```

Development: PageRank

- The theorem is derived from math 257 lecture notes
- https://learn.illinois.edu/pluginfile.php/8179968/mod_resource/content/36/CompleteLectureNotes--Filled.pdf

Theorem 65. *Let A be an $n \times n$-Markov matrix with only positive entries and let $\mathbf{z} \in \mathbb{R}^n$ be a probability vector. Then*

$$\mathbf{z}_\infty := \lim_{k \to \infty} A^k \mathbf{z} \text{ exists,}$$

and $\mathbf{z}_\infty$ is a stationary probability vector of A (ie. $A\mathbf{z}_\infty = \mathbf{z}_\infty$).

Proof.

For simplicity, assume that A has n different eigenvalues $\lambda_1, \lambda_2, \dots, \lambda_n$.

Since A is a Markov matrix and has only positive entries, we can assume that $\lambda_1 = 1$ and $|\lambda_i| < 1$ for all $i = 2, \dots, n$.

Let $\mathbf{v_1}, \mathbf{v_2}, \dots, \mathbf{v_n} \in \mathbb{R}^n$ such that $A\mathbf{v}_i = \lambda_i \mathbf{v_i}$.

Since eigenvectors to different eigenvalues are linear independent, the above eigenvectors form a basis of $\mathbb{R}^n$.

Thus there are scalar $c_1, \dots, c_n$ such that $\mathbf{z} = c_1\mathbf{v}_1 + \cdots + c_n\mathbf{v}_n$.

$$\rightsquigarrow A^k\mathbf{z} = c_1\lambda_1^k\mathbf{v_1} + \cdots + c_n\lambda_n^k\mathbf{v_n} \to c_1\mathbf{v_1},$$

because $|\lambda_i| < 1$ for each $i = 2, \dots, n$. Why is $c_1 \neq 0$? Because $A^k\mathbf{z}$ is a probability vector.

Development: PageRank

- LU Decomposition
- O(V^3)

```
lu:
real    3m18.756s
user    3m17.737s
sys     0m0.238s
```

```cpp
template <typename T>
std::vector<T> FindOneDimNullSpaceByLU(Matrix<T>& mat_a) {
    size_t n = mat_a.size();
    Matrix<T> mat_l, mat_u;
    std::tie(mat_l, mat_u) = LUDecomposition(mat_a);
    std::vector<T> vec_x(n, 0);
    vec_x[n - 1] = 1;
    for (size_t i = n - 2; i < n; --i) {
        for (size_t j = i + 1; j < n; ++j) {
            vec_x[i] -= mat_u[i][j] * vec_x[j];
        }
        vec_x[i] /= mat_u[i][i];
    }
    return vec_x;
}
```

Development: PageRank

- Gaussian Elimination
- O(V^3)

```
gaussian:
real	9m18.171s
user	9m16.281s
sys	0m0.183s
```

```cpp
template <typename T>
std::vector<T> FindOneDimNullSpaceByGaussian(Matrix<T>& matrix) {
    size_t n = matrix.size();
    size_t row = 0;
    for (size_t col = 0; col < n - 1; ++col) {
        RowScale(matrix[row], 1.0 / matrix[row][col]);
        // Elimate the value at this column
        for (size_t y = 0; y < n; ++y) {
            if (y != row) {
                double factor = matrix[y][col] / matrix[row][col];
                RowEliminate(matrix[y], matrix[row],factor);
            }
        }
        ++row;
    }
    std::vector<double> solution;
    for(size_t row = 0; row < n - 1; ++row) {
        solution.push_back(-matrix[row][n - 1]);
    }
    solution.push_back(1);
    return solution;
}
```

Development: Floyd-Warshall

- 2d matrix **distance** to store distance of all shortest path
- 2d matrix **next** to store the path
- Function **PathReconstruction** to reconstruct path
- O(V^3) Runtime
  - approximately 70 min to run full dataset

```
Matrix<size_t> FloydWarshall(Matrix<unsigned>& distance) {
    size_t n = distance.size();
    Matrix<size_t> next(n, std::vector<size_t>(n, kNoAirline));

    for (size_t i = 0; i < n; i++) {
        for (size_t j = 0; j < n; j++) {
            if (distance[i][j] != kNoAirline) {
                next[i][j] = j;
            }
        }
    }

    for (size_t k = 0; k < n; ++k) {
        for (size_t i = 0; i < n; ++i) {
            for (size_t j = 0; j < n; ++j) {
                // By the definition of kNoAirline,
                // this addition operation will not cause an overflow
                unsigned new_distance = distance[i][k] + distance[k][j];
                if (new_distance < distance[i][j]){
                    distance[i][j] = new_distance;
                    next[i][j] = next[i][k];
                }
            }
        }
    }
    return next;
}
```

Demo

```
---------------------------------------
 CS 225 Final Project: OpenFlights
 Team Members: tluo9-yanzhen4-yirongc3
 Algorithm Driver
---------------------------------------
You are using airport and airline dataset: data/airport.csv, data/route.csv
```

