MST

Sam McCauley

March 4, 2025

Welcome Back!

- Greedy algorithms handout posted on the website
- We'll talk about Midterm 1 momentarily
- Also scheduling Midterm 2 and the Final
- Any other questions?

Schedules and Updates

Midterm 2: Date Change

- Midterm 2 will be on April 28 (not April 24)
- No class on May 1
- Syllabus, website updated. Please update your calendars!

Final Exam

- May 25th at 1:30 PM
- Very late! On a Sunday
- Let me know early if you have a conflict
- I think it's likely I'll have two finals: one very early (likely during reading period), and one on May 25th. But only those! So make sure you prepare ahead of time with any conflicts.
- If I do this, the finals will have completely separate contents and grading; I'll curve if there are significant grade discrepancies

Midterm 1

Midterm 1 Scheduling

- During your class time *a week from today;* 75 minutes
- No computer/books/internet/etc.
- Let me know now if you need an accommodation
- My goal: time should not be a significantly limiting factor in the midterm
- We will do a review session on Thursday; come with questions
    - Answers to any assignment questions
    - Review of anything we've seen in class
    - Other questions/ideas you may have had
    - Questions about what will be covered are also OK

What the Midterm Will Look Like

- Some problem set problems were too long and open-ended and will not be on the midterm
  - Shorter proofs from assignments that follow more directly are possible (think Problem 1 from Assignment 2 about proving that a hospital and student who are each other's top rank is in every maximum matching)
  - Somewhat less-open-ended proofs asking for an algorithm (etc.) also work. Partial credit for writing significant portions of the correct answer.
  - Probably total $\approx$ 2 questions that are on the longer side/ask for proofs
- Another $\approx$ 3 short questions: multiple choice or true false/asking for running times/asking for counterexamples or a short explanations
- These are roughly similar in scope to what you've seen on the Homework. Less mechanical—more about testing knowledge
- Make sure you know things like running time, basic properties of algorithms we've seen

Midterm Topics

- Basics of running time, correctness, big-O/Ω/Θ notation
- Gale-Shapley and stable matchings
- BFS/DFS and properties (incl. Topological sort)
- Greedy algorithms
- Dijkstra’s algorithm
- Minimum spanning trees

Practice Midterm Posted

- On GLOW. Please don't distribute it
- Gives a *rough* idea of what the midterm will look like
- Not as polished! I took the better problems for the actual midterm
- Doesn't cover exactly all the topics (e.g. no Dijkstra's on the practice midterm but might be on the actual midterm)
- Version with solutions posted soon!

Dijkstra’s Practice/Wrapup

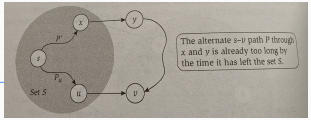
Dijkstra's Algorithm Implementation Pseudocode

```
function Dijkstra(Graph, source):
    for all v:
        initialize dist[v] ← ∞ and prev[v] ← 0
    dist[source] ← 0
    add source to Q
    while Q is not empty:
        remove u with minimum priority from Q
        dist[u] ← priority of u in Q
        for each neighbor v of u with dist[v] = ∞:
            if v is in Q:
                alt ← dist[u] + Graph.Edges(u, v)
                if alt < current priority of v:
                    reduce priority of v in Q to alt
                    prev[v] ← u
            if v is not in Q:
                add v to Q
                prev[v] ← u
    return dist[], prev[]
```

Negative Edge Weights

- Why doesn't Dijkstra's algorithm work if edge weights are negative?
- Definitely can't have any *cycles* whose total weight is negative. (Why is that?)
- Let's look at the proof to give us a hint

