Algorithms and Data Structures
Module 3. Dynamic programming

Lecture 14

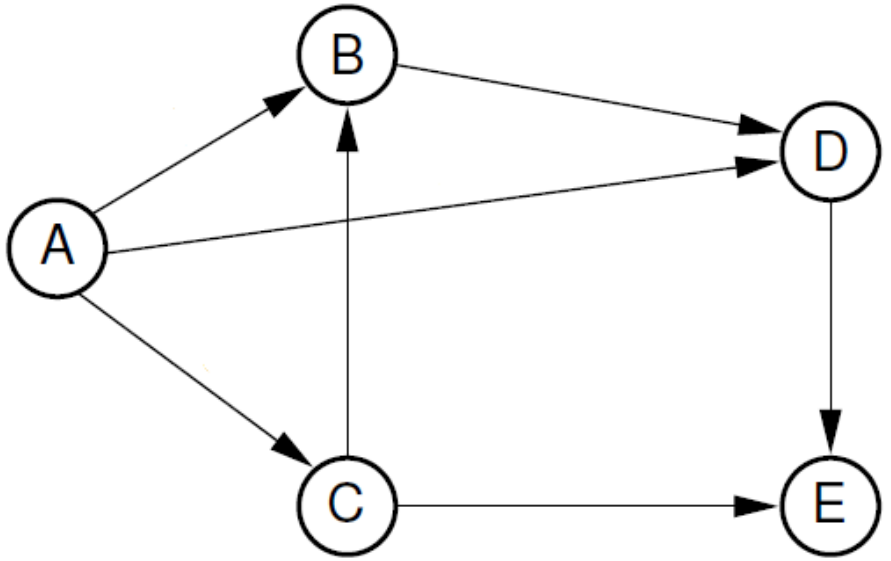
Dynamic programming for calculating distances in graphs. Part 1.

BFS: applications (lecture 5)

Graph G=(V,E).

A *distance* between vertices *u* and *v* is the minimum length of the path between *u* and *v*.

dist(A,E) = 2

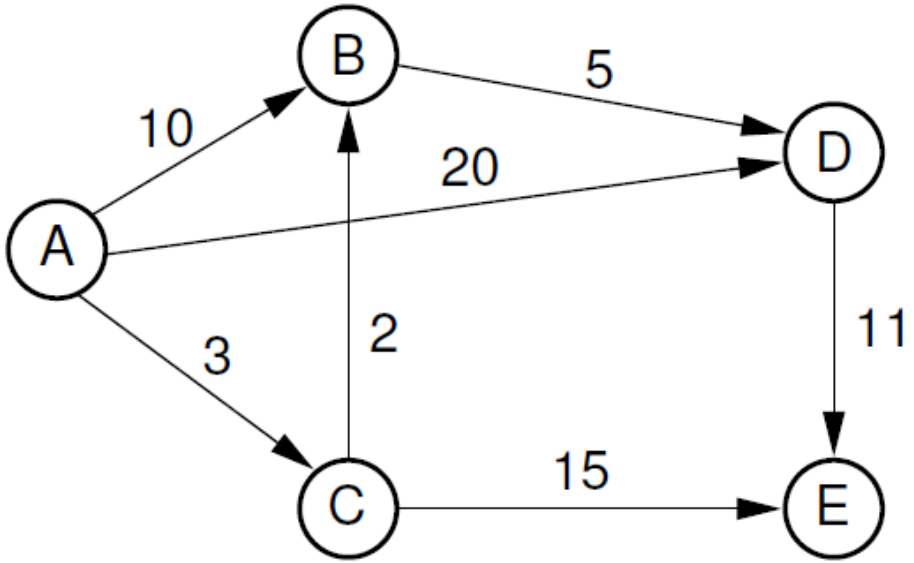


BFS: applications (lecture 5)

Weighted graph G=(V,E), $w: E \rightarrow R$

A *distance* between vertices *u* and *v* is the minimum weight (=sum of edges' weights) of the path between *u* and *v*.

dist(A,E) = 18



BFS: applications (lecture 5)

For unweighted graphs distances from $s \in V$ to all other vertices can be calculated using BFS.

For weighted graphs: Dijkstra algorithm works like a BFS and calculates distances (from $s \in V$ to each of other vertices) on a graph.

Problem definition

Given: a weighted graph $G(V, E)$, edge weights $w: E \to R$.

Problem 1: For vertices $s \in V$ (*source*) and $t \in V$ (*target*) find the distance and the shortest path from s to t.

Problem 2: For a vertex $s \in V$ (*source*) find distances and the shortest paths from s to every other vertex.

Problem 3: Find distances and the shortest paths from s to t for all pairs of vertices.

