Lakehead UNIVERSITY

COMP 4433: Algorithm Design and Analysis

Dr. Y. Gu

Feb. 15, 2023 (Lecture 10)

Greedy Algorithm and Minimum Spanning Tree (MST)

Overview

- Define the problem of MST
- Review of some elementary graph algorithm
    - Breadth-First Search
    - Depth-First Search
- Greedy Algorithm for MST
- The algorithms of Kruskal and Prim

Definition of MST Problem

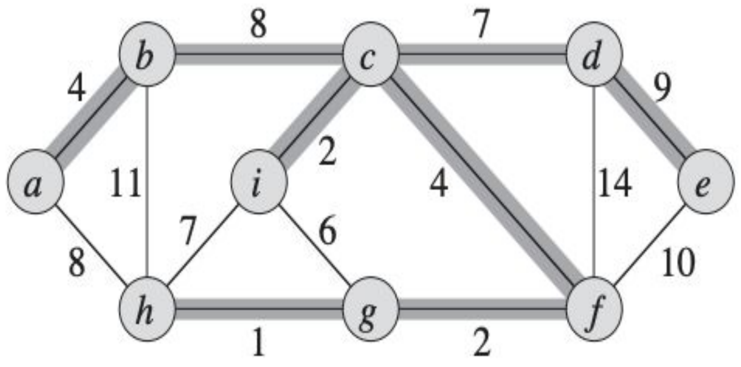
Let G = (V, E) be an undirected connected graph with a weight function: E → R. An acyclic set T ⊆ E that connects all of the vertices of G is called a spanning tree of G. We want to find T whose total weight

$$w(T) = \sum_{(u,v)\in T} w(u, v)$$

is minimum.

Such a problem is called minimum-spanning-tree problem.

Minimum Spanning Tree Example



The weights on edges are shown, and the edges in a minimum spanning tree are shaded. The total weight of the tree shown is 37. This minimum spanning tree is not unique: removing the edge (b, c) and replacing it with the edge (a, h) yields another spanning tree with weight 37.

Review Graph Algorithms

Representations of a graph – There are two representations of a graph.

For the adjacency-matrix representation of a graph G = (V, E), we assume that vertices are labeled as 1,2,...,|V|.

The representation is a |V|×|V| matrix A = (a_{ij}) such that

$$a_{ij} = \begin{cases} 1 & \text{if } (i,j) \in E, \\ 0 & \text{otherwise} \end{cases}$$

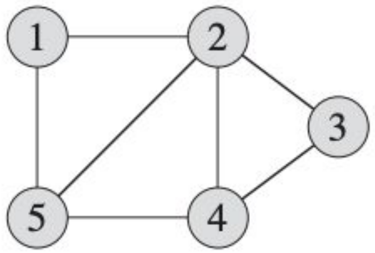
For weighted graph, instead of using 1 in the matrix, we can use $w(i,j)$ as a_{ij} if $(i,j) \in E$.

Review Graph Algorithms

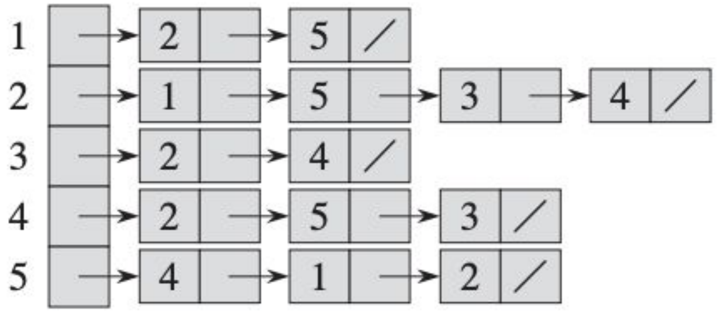
The adjacency-list representation of a graph G = (V, E) consists of an array adj of |V | lists, one for each vertex in V.

For each u ∈ V , the adjacency list *adj[u]* contains all the vertices adjacent to u in G. For weighted graph, we simply store the weight w(u, v) of the edge (u, v) with vertex v in u’s list.

Review Graph - Example



(a)



(b)

$$
\begin{array}{c|ccccc|}
 & 1 & 2 & 3 & 4 & 5 \\
\hline
1 & 0 & 1 & 0 & 0 & 1 \\
2 & 1 & 0 & 1 & 1 & 1 \\
3 & 0 & 1 & 0 & 1 & 0 \\
4 & 0 & 1 & 1 & 0 & 1 \\
5 & 1 & 1 & 0 & 1 & 0 \\
\hline
\end{array}
$$

(c)

An adjacency-list representation requires Θ(V + E) memory space, while an adjacency-matrix representation needs Θ(V^2) space.