```
-----------------------------------
 CS 225 Final Project: OpenFlights
 Team Members: tluo9-yanzhen4-yirongc3
 Result Interpreter
-----------------------------------
You are using result package: sample_result.tar.gz
Initializing...
Initialization finished
Usage:
dfs <origin-iata-code>: run DFS with the specific origin airport <origin-iata-code>
scc <iata-code>: find the index (unique identifier) of the strongly connected compon
ent contains airport <iata-code>
sp <departure-iata-code> <destination-iata-code>: find the shortest path from the ai
rport <departure-iata-code> to the airport <destination-iata-code>
top <limit-number>: find the <limit-number> most important airports
rank <iata-code>: find importance of the airport <iata-code>
>
```

Conclusion: Overall Summary

- Successful implementation of our algorithms
- Passed all test cases
- Able to run on the complete dataset and obtain correct result

```
===============================================================================
All tests passed (138 assertions in 4 test cases)

===============================================================================
All tests passed (37 assertions in 1 test case)

===============================================================================
All tests passed (331 assertions in 1 test case)

===============================================================================
All tests passed (331 assertions in 1 test case)

===============================================================================
All tests passed (48 assertions in 2 test cases)

===============================================================================
All tests passed (282 assertions in 6 test cases)
```

```
[yirongc3@linux-ssh-03 cs225-final]$ ./bin/tests_importance_mutual_actual
diff_peak_lu_gaussian = 0.000000
diff_peak_lu_it = 0.000000
diff_peak_gaussian_it = 0.000000
===============================================================================
All tests passed (9573 assertions in 1 test case)
```

Conclusion: The Most Important Airport

- The most important airport - Frankfurt am Main airport
  - Align with reality
    - Roughly center of Europe
    - Major site for global corporate headquarters

```
> top 10
```

Order	Iter	LU	Gaussian	Code	City	Airport Name
1	0.00642	242.43300	242.43300	FRA	Frankfurt	Frankfurt am Main Airport
2	0.00630	237.78200	237.78200	CDG	Paris	Charles de Gaulle International Airport
3	0.00628	236.94300	236.94300	IST	Istanbul	Istanbul Airport
4	0.00624	235.36200	235.36200	AMS	Amsterdam	Amsterdam Airport Schiphol
5	0.00586	221.19100	221.19100	ATL	Atlanta	Hartsfield Jackson Atlanta International Airport
6	0.00550	207.44000	207.44000	PEK	Beijing	Beijing Capital International Airport
7	0.00547	206.59500	206.59500	ORD	Chicago	Chicago O'Hare International Airport
8	0.00513	193.71600	193.71600	DME	Moscow	Domodedovo International Airport
9	0.00512	193.29400	193.29400	MUC	Munich	Munich Airport
10	0.00496	187.14000	187.14000	DFW	Dallas-Fort Worth	Dallas Fort Worth International Airport

Conclusion: The Shortest Path

- Expected runtime is extremely long given the size of the dataset

```
0,91,124,155,371,337,22564,21835,21872,24220,25599,25988,2147383647,21473836
91,0,175,163,423,277,22616,21887,21924,24272,25651,26040,2147383647,21473836
124,175,0,278,464,213,22657,21928,21965,24313,25692,26081,2147383647,2147383
155,163,278,0,273,440,22466,21737,21774,24122,25501,25890,2147383647,2147383
371,423,464,273,0,665,22193,21464,21501,23849,25228,25617,2147383647,2147383
337,277,213,440,665,0,22858,22129,22166,24514,25893,26282,2147383647,2147383
22564,22616,22657,22466,22193,22858,0,729,730,3078,4493,4882,2147383647,2147
21835,21887,21928,21737,21464,22129,729,0,224,2544,3764,4153,2147383647,2147
21872,21924,21965,21774,21501,22166,730,224,0,2348,3988,4377,2147383647,2147
24220,24272,24313,24122,23849,24514,3078,2544,2348,0,6308,6697,2147383647,214
25599,25651,25692,25501,25228,25893,4493,3764,3988,6308,0,1305,2147383647,214
25988,26040,26081,25890,25617,26282,4882,4153,4377,6697,1305,0,2147383647,214
2147383647,2147383647,2147383647,2147383647,2147383647,2147383647,2147383647
2147383647,2147383647,2147383647,2147383647,2147383647,2147383647,2147383647
25379,25431,25472,25281,25008,25673,4273,3544,3768,6088,696,1085,2147383647,
```

```
> sp CMI ATL
Distance of the shortest path: 432
CMI -> ORD -> ATL
> sp CMI FRA
Distance of the shortest path: 10771
CMI -> ORD -> FRA
> sp CMI SAT
Distance of the shortest path: 1138
CMI -> DFW -> SAT
> sp CMI LAX
Distance of the shortest path: 3347
CMI -> DFW -> PSP -> LAX
> sp CMI IRP
Distance of the shortest path: 14348
CMI -> ORD -> CUN -> HAV -> LAD -> FIH -> FKI -> GOM -> BNC -> BUX -> IRP
> sp CMI FKI
Distance of the shortest path: 13203
CMI -> ORD -> CUN -> HAV -> LAD -> FIH -> FKI
```

Conclusion: What is Next

- Minimal spanning tree
- Visual Map
- PageRank using damping factor
- Publicize an online platform