Efficient Route Planning

SS 2012

Lecture 1, Wednesday April 25th, 2012

(Intro, Organizational, OSM Data, Road Graph)

Prof. Dr. Hannah Bast
Chair of Algorithms and Data Structures
Department of Computer Science
University of Freiburg

UNI FREIBURG

Overview of this lecture

UNI FREIBURG

- Introduction
  - Demos + what you will learn in this course
- Organizational
  - Style of the course
  - Course Systems: Wiki, Forum, Daphne, SVN, Jenkins, …
  - Exercises + Exam
- And then let's start
  - Modeling road networks as graphs
  - OpenStreetMap (OSM) data
  - Exercise Sheet #1: build a graph from the OSM data of Saarland and Baden-Württemberg

Demos + what you will learn

UNI FREIBURG

- Demos
  - Routing in road and transit networks on Google Maps
  - http://wiki.openstreetmap.org/wiki/OpenTripPlanner
  - At the end of the course you will be able to build something like this ... and maybe even sth better
- What you will learn in this course
  - How to model road and transit networks
  - Where to get data and making sense of it
  - Clever algorithms for route planning on these networks
  - How to build web applications around this

Style of this course

- What I will do
  - Explain graph models, data, and the various algorithms
  - Give implementation advice + provide code skeletons
- What you will do
  - Implement graph builders (data → graph)
  - Implement the various algorithms
  - Do experiments with these algorithms / graphs
  - Explore variations / new ideas
  - Some theoretical tasks ... but not too many
  - Maybe have a look at some of the relevant research papers

Course systems

UNI FREIBURG

- Various systems supporting this course
  - The course Wiki is the **hub page** with links to each of the following
  - Daphne is our course management system
  - There is an SVN repository for your submissions, in particular for your code
  - There is a forum for asking questions
  - All the course materials will be put online: the lecture slides, the exercise sheets, the lecture recordings, as well as any code we write in the lectures
  - We will also provide a continuous build system (Jenkins) that automatically checks the code you commit to our SVN

Exercises + Exam

UNI FREIBURG

- There will be one exercise sheet per week
  - Usually an implementation / experimentation task
  - You can work on the sheets alone or in groups of 2 people
  - Submit the code to our **SVN** → URL on your Daphne page
  - Follow our **Coding Standards** → next slide
  - You can get 20 points per exercise sheet
  - The exercise sheets are **key** to a real understanding
- Exam in the end
  - You need 50% of the points to be admitted
  - The date of the exam has not been fixed yet, stay tuned ...

Our Coding Standards

UNI FREIBURG

- Please follow these guidelines when writing code
  - Write your programs in C++ or in Java
  - Document each class and each non-trivial method
  - Your code must conform to our style checkers
  - Write a unit test for every non-trivial function
  - Use a standardized Makefile / build.xml file
  - You find a comprehensive example on https://daphne.informatik.uni-freiburg.de/CodingStandards
  - Check your submissions on our build system Jenkins
  - We will walk through an example at the end of the lecture

How much work is it / ECTS points

- ECTS points = working time
  - You get 6 ECTS points for this lecture
  - That is 6 x 30 = 180 hours of work for the whole course
    - 60 hours for the exam + preparation
    - 120 hours for the lectures + exercise sheets
    - There are 12 lectures with exercise sheets
    - That's about 10 hours per week for this course
    - That's about 8 hours per exercise sheet
  - **Note:** when you have done all the exercise sheets (yourself) you are pretty much fit for the exam, without much more preparation needed

Road networks

- Model as graph
  - each crossing of two or road segments is a node in the graph
  - each road segment is a directed arc in the graph
  - in the simplest model, the cost of an arc is the time to travel along the corresponding road segment

OpenStreetMap

- OpenStreetMap (OSM)
  - Is an open-source initiative for gathering geo data
    - not only road network data, but also all kinds of other map data (e.g. where is a forest / river / building)
  - Started in 2004, very good coverage by now
    - 1.4 billion nodes, many 100 billions of arcs (April 2012)
  - Data can be downloaded for free, see link in References
  - Please use the data linked to from the Wiki, so that we all have exactly the same data sets
    - Saarland, version 24-Apr-2012 18:52
    - Baden-Württemberg, version 24-Apr-2012 19:44

Extracting road data from OSM files

UNI FREIBURG

- The OSM files contain not only road data
  - But all kinds of other map data, too
    - e.g. where is a forest / river / building
  - For now (in particular for Ex Sheet 1) all we need is
    - nodes (each with a latitude and a longitude)
    - ways (several arcs together) pertaining to roads
      - that is, with <tag k="highway" ...>
      - ways can also delineate forest boundaries etc.

