CS213/293 Data Structure and Algorithms 2025

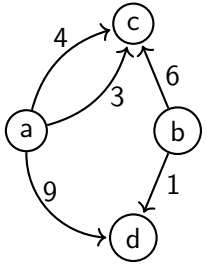
Lecture 17: Graphs - Shortest path

Instructor: Ashutosh Gupta

IITB India

Compile date: November 14, 2025

Labeled directed graph



Definition 17.1

A labeled directed graph $G = (V, E)$ is consists of

- set V of vertices and
- set $E \subseteq V \times \mathbb{Q}^+ \times V$.

For $e \in E$, we will write $L(e)$ to denote the label.

The above is a labeled graph $G = (V, E)$, where

$V = \{a, b, c, d\}$ and

$E = \{(a, 3, c), (a, 4, c), (a, 9, d), (b, 6, c), (b, 1, d)\}$.

$L((a, 3, c)) = 3$.

Shortest path

Consider a labeled directed graph $G = (V, E)$.

Definition 17.2

For vertices $s, t \in V$, a path from s to t is a sequence of edges $e_1, ..., e_n$ from E such that there is a sequence of nodes $v_1, ..., v_{n+1}$ such that $v_1 = s$, $v_{n+1} = t$, and $e_i = (v_i, \_, v_{i+1})$ for each $i \in 1..n$.

Definition 17.3

The length/weight of $e_1, .., e_n$ is $\sum_{i=1}^{n} L(e_i)$.

Definition 17.4

For vertex $s, t \in V$, a shortest path is a path from s to t such that the length of the path is minimum.

Commentary: Does the above definition work for $n = 0$?

Use case: navigation

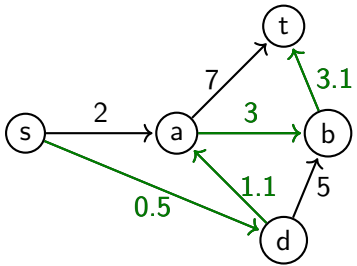
Example 17.1

Navigation via phones is a widely used application, which computes shortest paths.

Example: shortest path

Example 17.2

The shortest path from s to t is 0.5,1.1,3,3.1.



Exercise 17.1

a. How many simple paths are there from s to t?

b. Show that there are potentially exponentially many simple paths between two vertices.

Problem: single source shortest path(SSSP)

To compute a shortest path from s to t, we need to say that there is no shorter way to reach t.

We need to effectively solve the following problem.

Definition 17.5
Find shortest paths starting from a vertex s to all vertices in G.

Definition 17.6
Let $SP(x)$ denote the length of a shortest path from s to x.

Observation: relating SP of neighbors

An edge bounds the difference between the SP values of both ends.

For $(v, k, w) \in E$, we can conclude

$$SP(w) \leq SP(v) + k.$$

Exercise 17.2

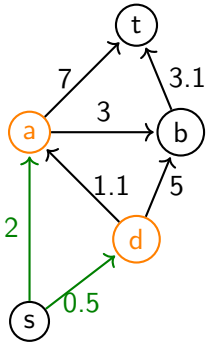
Write a formal proof of the above claim.

Commentary: Take your time to understand the above observation.

Observation: upper bounds of paths

Example 17.3

Considering only outgoing edges from s, what can we say about a shortest path from s to a and d?



$SP(s) = 0$

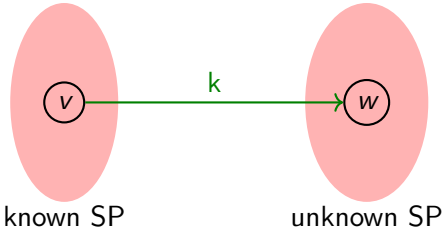
$SP(a) \leq 2 + SP(s)$

$SP(d) = 0.5 + SP(s)$

Observation: Since we know SP to s, we can compute SP to the closest neighbor and upper bound of SP for the other neighbors.

Can we lift the observation for a set of nodes?

Let us suppose we know SP for a set of vertices. What can we say about the remaining vertices?



$SP(w) \leq SP(v) + k$ holds for all edges that are in the cut between known and unknown.

Can we say something more about $SP(w)$ for which $SP(v) + k$ is the minimum among all edges on the cut?

Expanding known set

Consider labeled directed graph $G = (V, E)$.

Theorem 17.1

Let C be the cut for set $S \subset V$ in G.Let $d = min\{SP(v') + k | (v', k, \_) \in C\}$ and $(v, k, w) \in C$ achieves the minimum. Then, $SP(w) = d$.

Proof.

Let us suppose there is a path $e_1, .., e_n$ from s to w such that $L(e_1, ...e_n) < d$.The path has prefix

$$\underbrace{e_1 ... e_j}_{\in S} \underbrace{e_{j+1}}_{\in C} .....$$

Let $e_{j+1} = (v', k, w') \in C$. Therefore, $L(e_1, ...e_j e_{j+1}) \geq SP(v') + k$.