Review Graph Algorithms

The Breadth-first search

Given a graph G(V,E) and a distinguished source vertex v, we consider search algorithms, which explore the edges of G to discover every vertex that is reachable from s.

The Breadth-first search procedure assumes that the input graph is represented using adjacency list. The algorithm constructs a breadth-first tree, initially containing only its root, which is the source vertex s. Whenever the search discovered a vertex v in the course of scanning the adjacency list of an already discovered vertex u, the vertex v and the edge (u, v) are added to the tree.

Review Graph Algorithms

The Breadth-first search (continue)

For each vertex u ∈ V , we define several attributes on it. u.π denote u's predecessor (in the breadth-first tree). If u has no predecessor, then u.π = NIL. The attribute u.d holds the distance from the source vertex s to vertex u.

The algorithm uses a FIFO queue Q. The attribute u.color gives a color to u to indicate if it is processed. The white color means it is not processed, the gray color means it is put into the queue, and the black color means it has been processed. The attribute u.d holds the distance from the source s to vertex u computed by the algorithm.

Review Graph Algorithms

The Breadth-first search (continue)

For each vertex u ∈ V , we define several attributes on it. u.π denote u's predecessor (in the breadth-first tree). If u has no predecessor, then u.π = NIL. The attribute u.d holds the distance from the source vertex s to vertex u.

The algorithm uses a FIFO queue Q. The attribute u.color gives a color to u to indicate if it is processed. The white color means it is not processed, the gray color means it is put into the queue, and the black color means it has been processed. The attribute u.d holds the distance from the source s to vertex u computed by the algorithm.

BFS - Pseudo Code

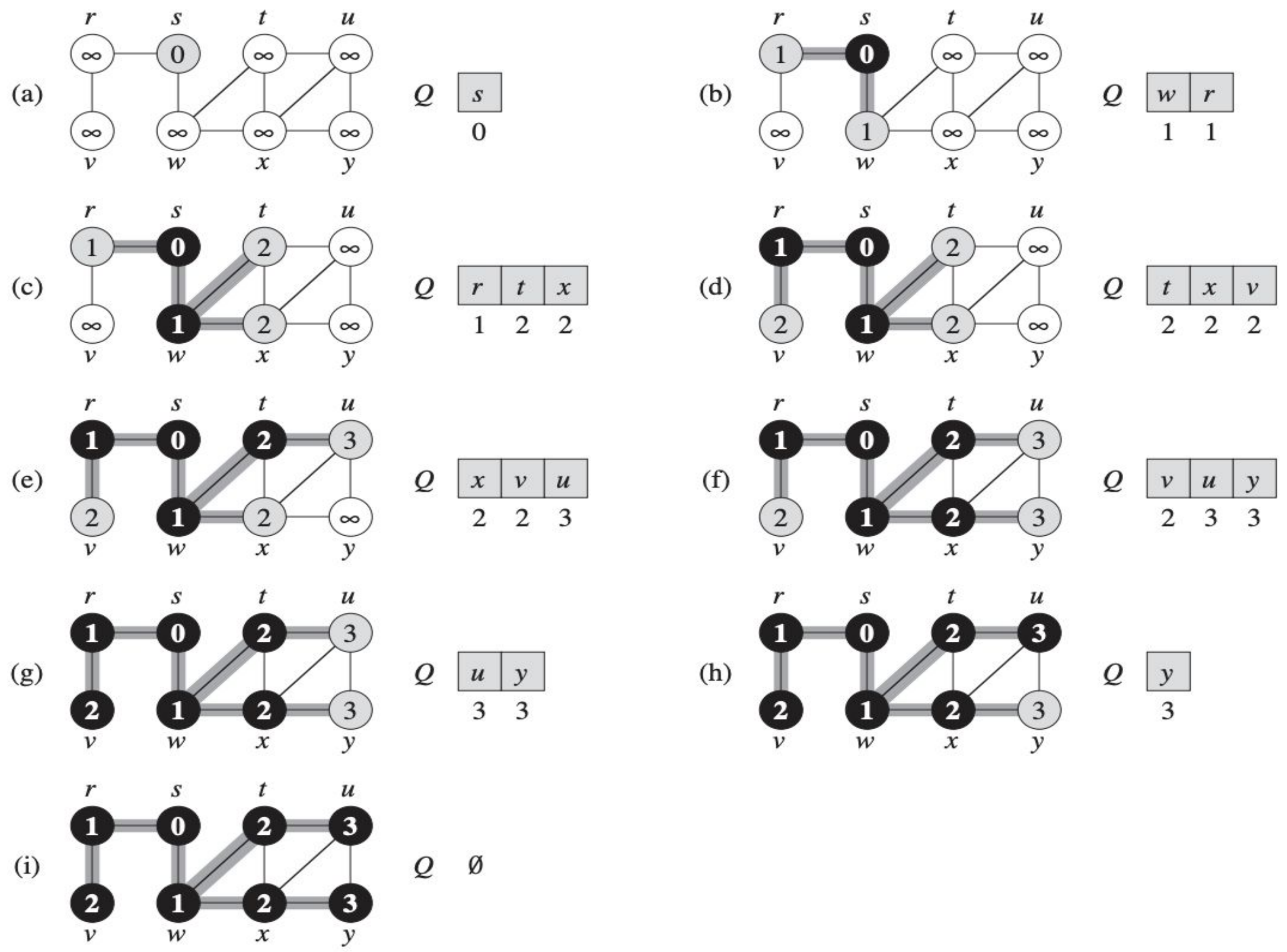
```
BFS($G, s$)
1   for each vertex $u \in G.V - \{s\}$
2       $u.color$ = WHITE
3       $u.d = \infty$
4       $u.\pi$ = NIL
5   $s.color$ = GRAY
6   $s.d = 0$
7   $s.\pi$ = NIL
8   $Q = \emptyset$
9   ENQUEUE($Q, s$)
10  while $Q \neq \emptyset$
11      $u$ = DEQUEUE($Q$)
12      for each $v \in G.Adj[u]$
13          if $v.color$ == WHITE
14              $v.color$ = GRAY
15              $v.d = u.d + 1$
16              $v.\pi = u$
17              ENQUEUE($Q, v$)
18      $u.color$ = BLACK
```

