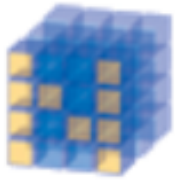


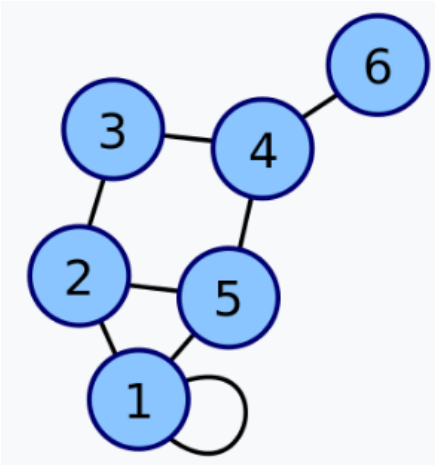
30.03.2017

Вычислительные модели с использованием научных библиотек Python

Алгоритмы для работы с графами

Способы представления графов

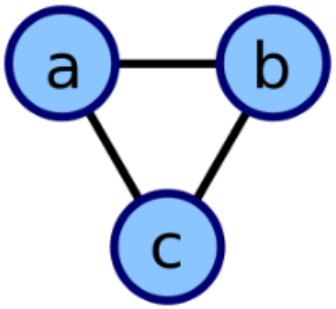
Матрица смежности



$$\begin{pmatrix} 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

Пример библиотеки: scipy.sparse.csgraph

Список смежности



[(b,c), (a,c), (a,b)]

Пример библиотеки: NetworkX

Библиотека NetworkX

```
>>> import networkx as nx
>>> G=nx.Graph()

>>> G.add_node(1)
>>> G.add_nodes_from([2,3])

>>> G.add_edge(1,2)
>>> e=(2,3)
>>> G.add_edge(*e) # unpack edge tuple*

>>> G.add_edges_from([(1,2),(1,3)])
```

```
>>> G.number_of_nodes()
8
 >>> G.number_of_edges()
 2
```

```
>>> G.nodes()
 ['a', 1, 2, 3, 'spam', 'm', 'p', 's']
>>> G.edges()
[(1, 2), (1, 3)]
>>> G.neighbors(1)
 [2, 3]
```

```
>>> G.add_node(1, time='5pm')
>>> G.add_nodes_from([3], time='2pm')
>>> G.node[1] {'time': '5pm'}
>>> G.node[1]['room'] = 714
>>> G.nodes(data=True)
 [(1, {'room': 714, 'time': '5pm'}), (3, {'time': '2pm'})]
```

```
>>> G.add_edge(1, 2, weight=4.7 )
>>> G.add_edges_from([(3,4),(4,5)], color='red')
>>> G.add_edges_from([(1,2,{'color':'blue'}),
(2,3,{'weight':8})])
>>> G[1][2]['weight'] = 4.7
>>> G.edge[1][2]['weight'] = 4
```

Библиотека NetworkX

```
>>> DG=nx.DiGraph()
>>> DG.add_weighted_edges_from([(1,2,0.5),(3,1,0.75)])
>>> DG.out_degree(1,weight='weight')
0.5
>>> DG.degree(1,weight='weight')
1.25
>>> DG.successors(1)
[2]
>>> DG.neighbors(1)
[2]
```

```
>>> MG=nx.MultiGraph()
>>> MG.add_weighted_edges_from([(1,2,.5), (1,2,.75), (2,3,.5)])
>>> MG.degree(weight='weight')
 {1: 1.25, 2: 1.75, 3: 0.5}
>>> GG=nx.Graph()
>>> for n,nbrs in MG.adjacency_iter():
...         for nbr,edict in nbrs.items():
...                 minvalue= ([d['weight'] for d in edict.values()])
...                 GG.add_edge(n,nbr, weight = minvalue) ...
>>> nx.shortest_path(GG,1,3)
 [1, 2, 3]
```

Поиск в ширину(BFS) и глубину(DFS)

BFS

```
BFS(G, s)
 1  for Каждой вершины u ∈ G.V − {s}
 2       u.color = WHITE
 3       u.d = ∞
 4       u.π = NIL
 5  s.color = GRAY
 6  s.d = 0
 7  s.π = NIL
 8  Q = ∅
 9  ENQUEUE(Q, s)
10  while Q ≠ ∅
11       u = DEQUEUE(Q)
12       for Каждой вершины v ∈ G.Adj[u]
13            if v.color == WHITE
14                 v.color = GRAY
15                 v.d = u.d + 1
16                 v.π = u
17                 ENQUEUE(Q, v)
18       u.color = BLACK
```

DFS

```
DFS(G)
1  for каждой вершины u ∈ G.V
2       u.color = WHITE
3       u.π = NIL
4  time = 0
5  for каждой вершины u ∈ G.V
6       if u.color == WHITE
7            DFS-VISIT(G, u)

DFS-VISIT(G, u)
 1  time = time + 1
 2  u.d = time
 3  u.color = GRAY
 4  for каждой v ∈ G.Adj[u]
 5       if v.color == WHITE
 6            v.π = u
 7            DFS-VISIT(G, v)
 8  u.color = BLACK
 9  time = time + 1
10  u.f = time
```