Due to the definition of d, $SP(v') + k \geq d$. Therefore, $L(e_1, .., e_n) \geq d$. Contradiction.

Therefore, $SP(w) \geq d$. □

Dijkstra's algorithm

Algorithm 17.1: SSSP(Graph $G = (V, E)$, vertex s)

```
1  Heap unknown; int sp[];
2  for v ∈ V do
3      v.visited := False;
4      unknown.insert(v, ∞);
5  unknown.decreasePriority(s, 0);
6  sp[s] := 0;
7  while unknown ≠ ∅ do
8      v := unknown.deleteMin();
9      for e = (v, k, w) ∈ E do
10         if ¬w.visited then
11             unknown.decreasePriority(w, k + sp[v]);
12             sp[w] := min(sp[w], k + sp[v]);
13     v.visited := True
```

Example: Dijkstra's algorithm

Consider the following graph. We start with vertex s. $SP(s) = 0$.
The cut has edges 1.9 and 0.5.

The minimum path on the cut is $SP(s) + 0.5$. $SP(d) = 0.5$.

Now the cut has edges 1.9, 1, and 5.

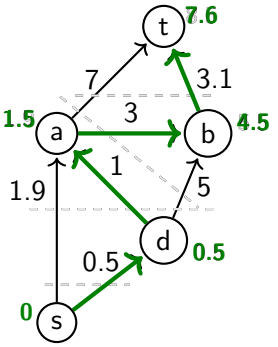
The minimum path on the cut is $SP(d) + 1$. $SP(a) = 1.5$.

Now the cut has edges 7, 3, and 5.

The minimum path on the cut is $SP(a) + 3$. $SP(b) = 4.5$.

Now the cut has edges 7 and 3.1.

The minimum path on the cut is $SP(b) + 3.1$. $SP(t) = 7.6$.



Exercise 17.3

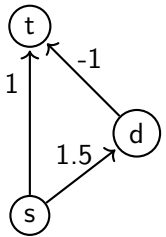
Modify Dijkstra's algorithm to construct the shortest paths from s to every vertex t.

Negative lengths

Dijkstra's algorithm does not work for negative lengths.

Example 17.4

On the following graph, Dijkstra's algorithm will return wrong shortest path.



The algorithm uses an argument that depends on the monotonic increase of length.

The algorithms that can handle negative lengths are Bellman-Ford and Floyd-Warshall.

Topic 17.1

A* algorithm (Not part of this course!)

Risk of exploring entire graph

Dijkstra's algorithm explores a graph without considering the target vertex.

The algorithm may explore a large part of the graph that is away from the target vertex.

Goal directed algorithm

A* algorithm is a modified Dijkstra's algorithm that is goal directed.

How to inform the algorithm about the potential direction of the target?

Heuristic function h

$$h : V \to \mathbb{R}^+$$

h estimates the shortest path from a node to the target.

Example 17.5

h can be the Euclidian distance between two points on map.

A* algorithm

Algorithm 17.2: A*(Graph $G = (V, E)$, vertex s, vertex t, heuristic function h)

```
1  Heap openSet; int sp[];
2  for v ∈ V do
3      sp[v] := ∞;
4      openSet.insert(v, ∞);
5  sp[s] := 0;
6  openSet.decreasePriority(s, sp[s] + h(s));
7  while openSet ≠ ∅ do
8      v := openSet.deleteMin();
9      if v = t then return sp[v]; ;
10     for e = (v, k, w) ∈ E do
11         if sp[v] + k < sp[w] then
12             sp[w] := sp[v] + k;
13             openSet.insertOrDecreasePriority(w, sp[w] + h(w));
```

Example: A run of A*

Example 17.6

Consider the following four cities, which are unit distance away from the center and connected via four roads. We aim to go from s to t.

Let $h(v)$ be the euclidian distance of v from t.

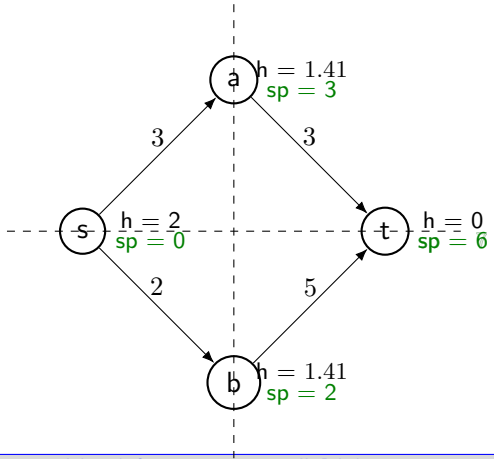
Initially: openSet $= \{s : 2\}$

After first iteration: openSet $= \{a : 4.41, b : 3.41\}$

After second iteration: openSet $= \{a : 4.41, t : 7\}$

After third iteration: openSet $= \{t : 6\}$

Last iteration: t is popped and algorithm ends



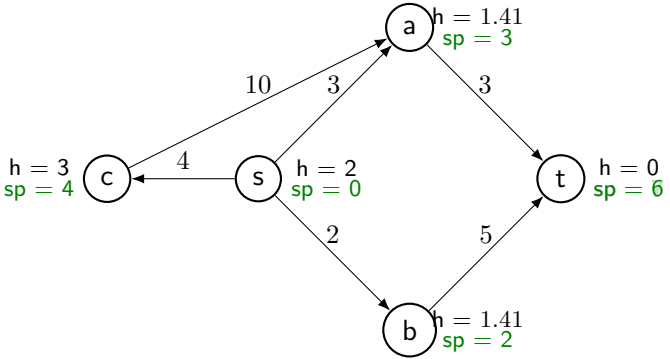
What is the advantage of A*?