In this procedure, initialization uses O(V) time, the queue operation is also using O(V) time because each vertex goes to the queue once.

The total time spent in scanning adjacency lists is O(E).

The running time of BFS procedure is O(V + E). Thus, breadth-first search runs in time linear in the size of the adjacency-list representation of G.

BFS - Example



(a) Q: s — 0
(b) Q: w r — 1 1
(c) Q: r t x — 1 2 2
(d) Q: t x v — 2 2 2
(e) Q: x v u — 2 2 3
(f) Q: v u y — 2 3 3
(g) Q: u y — 3 3
(h) Q: y — 3
(i) Q: Ø
r s t u
v w x y

BFS Example

- With the exception of the source vertex s, lines 1–4 paint every vertex white, set u.d to be infinity for each vertex u, and set the parent of every vertex to be NIL.
- Line 5 paints s gray, since we consider it to be discovered as the procedure begins. Line 6 initializes s.d to 0, and line 7 sets the predecessor of the source to be NIL. Lines 8–9 initialize Q to the queue containing just the vertex s.
- The while loop of lines 10–18 iterates as long as there remain gray vertices, which are discovered vertices that have not yet had their adjacency lists fully examined. This while loop maintains the following invariant:
- At the test in line 10, the queue Q consists of the set of gray vertices.

Shortest Paths

- Breadth-first search finds the distance to each reachable vertex in a graph G =(V, E) from a given source vertex s∈V.
- Define the **shortest-path distance** δ(s,v) from s to v as the minimum number of edges in any path from vertex s to vertex v, if there is no path from s to v, then δ(s,v)=∞.
- We call a path of length δ(s,v) from s to v a **shortest path** from s to v.
- Before showing that breadth-first search correctly computes shortest-path distances, we investigate an important property of shortest-path distances.

Shortest Paths

Lemma 22.1: Let G = (V, E) be a directed or undirected graph, and let $s \in V$ be an arbitrary vertex. Then for any edge $(u, v) \in E$, $\delta(s, v) \leq \delta(s, u) + 1$.

Proof:

If u is reachable from s, then so is v. In this case, the shortest path from s to v cannot be longer than the shortest path from s to u followed by the edge (u,v), and thus the inequality holds. If u is not reachable from s, then $\delta(s, u) = \infty$, and the inequality holds.

Shortest Paths

Lemma 22.2: Let G = (V, E) be a directed or undirected graph, and suppose that BFS is run on G from a given source $s \in V$. Then upon termination, for each vertex $v \in V$, the value v.d composed by BFS satisfies $v.d \geq \delta(s, v)$.

Proof:

We use induction on the number of Enqueue operations. Our inductive hypothesis is that $v.d \geq \delta(s, v)$ for all $v \in V$. The basis of the induction is the situation immediately after enqueuing s in BFS. The inductive hypothesis holds here, because

$s.d = 0 = \delta(s, s)$ and $v.d = \infty \geq \delta(s, v)$ for all $v \in V - \{s\}$.

■

Shortest Paths

Proof (continue):

For the inductive step, consider a white vertex v that is discovered during the search from a vertex u. The inductive hypothesis implies that u.d ≥ δ. From the assignment performed by line 15 and from Lemma 22.1, we obtain

$$v.d = u.d + 1 \geq \delta(s, u) + 1 \geq \delta(s, v).$$

Vertex v is then enqueued, and it is never enqueued again because it is also grayed and the then clause of lines 13 - 17 is executed only for white vertices. Thus, the value of v.d never changes again, and the inductive hypothesis is maintained.