If there are several shortest paths between two vertices, (usually) it is enough to find any of them.

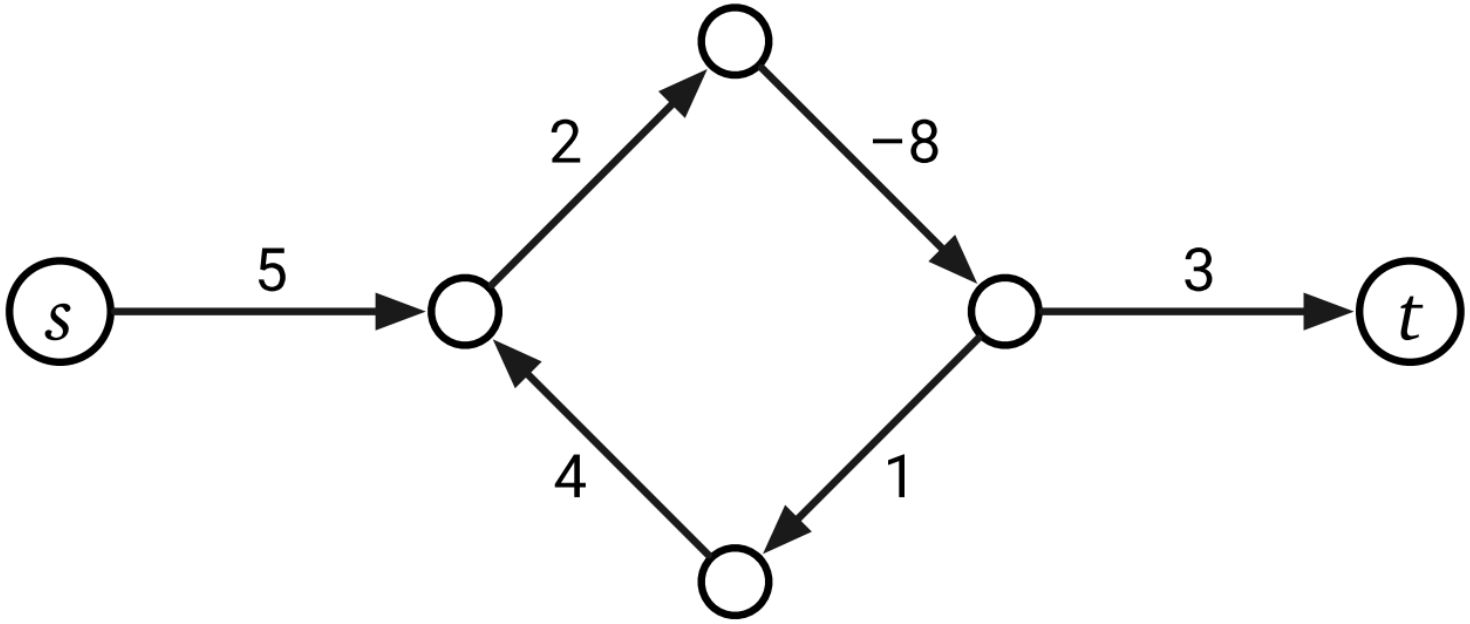
Negative weights

With respect to algorithmic issues, it is convenient to distinguish the general case and the case of problems with non-negative weights.

Reason: negative edge weights make several difficulties for algorithms, and for many practical applications weights are naturally non-negative.

Negative weights

If a graph contains a negative cycle (cycle whose total weight is negative), some pairs of vertices have no shortest paths.



http://jeffe.cs.illinois.edu/teaching/algorithms/

Negative weights

Definition. A path is called *simple* iff it does not contain any edge more than once.

For any pair of vertices s and t, if t is reachable from s then there is a shortest simple path from s and t, even in case of negative cycles. But if there are negative cycles, finding a shortest path becomes an NP-hard problem, i.e. it cannot be solved efficiently.

Negative weights

If a directed graph has negative edges but has no negative cycles, the shortest path problem can be solved efficiently with the algorithms considered in this lecture.

For undirected graphs, there are specialized algorithms, that will not be studied in this course.

