MAC323 Algoritmos e estruturas de dados II

Exercício-Programa 4 (Enunciado basicamente completo)

Um sistema de navegação primitivo

- Neste EP, você produzirá um sistema de navegação simples (um "GPS").
- Os dados geográficos que você usará vêm do projeto OpenStreetMap (OSM). Leia um pouco sobre esse projeto na entrada correspondente da Wikipedia. Naturalmente, veja também a página da própria OSM.
- O produto final que você deve produzir é um sistema que, dados dois pontos *s* e *t*, encontra um caminho mais curto de *s* para *t*.

Como proceder

Arquivos XML de mapas OSM

- Se você acessar a URL

  http://www.openstreetmap.org/export?bbox=-46.7425,-23.5651,-46.7207,-23.5523,

  e usar a função "export", você pode obter um arquivo XML com as informações contidas nesse mapa, como esse arquivo (seu arquivo pode ser levemente diferente).

- O arquivo XML do exemplo acima pode ser processado para se extrair os pontos de referência (`nodes`), com informações geográficas (isto é, latitude e longitude desses pontos de referência). Você terá de ler um pouco sobre o formato desses arquivos XML gerados pelo OSM; veja, por exemplo,

  http://wiki.openstreetmap.org/wiki/OSM_XML.

  Extraindo os nós com suas localizações geométricas, obtemos, a partir de nosso arquivo XML acima, esse arquivo (nesse arquivo, cada linha tem o formato `<node id> <latitude> <longitude>`). Usando uma variante de `PlotFilter.java` de S&W, você pode produzir imagens como essa:

- Boas formas de se explorar o conteúdo de mapas OSM são descritas em

http://wiki.openstreetmap.org/wiki/Browsing

- Você terá de calcular as distâncias entre `nodes` em um mapa. Para tanto, você precisará saber como determinar a distância entre dois pontos dados pelas suas latitudes e longitudes. Aqui, vamos supor que a terra é uma esfera perfeita, com raio de 6371 km. Para esta parte do EP, veja

  http://www.movable-type.co.uk/scripts/latlong.html

  Experimente usar a interface da página acima para calcular a distância entre os seguintes dois pontos:

  http://www.openstreetmap.org/?&mlat=-23.55727&mlon=-46.73398#map=17/-23.56/-46.73

  http://www.openstreetmap.org/?&mlat=-23.5633&mlon=-46.7216#map=17/-23.56/-46.73

  Os pontos acima tem latitudes e longitudes dadas em `mlat` e `mlon`.

  - **Observação.** Na verdade, supor que a terra é plana já seria suficiente neste EP. *Você pode fazer isso.*

Grafos dirigidos a partir de arquivos XML de OSM

- A organização das ruas estão codificadas em mapas OSM. Tal informação é exportada nos arquivos XML correspondentes.
- *Para simplificar este EP, vamos usar um programa já pronto para extrair essa informação dos arquivos XML.* Tal programa está escrito em `Python` e usa `NetworkX` (que é, aliás, um sistema que pode ser de seu interesse).
- Instale em seu sistema `NetworkX`:

  http://networkx.github.io

  Você precisará também de um programa chamado `gistfile1.py`. Use essa versão, que é uma versão levemente alterada da versão original:

  https://gist.github.com/aflaxman/287370/

- Para facilitar o uso de `gistfile1.py`, use também `xmltoadj.py`. Eis um exemplo de uso:

  ```
  $ python xmltoadj.py map.osm-USP.xml USP.adjlist
  ```

