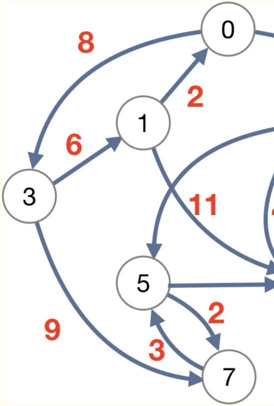
Introduction to Algorithms
Lecture 9 BFS and Shortest Paths

Xue Chen
xuechen1989@ustc.edu.cn
2025 spring in

Outline

Shortest Path Prob

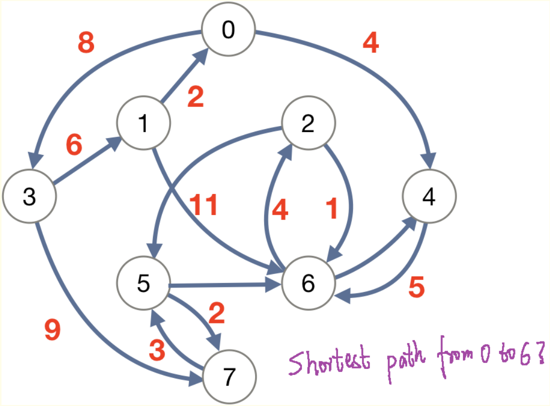
Given G and a pair of vertices s and t, find the shortest path $s \rightsquigarrow t$.



Shortest path from 0 to 6?

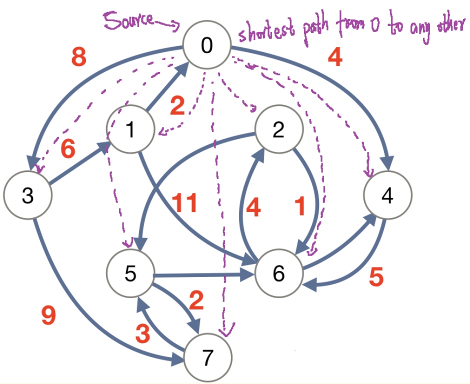
Shortest Path Prob

Given G and a pair of vertices s and t, find the shortest path $s \leadsto t$.



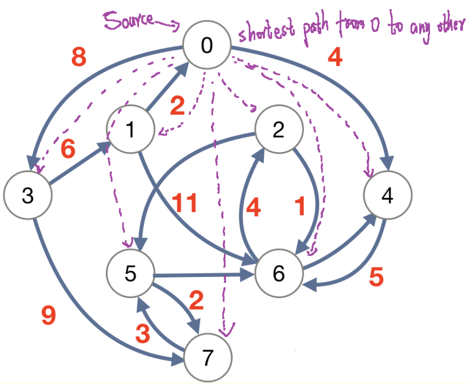
1. One of the most fundamental prob in CS
2. Many applications: Routing traffic, traveling/booking flights, experiments, ...
3. Various algorithms for different graphs

Overview



Consider the shortest path from a fixed source *s* to all other nodes

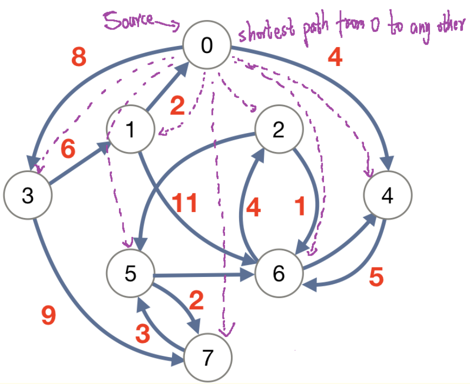
Overview



Consider the shortest path from a fixed source s to all other nodes

1. Unlike connected components, no difference between directed graphs and undirected graphs
2. Simplest case: BFS for unit-weight edges

Overview



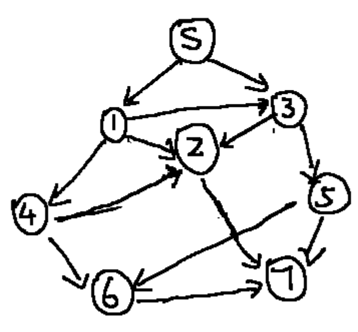
Consider the shortest path from a fixed source s to all other nodes

