Semester work in csharp at Charles University Prague

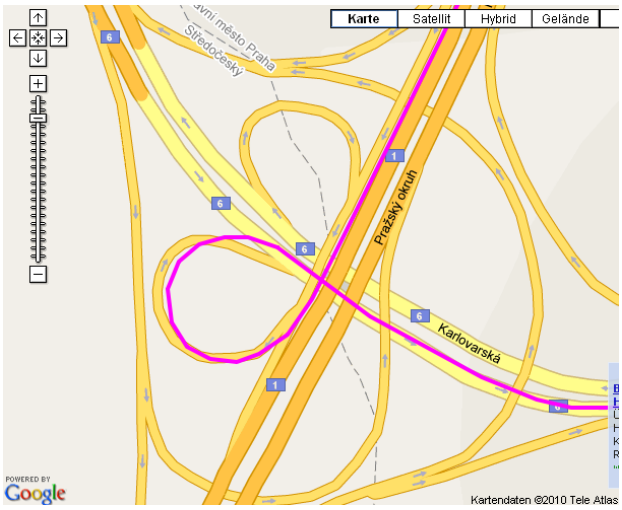
FINDING THE SHORTEST ROUTE USING DIJKSTRA ALGORITHM

Pavel Zbytovský <pavel@zby.cz>, 2010

Obsah

Program má za úkol vyhledat nejkratší cesty na mapě z projektu OpenStreetMap. Uživatelsky se zadají id dvou nodů a soubor s OSM xml daty. Výstup se uloží do GPX a KML pro zobrazení v Google Maps.

Příklad:
start: www.osm.org/?node=21312439
target: www.osm.org/?node=34593695
výstup na mapě:
http://www.gpswandern.de/gpxviewer/gpxshow.shtml?url=http://upload.zby.cz/output.gpx
... ukáže to prvně na Google Maps, takže cesta neodpovídá silnicím, když se nahoře přepne na OSM - už to je přesné :)



Uživatelská dokumentace

Příprava vstupních dat

Program čte OSM datové soubory, formát založený na XML je popsán na http://wiki.openstreetmap.org/wiki/.osm Tento obsahuje nody a ways. Můj routovací program nebude brát ohled na tagy a bude routovat po všech napojených hranách. Tedy je potřeba data nejprve zpracovat a zanechat pouze silnice, nebo pouze železnici, apod.

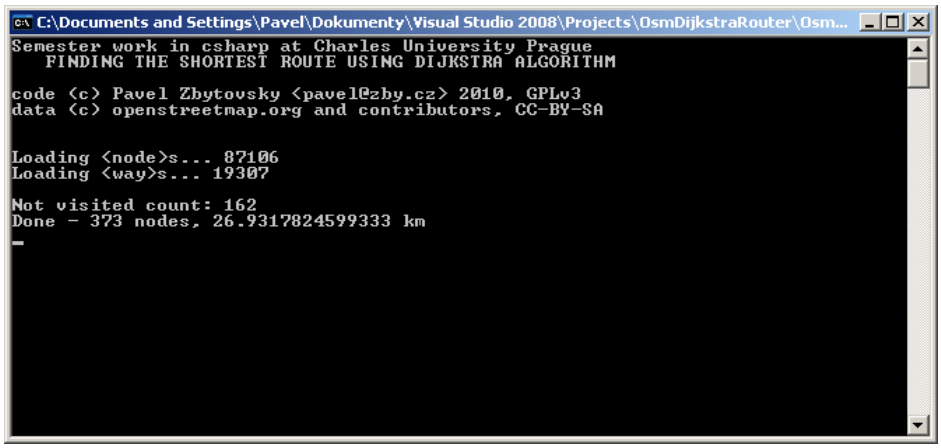
OSM datový soubor se získá z http://download.geofabrik.de/osm/europe/czech_republic.osm.bz2, pomocí programu osmosis lze vyříznout pouze cesty s tagem highway. Tento soubor má cca 300MB, na 1.7GHz stroji trvalo načtení zhruba půl minuty, dijkstra vteřinu. V RAM zabráno cca 200 MB.

```
$ bzcat czech*.osm.bz2 | osmosis \
        --read-xml enableDateParsing=no file=/dev/stdin \
        --way-key-value
keyValueList="highway.motorway,highway.motorway_link,highway.trunk,highway.trunk_link,highway.primary,highwa
y.primary_link,highway.secondary,highway.secondary_link,highway.tertiary,highway.unclassified,highway.road,highwa
y.residential" \
        --used-node \
        --write-xml file=cz_silnice.osm
```