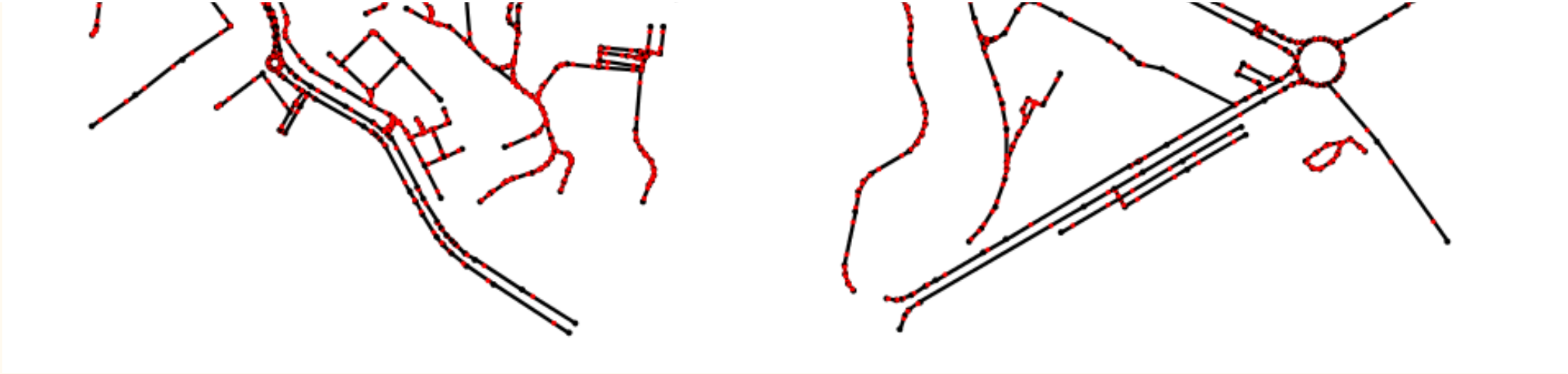
  O script `xmltoadj.py` lê o arquivo `map.osm-USP.xml` e produz o arquivo `USP.adjlist`.

- O arquivo `USP.adjlist` tem um formato natural: por exemplo, uma linha da forma `a b c` significa que o vértice `a` manda arcos para `b` e `c`. Veja

  http://networkx.github.io/documentation/latest/reference/readwrite.adjlist.html#format

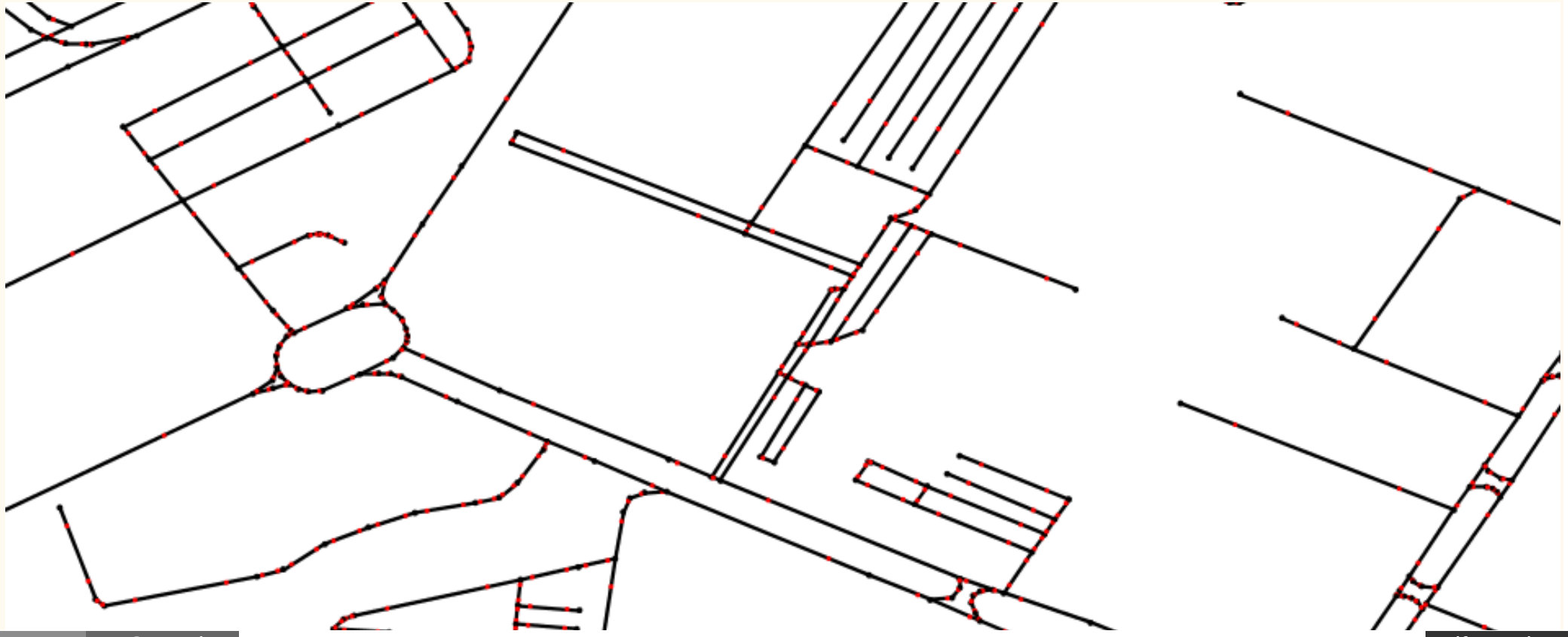
  Note que os nomes dos vértices que aparecem em `USP.adjlist` são os `id` dos `nodes` no arquivo XML (entretanto, nem todo `node` no arquivo XML ocorre no grafo).

