Rotinas para obtenção das estruturas auxiliares para o sistema de recomendação

Estas rotinas são usadas para a rápida recomendação de recursos do Participa.br para usuários e para linha editorial.

Está prevista a disponibilização destas estruturas já no endpoint SparQL através de inclusão das rotinas na triplificação dos dados. Por hora, o cálculo é feito pelo sistema de recomendação. Estas são as estruturas atualizadas com a visita ao caminho http://"dominio"/atualiza

In [88]:
```
from SPARQLWrapper import SPARQLWrapper, JSON
import networkx as x, nltk as k
import string, time, __builtin__
import cPickle as pickle
```

In [5]:
```
URL_ENDPOINT="http://localhost:82/"
URL_ENDPOINT_=URL_ENDPOINT+"participabr/query"
EXCLUDE=set(string.punctuation+u'\u201c'+u'\u2018'+u'\u201d'+u'\u2022'+u'\
STOPWORDS=set(k.corpus.stopwords.words('portuguese'))
PREFIX="""PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX ops: <http://purl.org/socialparticipation/ops#>
PREFIX opa: <http://purl.org/socialparticipation/opa#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX dc: <http://purl.org/dc/terms/>
PREFIX tsioc: <http://rdfs.org/sioc/types#>
PREFIX sioc: <http://rdfs.org/sioc/ns#>
PREFIX schema: <http://schema.org/>"""
stemmer = k.stem.RSLPStemmer()
```

Rede de amizades

In [6]:
```
q="""SELECT ?a ?b ?aname ?bname
       WHERE {
          ?a foaf:knows ?b .
       }"""
sparql=SPARQLWrapper(URL_ENDPOINT_)
sparql.setQuery(PREFIX+q)
sparql.setReturnFormat(JSON)
results = sparql.query().convert()
g=x.Graph()
for amizade in results["results"]["bindings"]:
    nome1=amizade["a"]["value"]
    nome2=amizade["b"]["value"]
    g.add_edge(nome1,nome2)
__builtin__.g=g
```

Rede de interação

In [9]:
```
q="""SELECT ?participante1 ?participante2 ?aname ?bname
       WHERE {
           ?comentario dc:type tsioc:Comment.
           ?participante1 ops:performsParticipation ?comentario.
           ?participante1 foaf:name ?aname.
           ?artigo sioc:has_reply ?comentario.
           ?participante2 ops:performsParticipation ?artigo.
           ?participante2 foaf:name ?bname.
       }"""
sparql=SPARQLWrapper(URL_ENDPOINT_)
sparql.setQuery(PREFIX+q)
sparql.setReturnFormat(JSON)
results = sparql.query().convert()
d=x.DiGraph()
for interacao in results["results"]["bindings"]:
    nome_chegada=interacao["participante1"]["value"]
    nome_partida=interacao["participante2"]["value"]
    if (nome_partida,nome_chegada) in d.edges():
        d[nome_partida][nome_chegada]["weight"]+=1
    else:
        d.add_edge(nome_partida,nome_chegada,weight=1.)
__builtin__.d=d
```

Bag-Of-Words com todos os comentarios e artigos do participa.br

In [39]:
```
q="SELECT ?comentario ?titulo ?texto WHERE \
          {?comentario dc:type tsioc:Comment.\
         OPTIONAL {?comentario dc:title ?titulo . }\
          OPTIONAL {?comentario schema:text ?texto .}}"
sparql=SPARQLWrapper(URL_ENDPOINT_)
sparql.setQuery(PREFIX+q)
sparql.setReturnFormat(JSON)
results = sparql.query().convert()
msgs_=results["results"]["bindings"]
msgs=[mm for mm in msgs_ if ("titulo" not in mm.keys()) or
      (("teste de stress" not in mm["titulo"]["value"].lower())
      and ("comunidade de desenvolvedores e nesse caso, quanto mais"
           not in mm["texto"]["value"].lower()))]
palavras=string.join([i["texto"]["value"].lower() for i in msgs])
palavras = ''.join(ch for ch in palavras if ch not in EXCLUDE)
#palavras = ''.join(ch for ch in palavras if ch not in EXCLUDE).encode('ut
palavras_=palavras.split()
palavras__=[stemmer.stem(pp) for pp in palavras_ if pp not in STOPWORDS]
fdist_=k.FreqDist(palavras__)
# escolhendo as 400 palavras mais incidentes para referência
palavras_escolhidas=fdist_.keys()[:400]
__builtin__.palavras_escolhidas=palavras_escolhidas
__builtin__.fdist_=fdist_
```

