

# Universidade Estadual do Sudoeste da Bahia

## Algoritmos e Estrutura de Dados II

### Trabalho da I Unidade

## 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=-40.0772,-13.8651,-40.0584,-13.8802> usar a função "export", você pode obter um arquivo XML com as informações contidas nesse mapa.
- 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](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 `plot_pontos.py`, 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. Para este trabalho veja

a distância Haversine: [https://pt.wikipedia.org/wiki/Fórmula de Haversine](https://pt.wikipedia.org/wiki/F%C3%B3rmula_de_Haversine)

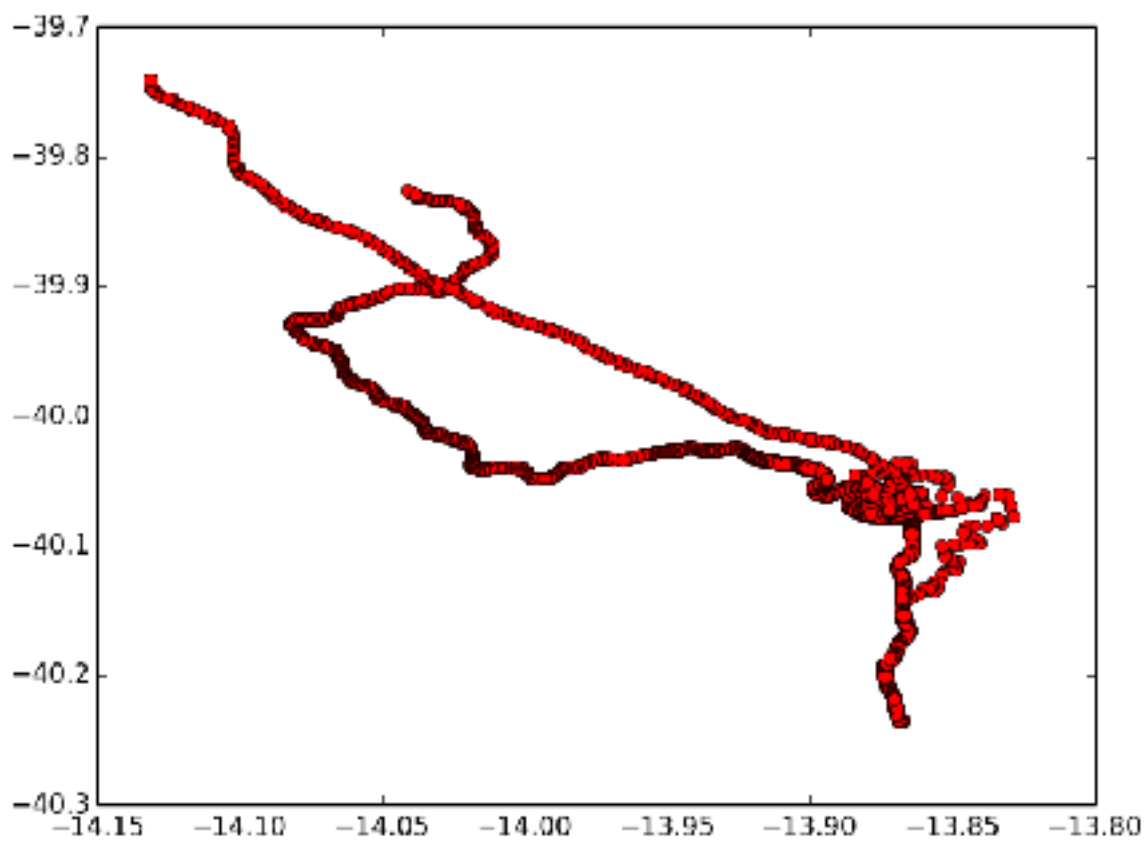
- Pode ser utilizado o algoritmo pronto como no <https://pypi.org/project/haversine/>