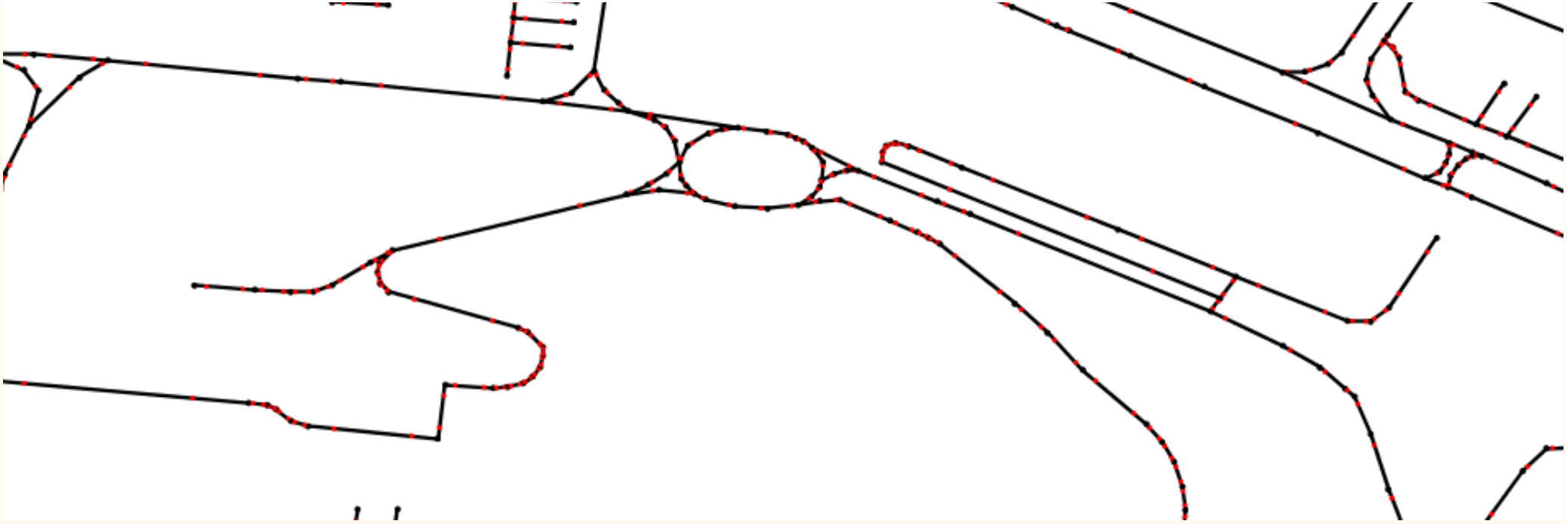
- O arquivo `USP.adjlist` pode ser lido usando-se `SymbolDigraph.java` (na verdade, o cabeçalho no arquivo `USP.adjlist` precisa ser removido).
- Eis uma figura do grafo em `USP.adjlist`:



Esse grafo tem 1271 vértices e 1926 arcos.