Spuštění

Pro kompilaci byl použit MS Visual C# 2008 Express, výsledný spustitelný soubor očekává jako první parametr cestu k OSM souboru. Druhé dva nepovinné parametry jsou ID osm bodů, pokud se nezadají, je možno je napsat po výzvě. Získat jdou pomocí tlačítka edit na sránce openstreetmap.org nebo datové vrstvy „Data“, která ukaáže všechny prvky mapy v surových datech.

Výstup je ve formátu GPX a KML hodí se pro zobrazení ve službě maps.google.com apod.



Programátorská dokumentace

Algoritmus

1. kontrola argumentů a nastavení xmlreaderu
2. načítání <node>s ze vstupního xml souboru
   - připravit datovou strukturu Dictionary nodesById
   - vytváření nových Vertex, přidávání do *nodesById*
   - nalezení start/target
3. načítání <way>s ze vstupu
   - při procházení xml stromu ukládáme hodnoty ref do LinkedList<id>, zpracujeme případný tag oneway=yes/1/-1/ostatní
   - při naražení na </way> se projde LinkedList, z id se pomocí *nodesById* získá Vertex a ten se připojí do LinkedListů neighbors
4. spuštění dijikstrova algoritmu pomocí RoutingAlgorithm(strStartTarget) případně spuštění interaktivního módu.
5. Dijkstrův algoritmus, viz níže
6. výstup souřadnic nalezené cesty do GPX/KML, vypsání do konzole
   - pokud byl cíl dosažen, vede nejkratší cesta přes target.previous – tedy projití cesty cyklem ve zpětném směru

Datové struktury

```
Vertex start, target;
GPWiki.BinaryHeap<Vertex> notVisited;
Dictionary<long, Vertex> nodesById;
```

Vertex

```
class Vertex : IComparable<Vertex>, GPWiki.IBinaryHeapIndexed
{
   public double lat;
   public double lon;
   public long id;
   public LinkedList<Vertex> neighbors;
   public double distance;
   public Vertex previous;

   public Vertex(){this.neighbors = new LinkedList<Vertex>(); }

   #region IComparable Members
   #region IBinaryHeapIndexed Members
}
public interface IBinaryHeapIndexed { int BinaryHeapIndex { get; set; } }  //dopsaný do kódu GPWiki
```

System.Collections.Generic.Dictionary<long,Vertex>

Vestavěný datový typ, dle dokumentace [1] asymptotická složitost:

Operation	Dictionary<K,V>
this[key]	O(1)
Add(key,value)	O(1) or O(n) - when it becomes necessary to enlarge the hash table
Remove(key)	O(1)
ContainsKey(key)	O(1)
ContainsValue(value)	O(n)

GPWiki.BinaryHeap<Vertex>

Implementace generické binární haldy ze serveru gpwiki.org [2]. Asymptotická složitost:

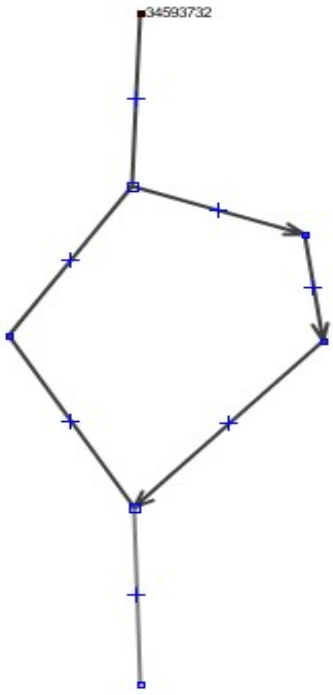
Operation	BinaryHeap<T>
Count	O(1) - vlastnost
Add(T)	viz UpHeap()
CustomUpHeap(T)	díky vlastnímu interface IBinaryHeapIndexed je vyhledání v poli pouze přístupem na index, a odtud již viz UpHeap()
UpHeap()	O(log(n)) dle dokumentace
T RemoveMin()	O(log(n)) dle dokumentace