Поиск в ширину(BFS) и глубину(DFS)

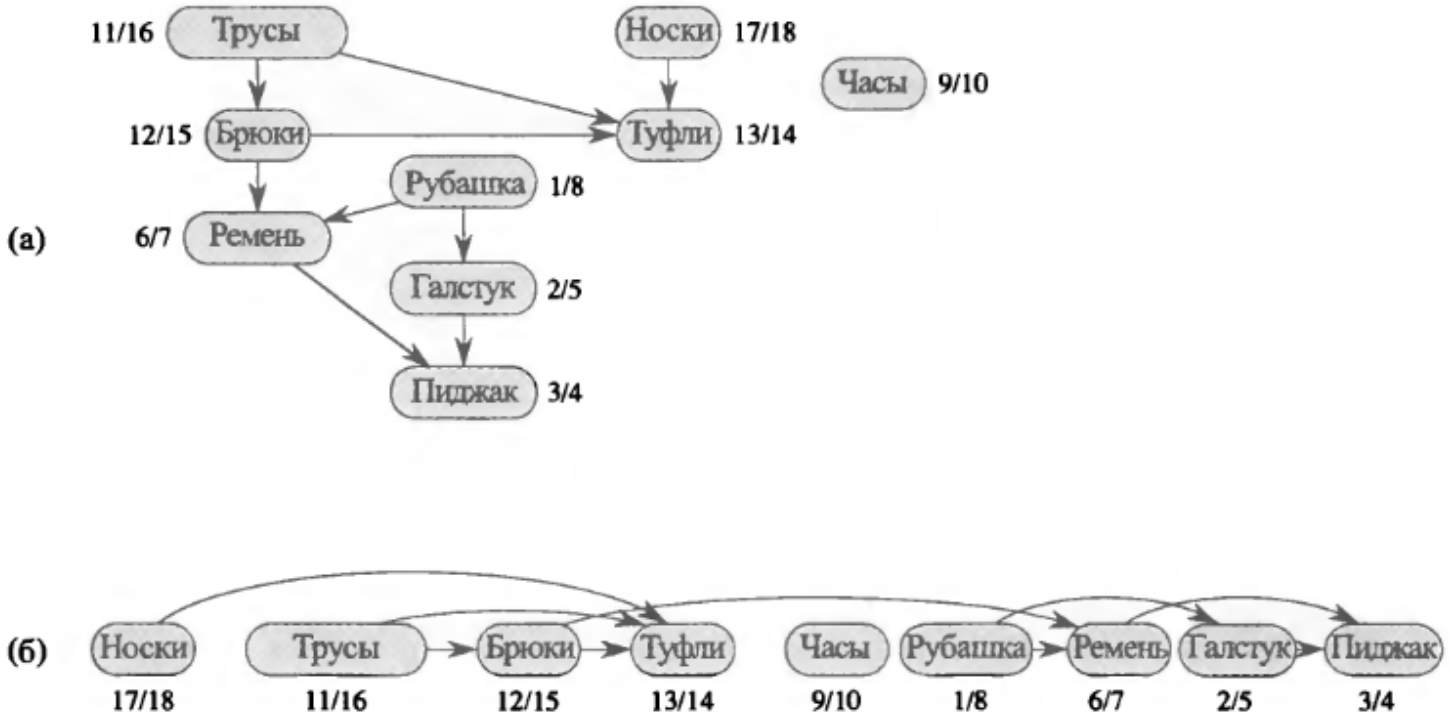
BFS

```
>>> G = nx.Graph()
>>> G.add_path([0,1,2])
>>> print((nx.bfs_edges(G,0)))
 [(0, 1), (1, 2)]
```

DFS

```
>>> G = nx.Graph()
>>> G.add_path([0,1,2])
>>> T = nx.dfs_tree(G,0)
>>> print(T.edges())
[(0, 1), (1, 2)]
```

Топологическая сортировка



TOPOLOGICAL-SORT(G)

1 Вызвать DFS(G) для вычисления времен завершения $v.f$
для каждой вершины v
2 По завершении работы над вершиной внести ее в начало
связанного списка
3 **return** связанный список вершин

>>> topological_sort

Минимальное остовное дерево

Алгоритм Крускала

```
MST-KRUSKAL(G, w)
1  A = ∅
2  for каждой вершины v ∈ G.V
3      MAKE-SET(v)
4  Отсортировать ребра G.E в неуменьшающемся порядке по весу w
5  for каждого ребра (u, v) ∈ G.E в этом порядке
6      if FIND-SET(u) ≠ FIND-SET(v)
7          A = A ∪ {(u, v)}
8          UNION(u, v)
9  return A
```

```
>>> G=nx.cycle_graph(4)
>>> G.add_edge(0,3,weight=2) # assign weight 2 to edge 0-3
>>> T=nx.minimum_spanning_tree(G)
>>> print(
(T.edges(data=True)))
[(0, 1, {}), (1, 2, {}), (2, 3, {})]
```