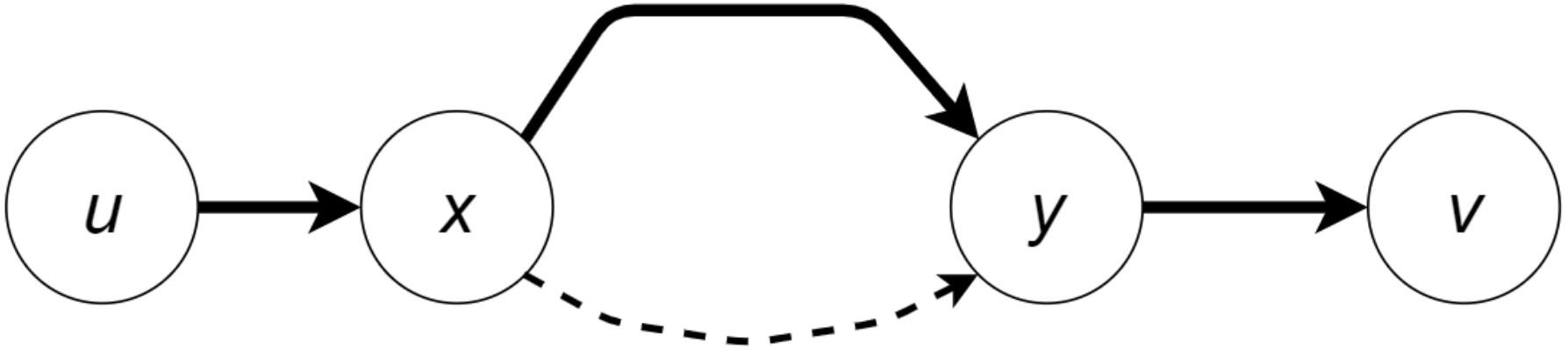
Principle of optimality

The principle of optimality is the basic condition for applicability of dynamic programming for optimization problems.

Principle of optimality for the shortest path problem. Let $G(V, E)$ be a graph with non-negative edge weights $w: E \to R_+$ and u, v – two vertices of G. Any part of a shortest path between u and v is a shortest path between its endpoints.

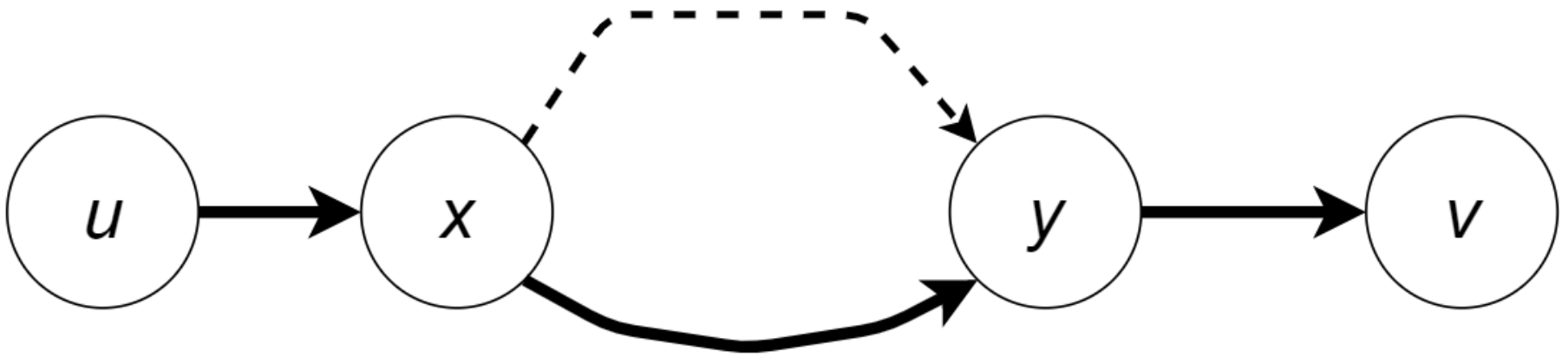
Principle of optimality

Proof. Let us consider a path p between u and v, which goes through vertices x and y (it may be that $x = u$ or $y = v$). Suppose that the part of p between x and y is not the shortest path between x and y.