System.Collections.Generic.LinkedList<Vertex>

First, Node.Next, AddLast()	O(1)
Clear()	O(n)

Vstup – maly-graf.osm

```
<?xml version='1.0' encoding='UTF-8'?>
<osm version='0.6' generator='JOSM'>
  <node id='60648429' visible='true' lat='0.46621061323240176' lon='4.372470117761364' />
  <node id='-13' visible='true' lat='-0.44810572330357706' lon='1.638544702515128 3' />
  <node id='-8' visible='true' lat='0.4662106132324606' lon='3.0507710759535813' />
  <node id='-6' visible='true' lat='1.4075584052145285' lon='2.4170797545388907' />
  <node id='-4' visible='true' lat='1.3894583321203549' lon='0.9958006479373708' />
  <node id='-2' visible='true' lat='0.4752630408529401' lon='0.1357909974460051' />
  <node id='34593732' visible='true' lat='0.4481057233035538' lon='-1.8558102984287361' />
  <way id='-19' visible='true'>
    <nd ref='-8' />
    <nd ref='-13' />
    <nd ref='-2' />
    <tag k='highway' v='residential' />
  </way>
  <way id='-17' visible='true'>
    <nd ref='-8' />
    <nd ref='60648429' />
  </way>
  <way id='-11' visible='true'>
    <nd ref='-2' />
    <nd ref='-4' />
    <nd ref='-6' />
    <nd ref='-8' />
    <tag k='highway' v='residential' />
    <tag k='oneway' v='yes' />
  </way>
  <way id='-3'  visible='true'>
    <nd ref='34593732' />
    <nd ref='-2' />
    <tag k='highway' v='residential' />
  </way>
</osm>
```



Vizualizace v programu JOSM
Tag oneway zobrazen šipkou

Dijkstrův algoritmus

```
//each Vertex in nodesById has previous=null and distance=MaxValue (start is in notVisited, distance=0)
while (notVisited.Count > 0)
{
   Vertex u = notVisited.Remove();
   if (u.distance == double.MaxValue)          // all remaining vertices are inaccessible from source
      break;

   if (u == target)               // target reached with shortest way
      break;

   foreach (Vertex v in u.neighbors)
   {
      double distThroughU = u.distance + Distance(u, v);

      if (v.distance == double.MaxValue)  //never been reached
      {
         v.distance = distThroughU;
         v.previous = u;
         notVisited.Add(v);
      }
      else if (distThroughU < v.distance)  //found shorter way, relaxing the edge
      {
         v.distance = distThroughU;
         v.previous = u;
         notVisited.CustomUpHeap(v);
      }
   }
}
```

Vhodné optimalizace

Možno implementovat algoritmu A*. <Vertex> třídit tedy podle double priority, která by obsahovala distance + Distance(v, target)

A* není problém udělat, ale Dijikstrův alg. je i na celorepublikových datech velice rychlý. Téměř zanedbatelně proti rychlosti načítání XML. V rychlosti by ještě pomohlo porovávání nějakých "nevzdálenosti", třeba kvadrátů, (ušetří se na odmocňování) - via efficiently measuring geographic distances.

Paměťové nároky – načítání dat

Upravený OSM soubor pro ČR má 300 MB, velikosti v RAM jsou měřeny Task Managerem.

	počet	RAM
csharp VM	-	40 MB
node	1,2 M	90 MB
ways	150 k	80 MB

Samotné routování navýší paměťové nároky jen minimálně.

Zdroje

[1] http://www.cs.aau.dk/~normark/oop-08/html/notes/collections_themes-dictionary-sect.html#collections_time-complexity-dictionaries_title_1
[2] http://gpwiki.org/index.php/C_sharp:BinaryHeapOfT

[JOSM] Editor souborů .osm i zobrazovač vygenerovaných GPX <http://josm.openstreetmap.de>