1. Unlike connected components, no difference between directed graphs and undirected graphs
2. Simplest case: BFS for unit-weight edges
3. Dijkstra's algorithm for non-negative weights
4. Bellman-Ford algorithm for negative weights

Outline

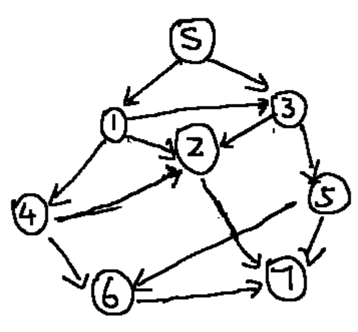
Introduction

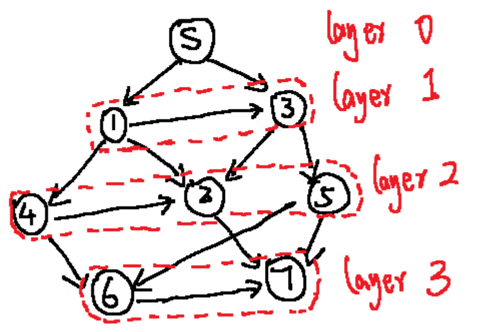
For a unweighted graph G and source $s \in V$, find the shortest distance from s to any other nodes



Introduction

For a unweighted graph G and source $s \in V$, find the shortest distance from s to any other nodes





Basic idea

Find all vertices in the 1st layer, 2nd layer, and so on

— focus on the breadth of each layer instead of the depth

Implementation

Instead of maintaining vertices on layers, use a first-in first-out queue to store all vertices in order

---

procedure BFS(s)
  Initialize $v.d = +\infty$, $v.\pi = NIL$, $v.color = White$ for all v except s
  $\textsc{Enqueue}(Q, s)$
  while Q is not empty **do**
    $u = \textsc{Dequeue}(Q)$
    for each outgoing-edge (u, w) **do**
      if $w.color = White$ **then**
        $w.color = Black$, $w.\pi = u$, $w.d = u.d + 1$
        $\textsc{Enqueue}(Q, w)$

Implementation

Instead of maintaining vertices on layers, use a first-in first-out queue to store all vertices in order

---

procedure BFS(s)
  Initialize $v.d = +\infty$, $v.\pi = NIL$, $v.color = White$ for all v except s
  ENQUEUE(Q, s)
  while Q is not empty **do**
    $u = $ DEQUEUE(Q)
    for each outgoing-edge (u, w) **do**
      if $w.color = White$ **then**
        $w.color = Black$, $w.\pi = u$, $w.d = u.d + 1$
        ENQUEUE(Q, w)

Lemma 22.3 in CLRS

At any moment, the head v_h and tail v_t in Q satisfy $v_t.d \leqslant v_h.d + 1$. Also, for two adjacent vertices v_i and v_{i+1} in Q, $v_i.d \leqslant v_{i+1}.d$.

Analysis

Running Time

$O(n + m)$ because each vertex will be added to Q at most once and each edge will be checked for at most once

Correctness: For any u, $u.d$ equals the length of the shortest path from s to u.

Analysis

Running Time

$O(n+m)$ because each vertex will be added to Q at most once and each edge will be checked for at most once

Correctness: For any u, $u.d$ equals the length of the shortest path from s to u.
Induction: If there is a length-d $s \rightsquigarrow v$ path, then $\exists$ u adjacent to v with a length-$(d-1)$ $s \rightsquigarrow u$ path.

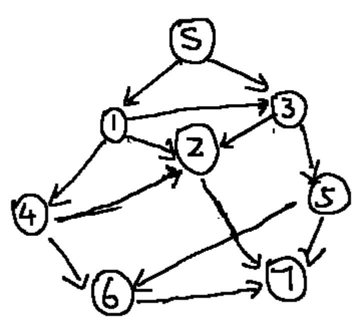
Discussion

1. Simplest algorithm of shortest paths
2. Crucially relies on unit-weight edges
3. Another algorithm to explore the graph — many differences compared to DFS
   - DFS uses a stack while BFS uses a queue
   - DFS goes depth first — very long path; but BFS is breadth first — expand one layer to the next
   - Different types of edges in DFS and BFS trees

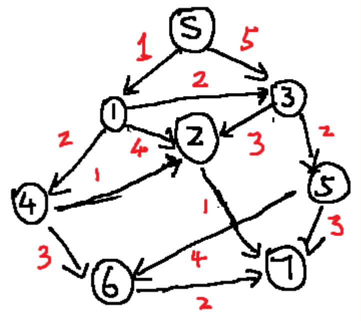
Outline

Intro

How about weighted graphs?