- Fazendo um *zoom* em uma região menor, podemos ver o grafo melhor:





Os vértices do grafo (que são `nodes` no mapa OSM) são representados por pontos pretos. Os arcos do grafo são representados por segmentos de reta. Os pontos vermelhos indicam a mão das ruas/orientação dos arcos: em um arco de `v` para `w`, há um ponto vermelho perto de `w` e este arco representa uma via com mão única, de `v` para `w`. Dois pontos vermelhos no segmento entre `v` e `w` indicam que a via é de mão dupla.

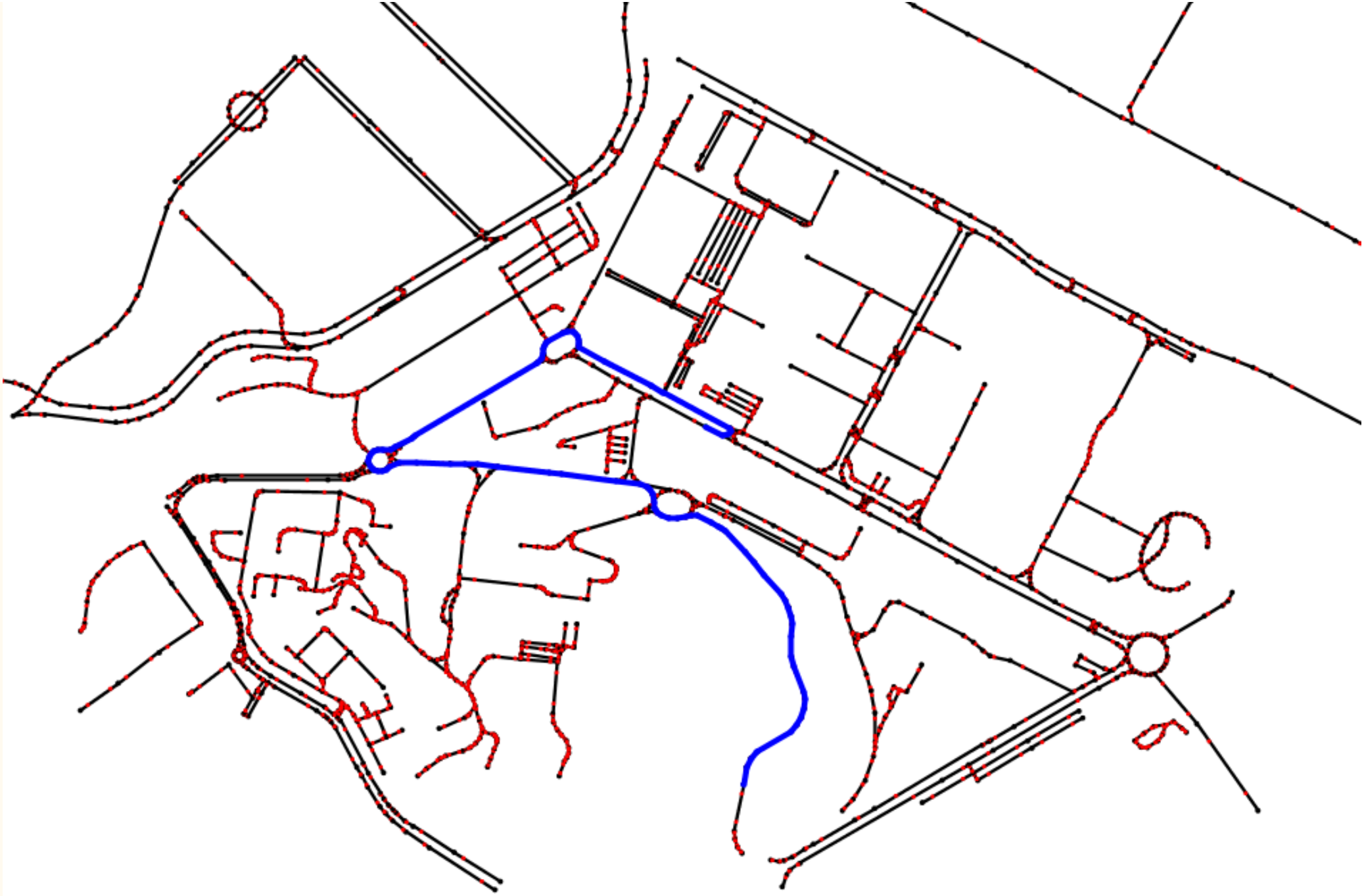
Caminhos mais curtos

- Suponha agora que queremos ir do `node 1931475238` ao `node 1831379092`, de carro, por um caminho curto. Você pode ver quais são esses nodes usando as URLs

  http://www.openstreetmap.org/node/1931475238

  http://www.openstreetmap.org/node/1831379092

  Seu sistema deve então executar o algoritmo de Dijkstra no grafo apropriado para encontrar um caminho mais curto. O resultado seria assim:



O caminho encontrado tem aproximadamente 2.7 km.

- Seu sistema deverá ser tal que o usuário deverá poder dar *pares latitude/longitude para especificar a origem e o destino*. Para tanto, seu programa deverá encontrar os vértices do grafo mais próximos dos pontos dados pelo usuário, e usar tais vértices como o par origem/destino.
- O usuário deverá poder dar vários pares origem/destino (como pares latitude/longitude).
- Seu programa deverá ter uma saída gráfica, exibindo o caminho encontrado para cada par origem/destino dado pelo usuário. Seu programa deverá também dizer o comprimento de cada caminho encontrado.

Outro exemplo

- Você pode obter um mapa para uma parte da cidade de São Paulo usando a URL