In [100]:
```
T=time.time()
q="SELECT ?cbody ?titulo ?abody WHERE \
          {?foo ops:performsParticipation ?participacao.\
         OPTIONAL { ?participacao schema:articleBody ?abody. }\
         OPTIONAL {?participacao dc:title ?titulo . }\
```

```
          OPTIONAL {?participacao schema:text ?cbody .}}"
sparql=SPARQLWrapper(URL_ENDPOINT_)
sparql.setQuery(PREFIX+q)
sparql.setReturnFormat(JSON)
results = sparql.query().convert()
msgs_=results["results"]["bindings"]
msgs=[mm for mm in msgs_ if ("titulo" not in mm.keys()) or
      (("teste de stress" not in mm["titulo"]["value"].lower())
      or ("cbody" not in mm.keys() or ("comunidade de desenvolvedores e ne
          not in mm["cbody"]["value"].lower()))))]
textos1=[i["cbody"]["value"] for i in msgs if "cbody" in i.keys()]
textos2=[i["abody"]["value"] for i in msgs if "abody" in i.keys()]
textos=textos1+textos2
# faz BoW e guarda num dict
texto=string.join(textos).lower()
texto_= ''.join(ch for ch in texto if ch not in EXCLUDE)

texto__=texto_.split()
#texto___=[stemmer.stem(pp) for pp in texto__]
texto___=[stemmer.stem(pp) for pp in texto__ if (pp not in STOPWORDS) and
bow=k.FreqDist(texto___)
radicais_escolhidos=bow.keys()[:400]
__builtin__.radicais_escolhidos=radicais_escolhidos
__builtin__.bow=bow
print("demorou %.2f segundos para fazer a BoW"%(time.time()-T,))
```

```
demorou 128.67 segundos para fazer a BoW
```

In [70]:

```
print(u"são %i textos postados e %i comentários,\n\
  totalizando %i palavras informativas em %i radicais"%
      (len(textos1), len(textos2),len(texto___), fdist.B()))
```

```
são 16253 textos postados e 1554 comentários,
  totalizando 2477698 palavras informativas em 18233 radicais
```

BoW de cada participante

In [105]:

```
T=time.time()
q="""SELECT DISTINCT ?participante
       WHERE {
          ?foo dc:contributor ?participante .
       }"""
sparql=SPARQLWrapper(URL_ENDPOINT_)
sparql.setQuery(PREFIX+q)
sparql.setReturnFormat(JSON)
results = sparql.query().convert()
participantes_=results["results"]["bindings"]
participantes=[i["participante"]["value"] for i in participantes_]
# inicia loop
if "palavras_escolhidas" not in dir(__builtin__):
    print(u"rode BoW antes, para saber do vocabulário geral do portal")
else:
    palavras_escolhidas=__builtin__.palavras_escolhidas
bows={}
for participante in participantes:
    # puxa todos os comentarios de cada usuario
    # e os article bodys
```

```
    q="""SELECT DISTINCT ?abody ?cbody
         WHERE {
           <%s> ops:performsParticipation ?participacao.
             OPTIONAL { ?participacao schema:articleBody ?abody. }
             OPTIONAL { ?participacao schema:text ?cbody. }
             OPTIONAL {?comentario dc:title ?titulo . }
         }"""%(participante,)
    sparql = SPARQLWrapper("http://localhost:82/participabr/query")
    sparql.setQuery(PREFIX+q)
    sparql.setReturnFormat(JSON)
    results = sparql.query().convert()
    results_=results["results"]["bindings"]
    results__=[mm for mm in results_ if ("titulo" not in mm.keys()) or
              (("teste de stress" not in mm["titulo"]["value"].lower()) or
              ("cbody" not in mm.keys() or
               ("comunidade de desenvolvedores e nesse caso, quanto mais"
                not in mm["cbody"]["value"].lower())))]

    textos1=[i["cbody"]["value"] for i in results__ if "cbody" in i.keys()
    textos2=[i["abody"]["value"] for i in results__ if "abody" in i.keys()
    textos=textos1+textos2
    # faz BoW e guarda num dict
    texto=string.join(textos).lower()
    texto_= ''.join(ch for ch in texto if ch not in EXCLUDE)
    texto__=texto_.split()
    texto___=[stemmer.stem(pp) for pp in texto__ if (pp not in STOPWORDS)
    fdist=k.FreqDist(texto___)
    ocorrencias=[fdist[i] for i in radicais_escolhidos]
    bows[participante]=(fdist,ocorrencias)
__builtin__.bows=bows
len(texto___)
print("%.2f segundos para fazer BoW de cada participante"%(time.time()-T,)
```

```
895.30 segundos para fazer BoW de cada participante
```

In [103]:

```
pickle.dump( g, open( "pickledir/g.p", "wb" ) )
pickle.dump( d, open( "pickledir/d.p", "wb" ) )
pickle.dump( bow, open( "pickledir/bow.p", "wb" ) )
pickle.dump( radicais_escolhidos, open( "pickledir/radicais_escolhidos.p",
pickle.dump( bows, open( "pickledir/bows.p", "wb" ) )
```