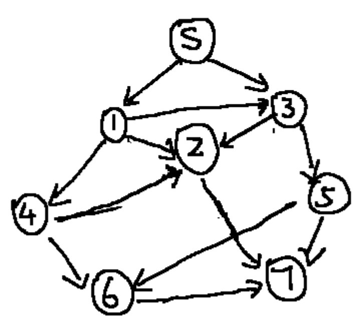


$\Rightarrow$

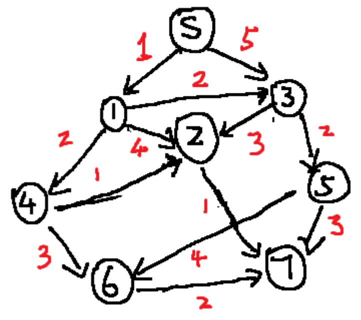


Intro

How about weighted graphs?



$\Rightarrow$



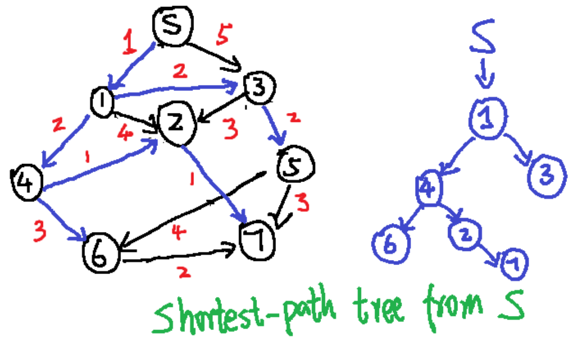
Observation

BFS does not work any more:

Shortest path $S \rightsquigarrow 3$ is $S \to 1 \to 3$ not $S \to 3$

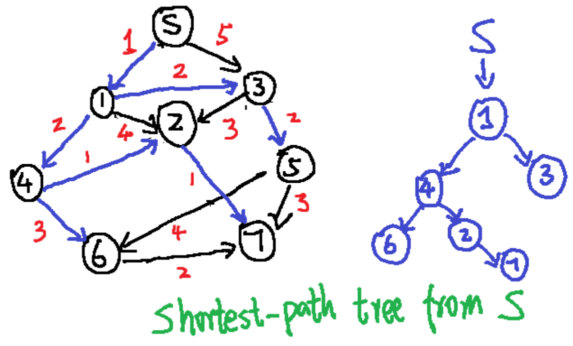
Basic Properties

Let $\delta(s, v)$ be the shortest distance from s to v and $w(u, v)$ denote the length of edge (u, v)



Basic Properties

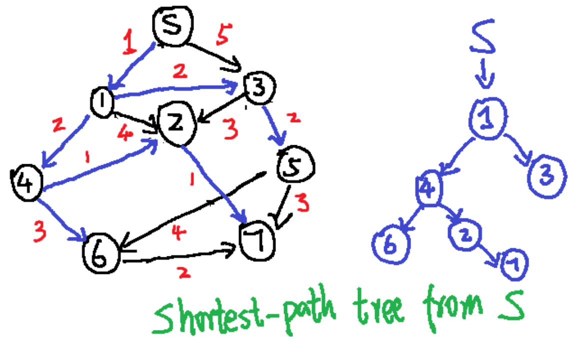
Let $\delta(s, v)$ be the shortest distance from s to v and $w(u, v)$ denote the length of edge (u, v)



1. $\delta(s, u) = \min_{\text{paths } i_0=s,\ldots,i_k=u} \sum_{j=1}^{k} w(i_{j-1}, i_j)$
2. $\delta(s, u) = \min_v \left\{\delta(s, v) + w(v, u)\right\}$

Basic Properties

Let $\delta(s, v)$ be the shortest distance from s to v and $w(u, v)$ denote the length of edge (u, v)