■

Shortest Paths

Lemma 22.3: Suppose that during the execution of BFS on a graph G = (V, E), the queue Q contains the vertices $\langle v_1, v_2, \ldots, v_r \rangle$, where v_1 is the head of Q and v_r is the tail. Then $v_r.d \le v_1.d + 1$ and $v_i.d \le v_{i+1}.d$ for i = 1, 2, …, r − 1.

Proof: The proof is by induction on the number of queue operations.

Initially, when the queue contains only s, the lemma certainly holds. For the inductive step, we must prove that the lemma holds after both dequeuing and enqueuing a vertex. If the head v1 of the queue is dequeued, v2 becomes the new head. (If the queue becomes empty, then the lemma holds vacuously.)

Shortest Paths

Proof (Continue): By the inductive hypothesis, $v_1.d \le v_2.d$. But then we have $v_r.d \le v_1.d + 1 \le v_2.d + 1$, and the remaining inequalities are unaffected. Thus, the lemma follows with v_2 as the head.

When we enqueue a vertex v in line 19 of BFS, it becomes v_{r+1}. At that time, we have already removed vertex u, whose adjacency list is currently being scanned, from the queue Q, and by the inductive hypothesis, the new head $v_1.d \ge u.d$. Thus, $v_{r+1}.d = v.d = u.d + 1 \le v_1.d + 1$. From the inductive hypothesis, we also have $v_r.d \le u.d + 1$, and so $v_r \le u.d + 1 = v.d = v_{r+1}.d$, and the remaining inequalities are unaffected.

Thus, the lemma follows when v is enqueued. ■

Shortest Paths

Corollary 22.4

Suppose that vertices v_i and v_j are enqueued during the execution of BFS, and that v_i is enqueued before v_j. Then $v_i.d \leq v_j.d$ at the time that v_j is enqueued.

Proof: Immediate from Lemma 22.3 and the property that each vertex receives a finite d value at most once during the course of BFS. ■

Shortest Paths

Theorem 22.5 (Correctness of breadth-first search)

Let G = (V, E) be a directed or undirected graph, and suppose the BFS is run on G from a given source vertex s ∈ V. Then during its execution, BFS discovers every vertex v ∈ V that is reachable from the source s, and upon termination, v.d = δ(s,v) for all v ∈ V .

Moreover, for any v≠ s that is reachable from s, one of the shortest paths from s to v is a shortest path from s to v.**π** followed by the edge (v.**π** , v).

Shortest Paths

Proof: Assume, for the purpose of contradiction, that some vertex receives a d value not equal to its shortest-path distance. Let v be the vertex with minimum δ(s, v) that receives such an incorrect d value; clearly v ≠ s.

By Lemma 22.2, v.d ≥ δ(s, v), and thus we have that v.d > δ (s, v). Vertex v must be reachable from s, for if it is not, then δ(s, v) = ∞ ≥ v.d. Let u be the vertex immediately preceding v on a shortest path from s to v, so that δ(s, v) = δ(s, u) + 1. Because δ(s, u) < δ(s, v), and because of how we chose v, we have u.d= δ(s, u). Putting these properties together, we have

$$v.d \quad > \quad \delta(s,v) = \delta(s,u) + 1 = u.d + 1. \qquad (2)$$

Shortest Paths

Proof (Continue): Now consider the time when BFS chooses to dequeue vertex u from Q. At this time, vertex v is either white, gray, or black. We shall show that in each of these cases, we derive a contradiction to inequality (2).

If v is white, then line 15 sets v.d = u.d + 1, contradicting inequality (2).

If v is black, then it was already removed from the queue and, by Corollary, we have v.d ≤ u.d, again contradicting inequality (2).

If v is gray, then it was painted gray upon dequeuing some vertex w, which was removed from Q earlier than u and for which v.d = w.d + 1.

By Corollary, however, w.d ≤ u.d, and so we have v.d = w.d + 1 ≤ u.d + 1, once again contradicting inequality (2).

Thus we conclude that v.d = δ(s, v) for all v ∈ V.

Shortest Paths

Proof (Continue):

All vertices v reachable from s must be discovered, for otherwise they would have $\infty = v.d > \delta(s, v)$. To conclude the proof of the theorem, observe that if $v.\pi = u$, then $v.d = u.d + 1$.

Thus, we can obtain a shortest path from s to v by taking a shortest path from s to $v.\pi$ and then traversing the edge $(v.\pi, v)$.

■

After Class

- After class: Part VI 22.1 and 22.2