Algorithms on Graphs

Module 2

Lecture 9

Shortest paths, part 2

Adigeev Mikhail Georgievich

mgadigeev@sfedu.ru

Dijkstra’s algorithm

Dijkstra’s* algorithm solves the Single Source Shortest Path problem (SSSP), i.e. problems 1 and 2 from slide 5.

For simplicity, we will consider the case of directed graphs.

This algorithm can be constructed as a kind of dynamic programming.

* See http://jeffe.cs.illinois.edu/teaching/algorithms/ for the historical survey and the discussion about the titles of algorithms.

Dijkstra’s algorithm

Let us start from a recursive expression for the value to be calculated, i.e. distance.

Let $\delta(v)$ denote the distance (= the weight of the shortest path) from the given source vertex s to a certain vertex v.

Dijkstra’s algorithm

A naïve way to define $\delta(v)$ is this:

$$\delta(v) = \begin{cases} 0, & v = s \\ \min_{(u,v)\in E}\{\delta(u) + w(u,v)\}, & otherwise \end{cases}$$

But this recurrence is valid for DAGs only. If the graph contains a directed cycle, we cannot use this recurrence directly.

Dijkstra’s algorithm

To overcome this issue, we introduce the second parameter i. Let $\delta(i, v)$ denote the minimum weight of a path from s to v which contains at most i edges.

$$\delta(i, v) = \begin{cases} 0, & if\ v = s\ and\ i = 0 \\ \infty, & if\ v \neq s\ and\ i = 0 \\ \min\left[\begin{matrix} \delta(i-1, v), \\ \min\limits_{(u,v)\in E}\{\delta(i-1, u) + w(u, v)\} \end{matrix}\right], & otherwise \end{cases}$$

Dijkstra’s algorithm

The pseudocode of the algorithm:

```
// Vertices are identified with
// their indices, 0..n-1
Create matrix d[0..n, 0..n-1].
// Initialization
d[0,s] = 0
for v = 0 to n-1:
    if v != s then d[0,v] = ∞
```

Dijkstra’s algorithm

```
// Filling the table
for i=1 to n-1:
    for each vertex v:
        d[i,v] = d[i-1,v]
        for each edge (u,v):
            if d[i-1,u]+w[u,v]<d[i,v]
                then d[i,v]=d[i-1,u]+w[u,v]
```

Dijkstra’s algorithm

```
// Filling the table
for i=1 to n-1:
    for each vertex v:
        d[i,v] = d[i-1,v]
        for each edge (u,v):
            if d[i-1,u]+w[u,v]<d[i,v]
            then d[i,v]=d[i-1,u]+w[u,v]
```

Each edge is processed exactly once.
The order does not matter!

Dijkstra’s algorithm

```
// Filling the table
for i=1 to n-1:
    for each vertex v:
        d[i,v] = d[i-1,v]
    for each edge (u,v):
        if d[i-1,u]+w[u,v]<d[i,v]
            then d[i,v]=d[i-1,u]+w[u,v]
```

Dijkstra’s algorithm

```
// Filling the table
for i=1 to n-1:
    for each vertex v:
        d[i,v] = d[i-1,v]
    for each edge (u,v):
        if d[i-1,u]+w[u,v]<d[i,v]
            then d[i,v]=d[i-1,u]+w[u,v]
```

We can omit index i !!!

Dijkstra’s algorithm

```
// Filling the table
for i=1 to n-1:
    for each edge (u,v):
        if d[u]+w[u,v]<d[v]
            then d[v]=d[u]+w[u,v]
```

Time complexity: $O(nm), n = |V|, m = |E|$.

Dijkstra’s algorithm: non-negative edges

Let us see at the Dijkstra’s algorithm for the general case.

```
// Filling the table
for i=1 to n-1:
    for each edge (u,v):
        if d[u]+w[u,v]<d[v]
            then d[v]=d[u]+w[u,v]
```

For the case of non-negative edges, we can organize calculations in a way that each edge is processed at most once.

Dijkstra’s algorithm: non-negative edges

For the case of non-negative edges, we can organize calculations in a way that each edge is processed at most once.

In order to get such improvement we need to analyze and process vertices in the order of increasing their distances from s. We select an unprocessed vertex v with the minimum tentative distance from s and build the shortest path to v by augmenting the path to some other previously processed vertex (the *predecessor* of v).

Dijkstra’s algorithm: non-negative edges

The initialization is essentially the same:

```
// Vertices are identified with
// their indices, 0..n-1
Create matrix d[0..n-1].
// Initialization
d[s] = 0
for v = 0 to n-1:
    if v != s then d[v] = ∞
```

Dijkstra’s algorithm: non-negative edges

But we will use a priority queue similar to BFS. The keys will be the tentative distances from s to all other vertices

```
for v = 0 to n-1: Enqueue(v,d[v]);
```

Then we iteratively process vertices; at each iteration we select the vertex with the minimum tentative distance.

Dijkstra’s algorithm: non-negative edges

```
While (Queue is not empty):
      u = GetMin()
      DelMin()
      for each edge (u,v):
            if d[u]+w[u,v]<d[v] then
                  d[v]=d[u]+w[u,v]
                  ChangePriority(v, d[v])
```

Each vertex is extracted from the priority queue only once. Hence, each edge is processed at most once. Hence, time complexity: $O(m \cdot \log n)$, where m is the quantity of edges, $O(\log n)$ is the complexity of a priority queue operation. Time complexity of the general version: $O(nm)$

Further issues

In the next lecture we will explore more issues related to the shortest path problem:

- Building the shortest paths, in addition to the distances.
- Problem 3 (all-to-all shortest paths problem).