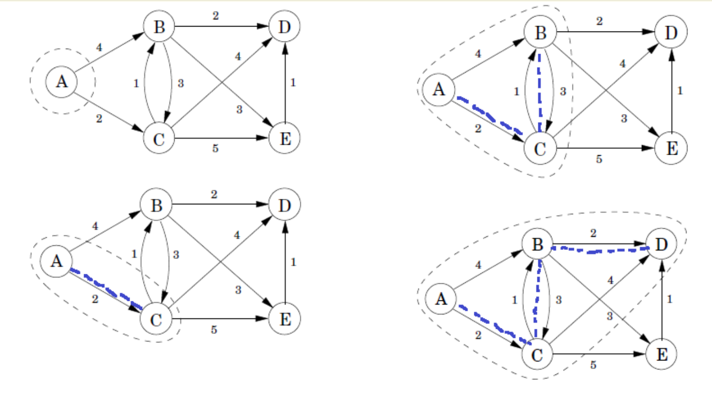


1. $\delta(s, u) = \min_{\text{paths } i_0 = s, \ldots, i_k = u} \sum_{j=1}^{k} w(i_{j-1}, i_j)$
2. $\delta(s, u) = \min_v \left\{\delta(s, v) + w(v, u)\right\}$
3. $\forall v, \delta(s, u) \leqslant \delta(s, v) + w(v, u)$
4. Lemma 24.17 in CLRS: All shortest paths from s form a tree

Dijkstra's Algorithm

Idea

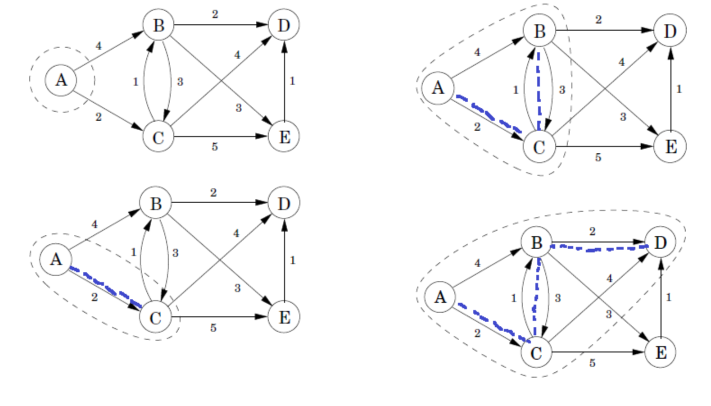
Starting from source, find shortest paths one by one according to the length — or expand the shortest-path tree by greedy



Dijkstra's Algorithm

Idea

Starting from source, find shortest paths one by one according to the length — or expand the shortest-path tree by greedy



Record those vertices whose shortest path has been determined and find the next one by greedy

Formal Description

procedure DIJKSTRA(s)
  Initialize $d[v] = +\infty$ for all v except s
  $S = \emptyset$
  Q is a min-heap maintaining $V - S$ according d
  while Q is not empty **do**
    $u = \textsc{Extract-Min}(Q)$
    $S = S \cup \{u\}$
    for each outgoing-edge (u, v) **do**
      if $d[v] > d[u] + w(u, v)$ **then**
        $d[v] = d[u] + w(u, v)$ and adjust v's position in the heap

Analysis

Running Time

The running time is $O(n \log n + m \log n)$ via a binary heap.

Analysis

Running Time

The running time is $O(n \log n + m \log n)$ via a binary heap.

1. Each node u will be extracted at most once
2. Each edge (u, v) will be checked once

Analysis

Running Time

The running time is $O(n \log n + m \log n)$ via a binary heap.

1. Each node u will be extracted at most once
2. Each edge (u, v) will be checked once
3. Each extraction and update of $d(v)$ will take $O(\log n)$-time in the heap

Correctness

Lemma 24.6 in CLRS

Dijkstra's algorithm finds the shortest distance from s to any other nodes for graphs with non-negative weights.

Induction: At any moment, any $v \in S$ always has $d[v] = \delta(s, v)$

Correctness

Lemma 24.6 in CLRS

Dijkstra's algorithm finds the shortest distance from s to any other nodes for graphs with non-negative weights.