  http://www.openstreetmap.org/export?&bbox=-46.7357,-23.606,-46.5613,-23.5036

  O arquivo XML correspondente é este. O grafo correspondente tem 62327 vértices e 98222 arcos.

- Ao se procurar um caminho do IME à Praça da Sé, seu sistema poderia devolver um caminho como esse:

Esse caminho tem aproximadamente 12km de comprimento. O grafo em que a busca foi feita tem 62327 vértices e 98222 arcos.

No caso de mapas grandes, você pode optar por mostrar apenas os vértices do grafo, para a imagem não ficar muito poluída (essa opção poderia ser dada ao usuário).

Requisitos

Delineamos aqui como deve ser seu EP do ponto de vista do usuário e de implementação.

- *Forma de uso*
  - O usuário determinará o mapa a ser usado e produzirá o arquivo XML correspondente (digamos, `map-osm.xml`), usando o OSM.
  - O usuário executará o *script* `xmltoadj.py` para produzir o arquivo com as listas de adjacência do grafo dirigido correspondente (digamos, `G.adjlist`).
  - O usuário então executará seu programa, digamos `EP4`, fornecendo como entrada o arquivo XML do mapa e o arquivo com as listas de adjacência (arquivos `map-osm,xml` e `G.adjlist`). Seu programa deve então entrar em um modo interativo. Nesse modo, o usuário deverá ser capaz de fazer várias coisas:
    - O usuário deverá ser capaz de definir a região do mapa a ser desenhado nas figuras, dando dois pontos: o ponto inferior esquerdo e o ponto superior direito. Esses pontos devem ser pares latitude/longitude.
      - "Desenhar o mapa" significa desenhar o grafo dirigido da região (o mais fácil é simplesmente ignorar os pontos que