Поиск кратчайшего пути

Алгоритм Дейкстры

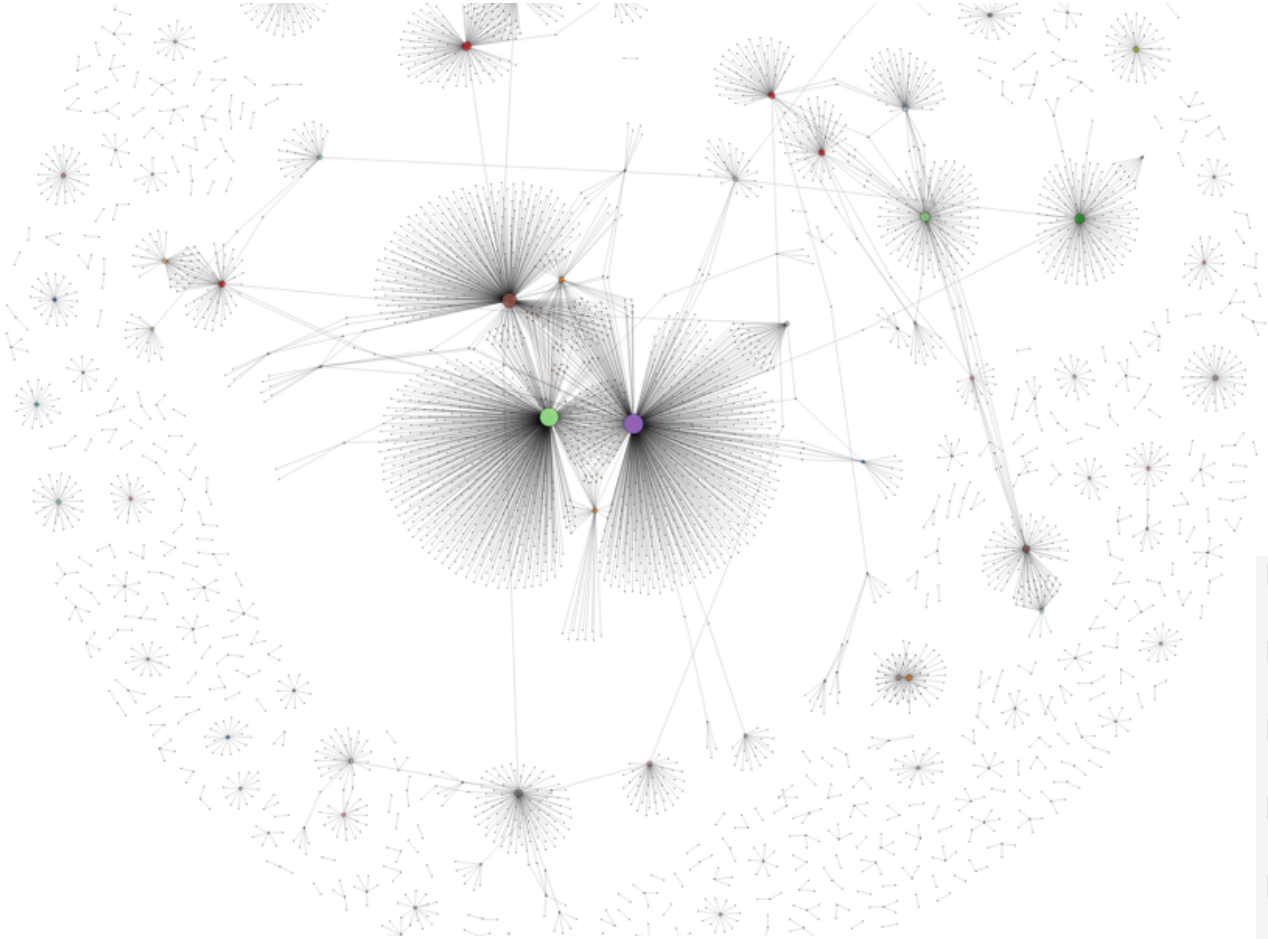
```
DIJKSTRA(G, w, s)
1  INITIALIZE-SINGLE-SOURCE(G, s)
2  S = ∅
3  Q = G.V
4  while Q ≠ ∅
5       u = EXTRACT-MIN(Q)
6       S = S ∪ {u}
7       for каждой вершины v ∈ G.Adj[u]
8            RELAX(u, v, w)
            // С соответствующими вызовами DECREASE-KEY
```

```
>>> G=nx.path_graph(5)
>>> print(nx.shortest_path(G,source=0,target=4))
 [0, 1, 2, 3, 4]
>>> p=nx.shortest_path(G,source=0) # target not specified
 >>> p[4] [0, 1, 2, 3, 4]
>>> p=nx.shortest_path(G,target=4)# source not specified
>>> p[0] [0, 1, 2, 3, 4]
>>> p=nx.shortest_path(G) # source,target not specified
>>> p[0][4] [0, 1, 2, 3, 4]
```

Поиск по социальному графу

Постановка задачи:

Для заданного пользователя найти 10 потенциальных друзей по максимальному количеству общих друзей.



Nodes	211.2K
Edges	1.5M
Density	6.74959e-05
Maximum degree	2K
Minimum degree	1
Average degree	14