Induction: At any moment, any $v \in S$ always has $d[v] = \delta(s, v)$

1. $S = \emptyset$ in initialization
2. For contradiction, consider the 1st node u in S with $d[u] \neq \delta(s, u)$ to S
3. Consider the correct shortest path from s to u

Correctness

Lemma 24.6 in CLRS

Dijkstra's algorithm finds the shortest distance from s to any other nodes for graphs with non-negative weights.

Induction: At any moment, any $v \in S$ always has $d[v] = \delta(s, v)$

1. $S = \emptyset$ in initialization
2. For contradiction, consider the 1st node u in S with $d[u] \neq \delta(s, u)$ to S
3. Consider the correct shortest path from s to u
4. Say the 1st vertex not in S is w (could be u)
   $\Rightarrow$ w's parent node is in S
5. Then $d[w] = \delta(s, w)$ — we shall add w to S, which provides the contradiction

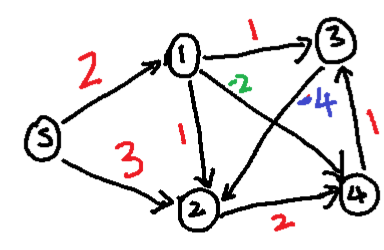
Discussion

1. Dijstra's algorithm is a greedy algorithm that extends BFS from unit weights to non-negative weights
2. Heaps improve the running time to $O(n \log n + m \log n)$
3. Shortest paths form a tree
4. Next: How about graphs with negative weights?

Outline

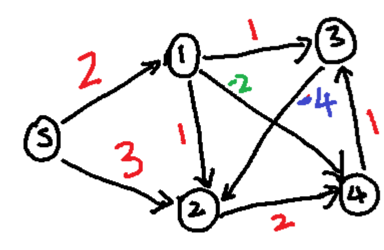
Intro

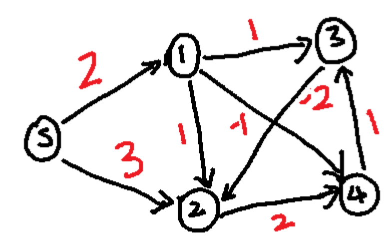
How to find the shortest distance from s for a graph with negative weights?



Intro

How to find the shortest distance from s for a graph with negative weights?



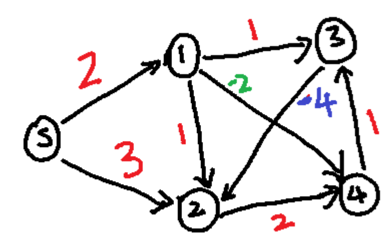


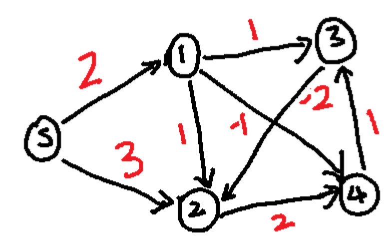
Two Cases

1. $\exists$ a cycle C whose sum is negative $\Rightarrow$ no shortest path!
2. $\nexists$ such a cycle

Intro

How to find the shortest distance from s for a graph with negative weights?



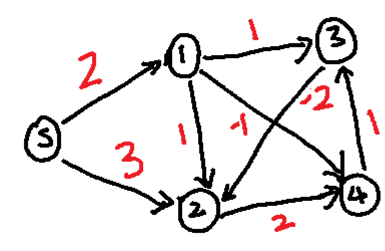


Two Cases

1. $\exists$ a cycle C whose sum is negative $\Rightarrow$ no shortest path!
2. $\nexists$ such a cycle

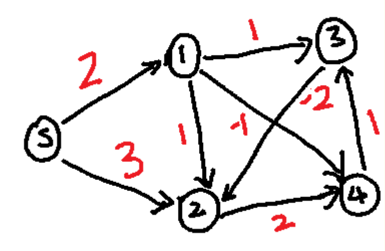
In the 1st case, the shortest path is undefined for some nodes (or say does not exist)