Reminder: Dijkstra's Algorithm Inductive Step



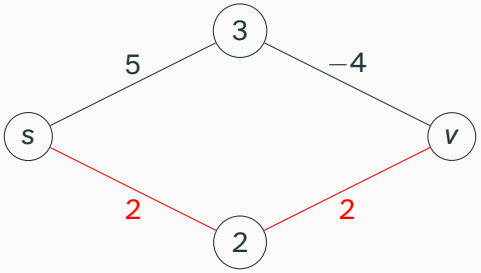
- Assume: after k vertices are finished, for all finished vertices w, $d[w] = d(s, w)$
- We find the edge $e = (u, v)$ with u finished and v unfinished that minimizes $d[u] + w_e$; mark v finished; set $d[v] = d[u] + w_e$. To show: $d[v] = d(s, v)$
- We have: for *any* fringe vertex y, $d(s, y) \geq d[u] + w_e$
- Now: there cannot be a path p' to v with length less than $d[u] + w_e$
  - Idea: assume contrary. Let y be the first vertex in p' not finished. Then the length of p' is at least $d(s, y) + d(y, v)$
  - $d(s, y) \geq d[u] + w_e$ from claim above; $d(y, v) \geq 0$ **since all edge weights are positive**

Negative Edge Weights

- Why doesn't Dijkstra's algorithm work if edge weights are negative?
- Definitely can't have any *cycles* whose total weight is negative. (Why is that?)
- If there are negative edge weights, then the alternate path might be better! Take a larger cost to get to the fringe, then a negative path to recover that cost to get to our vertex
- **In pairs:** come up with a graph with negative edges where Dijkstra's algorithm fills in an incorrect distance (there are small graphs that work)

Example

We want an example where Dijkstra's finds some path to some vertex v. Consider the red edges in the following example.



The Dijkstra's proof says that it would never be better go to from s to v through vertex 3, since by the time we get to 3 we already have length longer than 4. But with a negative edge, we can make up for the initial long edge using a subsequent negative edge.

Dijkstra's algorithm sets $d[v] = 4$, but $d(s, v) = 1$.

Minimum Spanning Trees

Minimum Spanning Tree (MST)

- A “greedy” graph algorithm
- How many of you have seen a minimum spanning tree algorithm before?
- We’ll see two, and talk about MST structure

Moravia

MST Problem Definition

Given a connected undirected graph G with *edge weights* w_e, a spanning tree is a set of edges $T \subseteq E$ such that:

- T is a spanning tree: T is a tree that connects all vertices, and
- T has minimum weight: for any spanning tree T',

$$\sum_{e \in T} w_e \leq \sum_{e \in T'} w_e.$$

In this class we will assume that *all edge weights are distinct*. It just makes the proofs simpler; the algorithms we'll see (Prim's and Kruskal's algorithm) work without this assumption.

Building to an MST Algorithm

- Can we create an optimal MST on one vertex?
    - How about on two vertices?
    - What if I start with a vertex s. Can I build the MST starting from s?
- Idea: add minimum weight neighbor of s to the tree
- Intuition as to why this is optimal?

Prim's Algorithm (Jarník's Algorithm)

First, choose a starting vertex u. Create a set of vertices, starting with $S \leftarrow \{u\}$ and a tree starting with $T \leftarrow \emptyset$. (S is essentially the set of "finished vertices.")

- Can we find the best "fringe vertex" as we did in Dijkstra's algorithm?
- One idea: take the cheapest *edge*. (Not total path!)
- Formally: While $|T| \leq n - 1$, find the min-cost edge $e = (u, v)$ such that one end $u \in S$ and $v \in V \setminus S$. Set $T \leftarrow T \cup \{e\}$ and $S \leftarrow S \cup \{v\}$.

Let's do an example [Blackboard]

First, how can we prove correctness? (Then we'll discuss how to find e efficiently, and the running time.)

Cut Property of MST

A *cut* is a partition of the vertices V into two subsets: S, and $V \setminus S$. A *cut edge* is an edge with one endpoint in S and the other in $V \setminus S$.

Lemma

Let G be a graph where all edge weights are distinct. For any cut S, let $e = (u, v)$ be the minimum weight cut edge. Then e is in every minimum spanning tree of G.