caem fora do "canvas").
    - Em qualquer momento, o usuário deve ser capaz de pedir que a figura seja atualizada. O usuário deverá ser capaz de dizer se ele quer que sejam desenhadas as arestas do grafo ou não (os vértices devem ser sempre desenhados). Note que, como o usuário pode especificar a região do mapa a ser desenhada, ele poderá fazer *zoom ins* e *zoom outs* na figura do grafo.
    - O usuário deverá ser capaz de pedir um caminho de comprimento mínimo entre um par de pontos (origem e destino), também dados por pares latitude/longitude. Seu programa deve encontrar os vértices do grafo mais próximos dos pontos dados, e deve encontrar um caminho mínimo entre eles (ou dizer que o destino não é acessível dessa origem). Uma vez encontrado um caminho mínimo, ele deve ser indicado com alguma cor diferente na figura atual (os vértices e as arestas devem ser dessa cor). Seu programa deve também dizer o comprimento do caminho encontrado.
    - *O usuário deverá ser capaz de dar os pontos de origem e destino com o mouse.* Para tanto, o usuário deverá ter um modo de alternar entre interação via mouse e via teclado.
    - O usuário deverá ser capaz de "limpar" a figura, removendo-se o caminho mínimo atual (o mais fácil é redesenhar a figura).
- *Implementação*. É natural decompor seu sistema em várias classes. As seguintes classes são naturais. Se você encontrar outra decomposição melhor, você pode usá-la. Se você não usar uma boa decomposição, sua nota pode sofrer reduções.
    - `EdgeWeightedDigraph.java`. A classe de S&W para grafos dirigidos com pesos nos arcos.
    - `SymbolEWDigraph.java`. Deve ter a mesma relação com a classe `EdgeWeightedDigraph.java` como as classes `SymbolDigraph.java` e `Digraph.java` tem entre si.
    - `Location.java`. Objetos dessa classe especificam um ponto na superfície da terra. Você poderia implementar como pares latitude/longitude.
    - `GeoInto.java`. Um objeto dessa classe deve ter, como componente principal, uma tabela de símbolos com elementos da forma `<node id, location>`, onde `node id` é o identificador de um nó de um mapa OSM e `location` é a localização desse nó.
    - `SymbolGeoEWDigraph.java`. Um objeto dessa classe deve ter, como elementos principais, um `SymbolEWDigraph` e um `GeoInfo`, este último contendo a localização geográfica dos vértices no `SymbolEWDigraph`.

Vocês podem fazer este EP em duplas

- Cada dupla deve entregar um único trabalho. Um membro de cada par deve entregar o trabalho no Paca (*não esqueçam de colocar os nomes de ambos os integrantes do par no trabalho*). O outro membro da equipe deve entregar um texto, dizendo quem é seu parceiro.

Alterações, correções e atualizações

Como estou divulgando versões parciais do enunciado, vou manter um *log* das alterações, correções e atualizações mais importantes.

- [2015-06-23 Tue 06:50] Requisito sobre interação via mouse adicionado.
- [2015-06-21 Sun 19:06] Requisitos do EP definidas. Enunciado próximo de completo.
- [2015-06-21 Sun 18:00] Exemplo com mapa de parte da cidade de São Paulo adicionada.

- [2015-06-20 Sat 16:07] Exemplo de caminho mínimo adicionado. Especificação da saída do EP adicionada.
- [2015-06-20 Sat 14:50] `gistfile1.py` refinado, para levar em conta oneway tags que são -1. Grafos do enunciado gerados novamente de acordo.
- [2015-06-20 Sat 13:31] Os grafos nos enunciado foram gerados novamente, com a nova versão de `gistfile1.py`.
- [2015-06-20 Sat 13:30] O script `gistfile1.py` foi refinado, para levar em conta a mão das rotatórias.
- [2015-06-20 Sat 13:30] O script `gistfile1.py` foi refinado, para não incluir passagens de pedestres, ciclovias, etc.

Author: Yoshiharu Kohayakawa
Email: yoshi@ime.usp.br
Created: 2015-06-23 Tue 06:51
Emacs 24.4.51.2 (Org mode 8.2.10)
Validate