Basic Idea



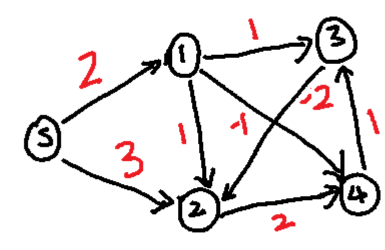
1. If $\exists$ negative cycles, algorithm shall output $\perp$
2. Otherwise it finds all shortest paths from s

Basic Idea



1. If $\exists$ negative cycles, algorithm shall output $\perp$
2. Otherwise it finds all shortest paths from s
3. Dijstra's algorithm does not work b.c. of negative edges
4. In the above example, we need to re-put vertex 2 in Q — repeat this for many times

Basic Idea



1. If $\exists$ negative cycles, algorithm shall output $\perp$
2. Otherwise it finds all shortest paths from s
3. Dijstra's algorithm does not work b.c. of negative edges
4. In the above example, we need to re-put vertex 2 in Q — repeat this for many times
5. How many repetitions for each node? At most $n-1$ times — the longest path will have $n-1$ edges o.w. $\exists$ negative cycles

Description of Bellman-Ford

procedure BELLMAN-FORD(s)
  Initialize $d[v] = +\infty$ for all v except s
  for $i = 1, \ldots, n-1$ **do**
    for each edge (u, v) **do**
      if $d[v] > d[u] + w(u, v)$ **then**
        $d[v] = d[u] + w(u, v)$
  for each edge (u, v) **do**
    if $d[v] > d[u] + w(u, v)$ **then**
      Report $\exists$ negative cycles
  Otherwise claim $d[v]$ stores the shortest distance to s

Description of Bellman-Ford

```
procedure BELLMAN-FORD(s)
    Initialize d[v] = +∞ for all v except s
    for i = 1, ..., n − 1 do
        for each edge (u, v) do
            if d[v] > d[u] + w(u, v) then
                d[v] = d[u] + w(u, v)
    for each edge (u, v) do
        if d[v] > d[u] + w(u, v) then
            Report ∃ negative cycles
    Otherwise claim d[v] stores the shortest distance to s
```

Running time: $O(n \cdot m)$.

Analysis

Correctness

1. If $\exists$ a negative cycle, it always outputs negative cycles
2. Otherwise show $d[v] = \delta(s, v)$ for all v

Analysis

Correctness

1. If $\exists$ a negative cycle, it always outputs negative cycles
2. Otherwise show $d[v] = \delta(s, v)$ for all v

Let the length of a path p be the number of edges on it:

$$\delta(s, v) = \min_{\text{any path } p} \text{weight}(p) = \min_{\text{length } i=1,2,\dots} \left\{ \min_{p \text{ of length } i} \text{weight}(p) \right\}.$$

Analysis

Correctness

1. If $\exists$ a negative cycle, it always outputs negative cycles
2. Otherwise show $d[v] = \delta(s, v)$ for all v

Let the length of a path p be the number of edges on it:

$$\delta(s, v) = \min_{\text{any path } p} \text{weight}(p) = \min_{\text{length } i=1,2,\ldots} \left\{ \min_{p \text{ of length } i} \text{weight}(p) \right\}.$$

If there is no negative cycle, we can stop at $i = n - 1$:

$$\delta(s, v) = \min_{\text{length } i \leqslant n-1} \left\{ \min_{p \text{ of length } i} \text{weight}(p) \right\}$$

Analysis

Correctness

1. If $\exists$ a negative cycle, it always outputs negative cycles
2. Otherwise show $d[v] = \delta(s, v)$ for all v

Let the length of a path p be the number of edges on it:

$$\delta(s, v) = \min_{\text{any path } p} \text{weight}(p) = \min_{\text{length } i=1,2,\ldots} \left\{ \min_{p \text{ of length } i} \text{weight}(p) \right\}.$$

If there is no negative cycle, we can stop at $i = n - 1$:

$$\delta(s, v) = \min_{\text{length } i \leqslant n-1} \left\{ \min_{p \text{ of length } i} \text{weight}(p) \right\}$$

After iteration i in BELLMAN-FORD, $d[v] = \min_{p \text{ of length } \leqslant i} \text{weight}(p)$

Summary

Various algorithm for various graphs

1. No difference between directed and undirected graph (except the negative edge)
2. BFS, Dijkstra, Bellman-Ford, ...
3. Next: All pairs shortest paths!

Questions?