- 

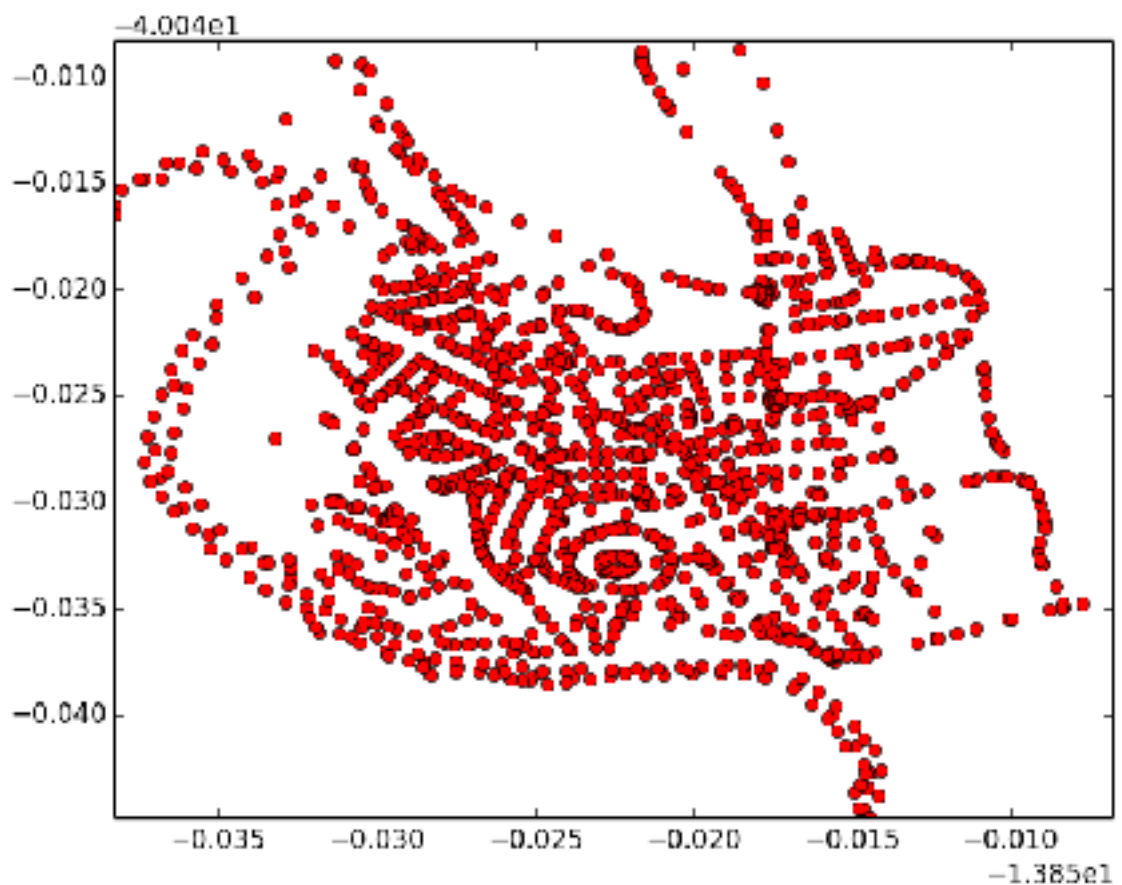
## **Grafos dirigidos a partir de arquivos XML de OSM**