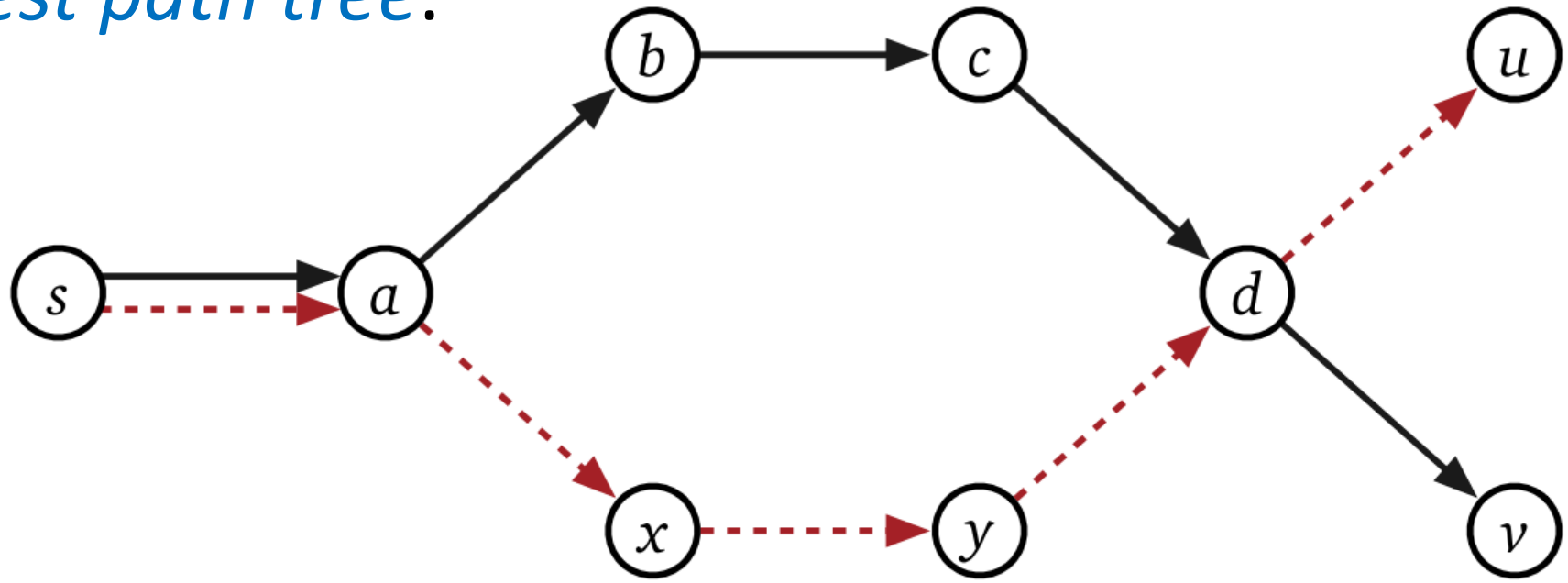
Principle of optimality

Let us replace this part of p with the shortest path between x and y. We will yield the path p', whose weight is less then the weight of p. Thus, p is not the shortest path between u and v.



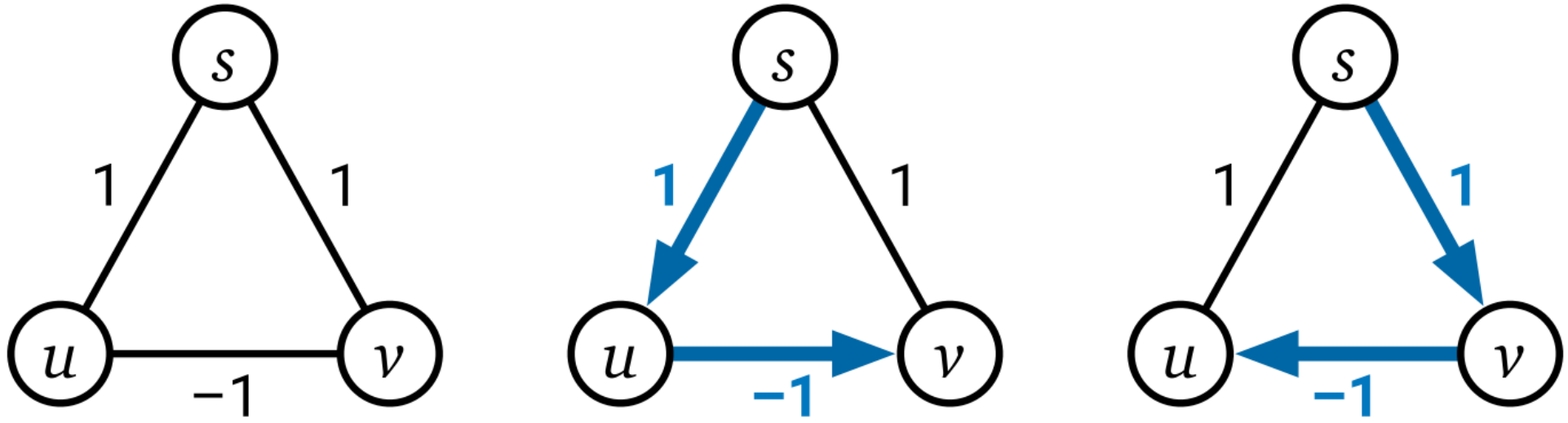
Principle of optimality

Due to the principle of optimality, the shortest paths from a given source vertex to all other vertices make the *shortest path tree*.



Principle of optimality

For undirected graphs with negative edges, the principle of optimality does not hold.



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:

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

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.

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

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

Further issues

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

- Dijkstra's algorithm version for the case of non-negative edge weights (based on the 'best-first' graph traversal).
- Building the shortest paths, in addition to the distances.
- Problem 3 (all-to-all shortest paths problem).