It avoids unnecessary exploration.

Exercise 17.4

Consider the following road network with an extra city.

c will enter the openSet with priority 7 and will not be popped before t.



h makes $A*$ miss paths and it stops as soon as we find the target. Can we miss the shortest path?

Correctness of A*

Definition 17.7

h is admissible if $h(v)$ is less than or equal to the shortest path to the target for each v.

Theorem 17.2

If h is admissible, then A* will find the optimal path.

Proof.

Let us assume A* found a non-optimal path to the target t with length l'.

Let $v_0, ...v_n$ be the shortest path to t with length $l < l'$.

Let us suppose the path was not explored by A* beyond v_i.

v_i must have entered openSet and never left. ...

Correctness of A* (2)

Proof.

The priority of v_i in openSet is $sp[v_i] + h(v_i)$.

Since h is admissible, $sp[v_i] + h(v_i) \leq sp[v_i] + Length(v_{i+1}, ..., v_n) \leq l$.

Since t was deleted from openSet before v_i, the priority of v_i in OpenSet must be greater than l'.
Contradiction. □

Connection with Dijkstra's algorithm

A* is more general algorithm than Dijkstra's algorithm.

Dijkstra's algorithm is unguided but never visits a node again.

In A*, a vertex can leave openSet and enter again. When A* is like Dijkstra?

Definition 17.8

h is consistent if for each $(v, k, w) \in E$, $h(v) \leq h(w) + k$.

Theorem 17.3

If h is consistent, all vertex will enter openSet only once.

Exercise 17.5

a. Show that if $h(v) = 0$ for all v, A* is Dijkstra's algorithm.
b. Prove the above theorem.

Commentary: A* will be handled in-depth in the next semester in AI/ML course.

Topic 17.2

Tutorial problems

Example: Counting paths

Exercise 17.6

Modify Dijkstra's algorithm to compute the number of shortest paths from s to every vertex t.

Example: Negative edges

Exercise 17.7

Show an example of a graph with negative edge weights and show how Dijkstra's algorithm may fail. Suppose that the minimum negative edge weight is -d. Suppose that we create a new graph G' with weights w', where G' has the same edges and vertices as G, but w'(e)=w(e)+d. In other words, we have added d to every edge weight so that all edges in the new graph have edge weights non-negative. Let us run Dijkstra on this graph. Will it return the shortest paths for G?

Example: Road network

Exercise 17.8

Let G(V,E) be a representation of a geography with V as cities and (u,v) an edge if and only if there is a road between the cities u and v. Let d(u,v) be the length of this road. Suppose that there is a bus plying on these roads with fare f(u,v)=d(u,v). Next, suppose that you have a free coupon that allows you one free bus ride. Find the least fare paths from s to another city v using the coupon for this travel.

Exercise 17.9

Same as above. Suppose w(u,v) is the width of the road between the cities u and w. Given a path pi, the width w(pi) is the minimum of widths of all edges in pi. Given a pair of cities s and u, is it possible to use Dijkstra to determine d such that it is the largest width of all paths pi from s to v?

Exercise 17.10

Same as above. For a path pi, define hop(pi)=max(d(e)) for all e in pi. Thus if one is traveling on a motorcycle and if fuel is available only in cities, then hop(pi) determines the fuel capacity of the tank of your motorcycle needed to undertake the trip. Now, for any s and u, we want to determine the minimum of hop(pi) for all paths pi from s to u. Again, can Dijkstra be used?

Topic 17.3

Problems

True or False

Exercise 17.11

Mark the following statements True / False and also provide justification.

1. Dijkstra's algorithm does not work for the directed graphs with cycles.
2. Dijkstra's algorithm does not work for the graphs that have negative edges.

Example: Travel plan

Exercise 17.12

You are given a timetable for a city. The city consists of n stops V=v1,v2,...,vn. It runs m services s1,s2,...,sm. Each service is a sequence of vertices and timings. For example, the schedule for service K7 is given below. Now, you are at stop A at 8:00 a.m. and you would like to reach stop B at the earliest possible time. Assume that buses may be delayed by at most 45 seconds. Model the above problem as a shortest path problem. The answer should be a travel plan.

Example: Preferred paths

Exercise 17.13

Given a graph G(V,E) and a distinguished vertex s and a vertex v, there may be many shortest paths from s to v. Which shortest path is identified by Dijkstra?

End of Lecture 17