- A organização das ruas estão codificadas em mapas OSM. Tal informação é exportada nos arquivos XML correspondentes.
- *Para simplificar este Trabalho, 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 a versão disponibilizada, 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.xml UESB.adjlist
```
- O script `xmltoadj.py` lê o arquivo `map.osm.xml` e produz o arquivo `UESB.adjlist`.
- O arquivo `UESB.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 <https://networkx.github.io/documentation/latest/reference/readwrite/adjlist.html?highlight=adj#module-networkx.readwrite.adjlist>  
Note que os nomes dos vértices que aparecem em `UESB.adjlist` são os id dos nodes no arquivo XML (entretanto, nem todo node no arquivo XML ocorre no grafo).
- O arquivo `UESB.adjlist` pode ser lido usando-se `plot_pontos.py`
- Eis uma figura do grafo em `UESB.adjlist`:



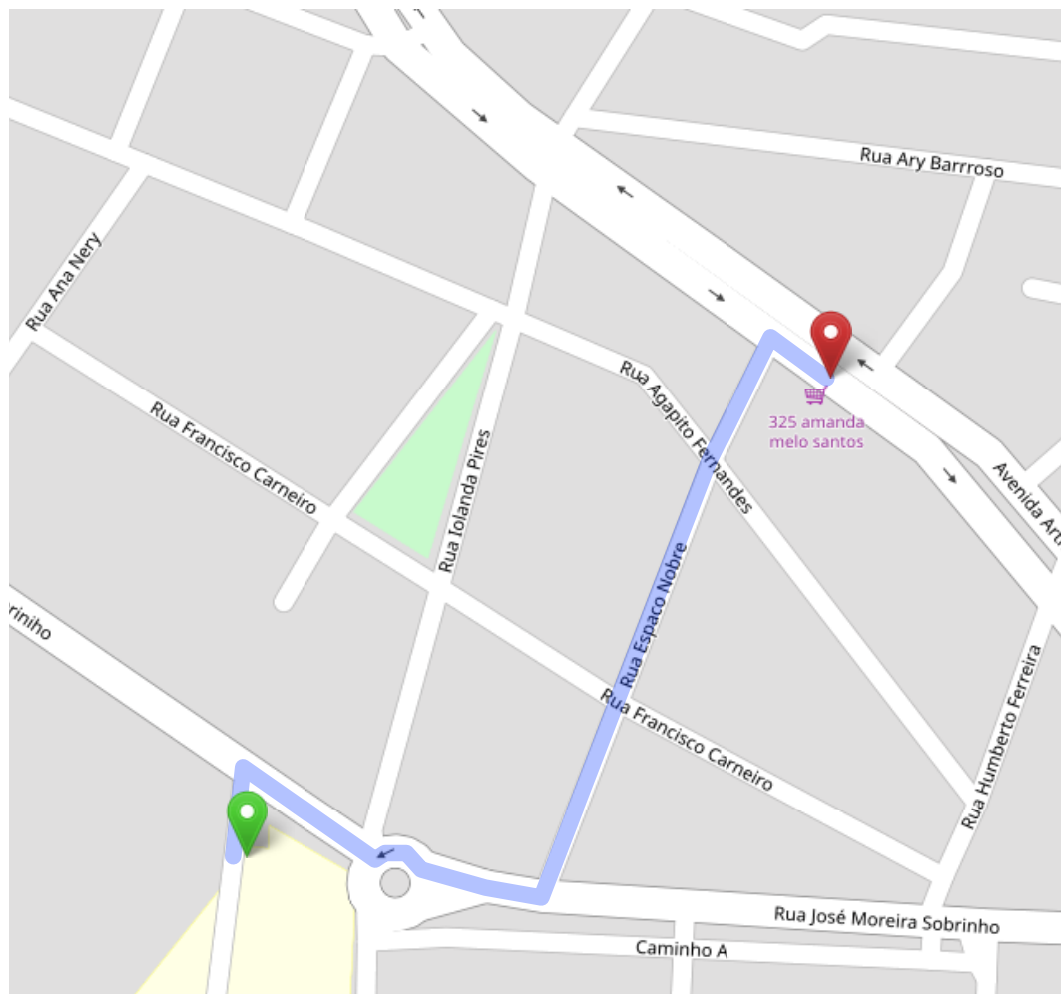
- Fazendo um *zoom* em uma região menor, podemos ver o grafo melhor:



- Os vértices do grafo (que são nodes no mapa OSM) são representados por pontos vermelhos. 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 2271245951 ao node 2323108861, por um caminho curto. Você pode ver quais são esses nodes usando as URLs <https://www.openstreetmap.org/node/2271245951>  
<https://www.openstreetmap.org/node/2323108861>  
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 388m.



- 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.
- 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.

## Requisitos

Delineamos aqui como deve ser seu Trabalho 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 `rota.py`, 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 dar os pontos de origem e destino com o mouse.*
- O usuário deverá ser capaz de "limpar" a figura, removendo-se o caminho mínimo atual (o mais fácil é redesenhar a figura).

### **Vocês podem fazer este Trabalho em duplas**

- Cada dupla deve entregar um único trabalho. Um membro de cada par deve entregar o trabalho no classroom (*não esqueçam de colocar os nomes de ambos os integrantes do par no trabalho*).
- Teremos reuniões semanais para planejamento e execução das atividades.