Side note: this lemma implies that there is actually only one minimum spanning tree if all edge weights are distinct.

Cut Property of MST: Proof

Lemma

For any cut S, let $e = (u, v)$ be the minimum weight cut edge. Then e is in every minimum spanning tree of G.

Proof: Assume the contrary: there is an MST T such that $e \notin T$.

There must be some path p from u to v in T. Let $e' = (u', v')$ be the first cut edge in p. Let's draw a diagram [Blackboard]

Consider the set T' created by removing e' from T and adding e. Therefore, T' has smaller weight than T. We claim that T' is a spanning tree: for any two vertices x, y there is a path from x to y in T'. Note that T is a spanning tree, so there must be some path q from x to y in T.

If q does not contain e' then we are done, since all edges in q are in T'.

Otherwise, the path from x to y in T contains e'. (Continued on next slide)

Cut Property of MST: Proof

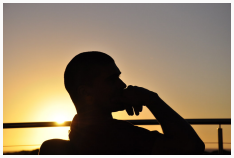
Lemma

For any cut S, let $e = (u, v)$ be the minimum weight cut edge. Then e is in every minimum spanning tree of G.

Proof continued: Recall that we have a minimum cut edge $e = (u, v)$ and an MST T with $e \notin T$. We took the first cut edge $e' = (u', v')$ on the path p from u to v in T. We defined $T' = T \setminus \{e'\} \cup \{e\}$. We now want to show that for any two vertices x and y connected by a path q in T with $e' \in q$, x and y are connected by edges in T'.

We can decompose q into three parts: first a path from x to u', then e', then a path from v' to y. We replace e' with a path from u' to v' using only edges in T': we can go from u' to u (along p), then across e, then from v' to v (again along p). This replacement gives a path from x to y and we are done.

Proving Prim's Correct



- **In pairs:** How can we use the cut property to prove Prim's algorithm correct?

- Answer: Every edge we add is the smallest cut edge between S and $V \setminus S$; by the cut property it is in every MST.

Implementing Prim's Algorithm

What do we need to be able to do?

- Maintain all cut edges!
- Must be able to insert new edges when adding a vertex to S
- Must be able to find minimum-weight cut edge (i.e. minimum-weight edge in the data structure) and remove it
- Priority Queue!
- Note that: we will (again!) wind up with some edges from S to S in the data structure (why?). If we remove such an edge we'll just skip it.
    - Reason: when we add v to S, there could be some extra edges to v already in the priority queue

Prim's Algorithm (Jarník's Algorithm)

First, choose a starting vertex u. Create a set of vertices, starting with $S \leftarrow \{u\}$ and a tree starting with $T \leftarrow \emptyset$. Add all edges adjacent to u to a priority queue PQ.

While $|T| \leq n - 1$, find the min-cost edge $e = (u, v)$ such that one end $u \in S$ and $v \in V \setminus S$. Set $T \leftarrow T \cup \{e\}$ and $S \leftarrow S \cup \{v\}$.

To implement: each time we add a vertex to S, add its incident edges to PQ. To find the minimum cut edge, remove edges from PQ until we find a cut edge. Cost?

Need to do $\leq 2m$ inserts, and $\leq 2m$ extract mins (why?).

Running time: $O(m \log m)$.

As with Dijkstra's, can use Fibonacci heap to improve to $O(m + n \log n)$.

Prim's Algorithm Pseudocode

```
1  Prims(G):
2      pick a vertex v as the starting vertex
3      let T be an empty set and pq be an empty priority queue
4      add v to T; mark v as finished
5      for each edge e′ adjacent to w:
6          pq.insert(e′)
7
8      while pq is not empty:
9          e = pq.removeMin()
10         if e has an unvisited endpoint w:
11             add w to T; mark w as visited
12             for each edge e′ adjacent to w:
13                 pq.insert(e′)
```

MST