Travel time along an arc

- The OSM data provides
  - ... node coordinates and road types
  - See the link to the OSM Map Features in the References
  - See our Wiki for a translation: road type → speed
  - This gives us travel time via the formula

    speed = distance traveled / travel time (v = s / t)
  - The obvious formula for the distance between nodes is the Euclidean distance between the corresponding points in 2D
  - More precisely, however, the path between two points on the earth's surface is **not** a straight line, but follows a so-called great circle (Großkreis) http://en.wikipedia.org/wiki/Great_circle
  - But for us here, Euclidean distance is good enough

Graph representation

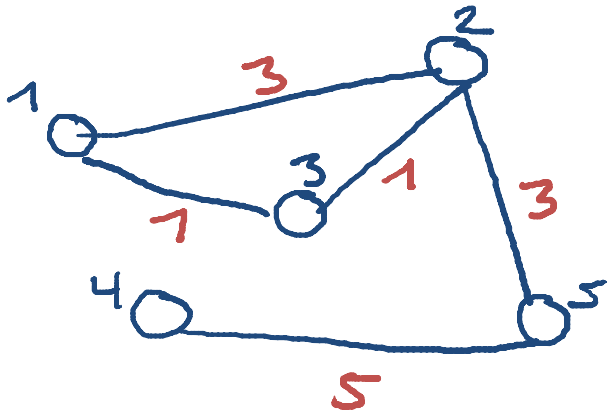
- Adjacency matrix
  - Store the arc costs in an $n \times n$ matrix, where n = #nodes
    - if arc does not exist, put some special value, e.g. ∞
  - Needs space $\Theta(n^2)$
  - Ok when m = #arcs is very large (so-called dense graphs)
- Adjacency lists
  - For each node, store an array of the outgoing arcs + their costs
  - Needs space $\Theta(n + m)$
  - Method of choice when $m << n^2$ (so-called sparse graphs)
  - For road networks, average degree of a node is around 2.5

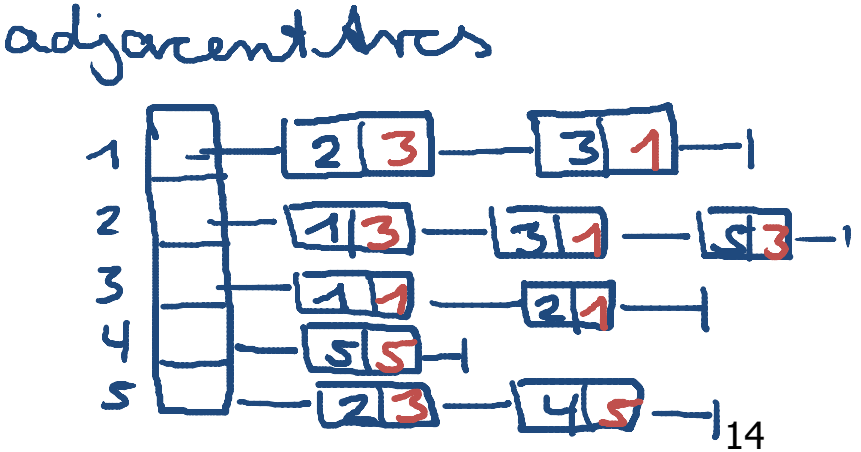
Adjacency lists

- Implementation advice
  - The straightforward implementation is

    Array<Array<Arc>> adjacentArcs;

    where Array could be vector in C++ and ArrayList in Java
  - The Arc class simply contains a data member for
    - the head node of each arc (the tail node is already implicit in the position of the Arc object in the Array)
    - the cost of the arc (in our case, the travel time)





Unit Tests, why and how

UNI FREIBURG

- Correctness check for each (non-trivial) method
  - Otherwise 1: debugging becomes a nightmare
  - Otherwise 2: no trust in your experimental results
  - Let's look at some unit tests together ...
- Problem: testing equality of complex objects
  - For example, a whole road network object
  - Simple solution: for each class provide a method DebugString which outputs the object in a simple human-readable form
  - Then your test can check simple string equality, e.g.

```
rn.readFromOsmFile("test-file.osm");
ASSERT_EQ("[3,2,{(1,2)},{2,3},{3,1}]", rn.asString());
```

References

- OpenStreetMap (OSM)
  - http://www.openstreetmap.org/
  - http://en.wikipedia.org/wiki/OpenStreetMap
  - http://wiki.openstreetmap.org/wiki/Stats
  - http://wiki.openstreetmap.org/wiki/XML
  - http://wiki.openstreetmap.org/wiki/Data_Primitives
  - http://wiki.openstreetmap.org/wiki/Map_Features#Highway
- Download of OSM extracts by country / region
  - http://download.geofabrik.de/osm
  - http://download.geofabrik.de/osm/europe/germany