

# Advanced Algorithms Notes

Alberto Lazari

II Semester – 2023

## Index

|                                                |   |
|------------------------------------------------|---|
| 1 – DFS (Depth First Search) .....             | 3 |
| 1.1 – Applications .....                       | 3 |
| 2 – BFS (Breadth First Search) .....           | 3 |
| 3 – MST (Minimum[-weight] Spanning Tree) ..... | 3 |
| 3.1 – Prim .....                               | 3 |
| 3.1.1 – Prim with heap .....                   | 3 |
| 3.2 – Kruskal .....                            | 4 |
| 3.2.1 – Kruskal with disjoint sets .....       | 4 |
| 4 – SS (Single-Source) Shortest Paths .....    | 4 |
| 4.1 – Non-negative weights – Dijkstra .....    | 4 |
| 4.1.1 – Dijkstra with heap .....               | 4 |
| 4.2 – General case – Bellman-Ford .....        | 5 |
| 5 – AP (All Pair) Shortest Paths .....         | 5 |
| 5.1 – Floyd-Warshall .....                     | 5 |
| 6 – Maximum flows .....                        | 5 |
| 6.1 – Definitions .....                        | 5 |
| 6.2 – Ford-Fulkerson .....                     | 6 |
| 7 – NP-hardness .....                          | 6 |
| 7.1 – Reduction .....                          | 6 |
| 7.2 – NP-hard Problems .....                   | 6 |
| 8 – Approximation algorithms .....             | 7 |
| 8.1 – Vertex cover .....                       | 7 |
| 8.2 – Metric TSP .....                         | 7 |
| 8.2.1 – Eulerian circuit approach .....        | 7 |
| 8.3 – Set cover .....                          | 7 |

|                                 |   |
|---------------------------------|---|
| 9 – Randomized algorithms ..... | 7 |
| 10 – Minimum cut – Karger ..... | 8 |
| 10.1 – Algorithm .....          | 8 |
| 10.2 – Definitions .....        | 8 |
| 10.2.1 – Multigraphs .....      | 8 |
| 10.2.2 – Full Contraction ..... | 8 |
| 10.3 – Karger-Stein .....       | 8 |
| 10.4 – 2020 version .....       | 8 |
| 11 – Chernoff bounds .....      | 9 |
| 11.1 – Variants .....           | 9 |

## # Lecture 2

# 1 – DFS (Depth First Search)

**Complexity**  $O(n + m)$

## 1.1 – Applications

Derived using DFS (or BFS) in  $O(n + m)$

- Path between source vertex  $s$  to arbitrary  $t$ : add a parent field to vertices.  
When  $t$  is found return the parents backtrace
- Find cycle: use parent field on vertices and ancestor on edges
- Connected components:
  1. run DFS (or BFS)  $n$  times
  2. Keep a counter  $k$  to increment on every “untouched” source vertex
  3. Assign  $k$  to  $v$ .id, instead of 1  $\rightarrow$  label vertexes based on its component
  4. If at the end  $k > 1$ , then multiple components were found

## # Lecture 3

# 2 – BFS (Breadth First Search)

**Complexity**  $O(n + m)$

# 3 – MST (Minimum[-weight] Spanning Tree)

MST  $(G = (V, E), s)$

Tree created from a source vertex  $s$ , the root of the tree

## # Lecture 4

## 3.1 – Prim

**Complexity**  $O(m \cdot n)$

Make cuts to separate a growing set  $A$  (initialized to  $\{s\}$ ), and find *light edges*. Add the light edge found with the cut to  $A$  and repeat, until you have a tree (no more vertices outside  $V \setminus A$ )

The search for the light edge is  $O(m)$  and is repeated  $n$  times, but it can be optimized

### 3.1.1 – Prim with heap

**Complexity**  $O(m \log n)$

Use a heap to store vertices, ordered on their cost to reach from a vertex already processed (light edge that crosses the cut) For every vertex that you

put in  $A$  (actually that you extract from the heap  $H$ ) check if you can update the cost of the vertices still in  $H$

In order to keep trace of the actual edges, instead of the vertices, it's needed to save the parent of every vertex you update

The complexity is actually  $O(n \log n + m \log n)$ , but graph  $G$  is connected  
 $\Rightarrow m \geq n - 1$

## # Lecture 5

### 3.2 – Kruskal

**Complexity**  $O(m \cdot n)$  (when implemented with adjacency list, because of frequent cycle checks)

Extremely simple:

1.  $A$  is an empty forest;
2. Sort  $E$  by weight (ascending order);
3. If adding  $e \in E$  to  $A$  keeps it a forest (doesn't introduce cycles) add it

#### 3.2.1 – Kruskal with disjoint sets

**Complexity**  $O(m \log n)$  (same of Prim with heap)

Use union-find data structure: connected components are disjoint sets to join in  $O(\log n)$  time. Finds if a node is in a set in  $O(\log n)$  time  $\Rightarrow$  cycle checks in logarithmic time

It's still an open problem to find MST implementation in  $O(m)$

## # Lecture 7

### 4 – SS (Single-Source) Shortest Paths

SSSP ( $G = (V, E), s \in V$ ), where  $G$  directed, weighted graph

Returns:  $\text{len}(v) = \text{dist}(s, v), \forall v \in V$

#### 4.1 – Non-negative weights – Dijkstra

**Complexity**  $O(m \cdot n)$

Similar to Prim:

1. Growing region (vertices set)  $X = \{s\}$
2. Select minimum-weight vertex  $e$  between  $X$  and  $V \setminus X : e = (u, v)$ , where  $u \in X$  and  $v \notin X$
3. Add  $v$  to  $X$  and set  $\text{SP}(v) = \text{SP}(u) + w(e)$

##### 4.1.1 – Dijkstra with heap

**Complexity**  $O((m + n)\log n)$

Similar to Prim implementation with heaps

# Lecture 8

## 4.2 – General case – Bellman-Ford

**Complexity**  $O(m \cdot n)$

Need to forbid negative cycles in shortest paths, they lead to infinitely small paths → doesn't even make sense to speak about shortest paths

Bellman-Ford returns either SSSP  $(G, s)$  or a declaration that  $G$  has a negative cycle

Refine every shortest path every iteration (check every edge). In  $n - 1$  iterations it reaches a fix-point. If it doesn't it means a negative cycle exist

In 2022 a **near-linear** algorithm was found

## 5 – AP (All Pair) Shortest Paths

Returns:  $\text{dist}(v, u), \forall v, u \in V$

Running Bellman-Ford  $n$  times have complexity  $O(m \cdot n^2)$ . With dynamic programming complexity can be reduced up to  $O(n^3 \log n)$

### 5.1 – Floyd-Warsh

**Complexity**  $O(n^3)$

Iterate on 3 vertices  $u, v, k \in V$  in 3 nested loops, testing whether using  $k$  in the path is better

To catch negative cycles it's sufficient to check that  $\text{dist}(v, v) \geq 0, \forall v \in V$

# Lecture 10

## 6 – Maximum flows

### 6.1 – Definitions

**Flow network** graph where edges have a capacity  $c : E \rightarrow \mathbb{R}^+$ .

A source  $s$  and a sink  $t$  are specified

**Flow**  $f : E \rightarrow \mathbb{R}^+, |f| = \sum_{(s,v) \in E} f(s, v)$ , basically the flow on the first edges

Flow is conserved through the graph and has to be  $\leq$  than capacity for all edges

## 6.2 – Ford-Fulkerson

**Complexity**  $O(m \cdot |f^*|)$ , where  $|f|$ : maximum flow

# Lecture 11

## 7 – NP-hardness

Similar polynomial and NP-hard problems:

- Eulerian vs Hamiltonian circuit: cycle traversing every edge ( $O(n)$ ) vs vertex (NP-hard) only once
- MST vs TSP: give paths to (spanning tree,  $O(m \log n)$ ) vs a tour between (NP-hard) all vertices, minimizing the sum of the weights of the edges used
- Class P: Polynomial time problems
- Class NP: Non-deterministic Polynomial
- Class NP-hard: if proving a problem polynomial would mean all NP is polynomial it's NP-hard

### 7.1 – Reduction

$A < B \rightarrow B$  is used to solve A

$A <_p B \rightarrow A$  reduces to B in polynomial time: a polynomial algorithm exists to convert an input instance for A in one for B that is then used to solve A

if A is NP-hard and  $A <_p B \implies B$  is NP-hard

### 7.2 – NP-hard Problems

- **SAT**: first NP-hard proved, by Cook-Levin theorem
- **3-SAT**:  $\text{SAT} <_p \text{3-SAT}$
- **Maximum Independent Set**:  $\text{3-SAT} <_p \text{MIS}$  (maximum number of vertices with no edge between them)
- **Hamiltonian circuit**
- **TSP** (Traveling Salesperson Problem):  $\text{Hamiltonian circuit} <_p \text{TSP}$
- **Metric TSP**: TSP with triangular inequality on paths (direct paths are always shorter than the ones using other vertices)
- **Maximum clique**: largest complete sub-graph
- **Minimum vertex cover**: minimum number of vertices that “touches” all edges
- **Minimum set cover**:  $\text{vertex cover} <_p \text{set cover}$  (minimum number of subsets to cover an original set)

# Lecture 12

## 8 – Approximation algorithms

### 8.1 – Vertex cover

**Complexity**  $O(n + m)$

**Approximation factor** 2

**Matching** set of edges with no common vertex

### 8.2 – Metric TSP

**Complexity**  $O(m \log n)$

**Approximation factor** 2 (tight)

Build an MST with Prim/Kruskal and return the full preorder chain (DFS with pre and post visits (with repetitions)) of the tree

#### 8.2.1 – Eulerian circuit approach

**Complexity** polynomial

**Approximation factor**  $2 / 3$

Find a minimum weight perfect matching between odd-degree vertices and add those edges to the MST. Now the graph has all vertices with even degree  $\Rightarrow$  it is Eulerian

Return the Eulerian cycle of the graph

A  $3 / 2 - \varepsilon$  approximation has been found, where  $\varepsilon = 10^{-36}$

# Lecture 17

### 8.3 – Set cover

**Complexity**  $O(n \cdot |F| \cdot \min\{n, |F|\})$ , where  $n = |X|$  (cubic)

**Approximation factor**  $\lceil \log_2 n \rceil + 1 = \Theta(\log n)$

Variables:

- $X$ : original set, with all possible elements
- $F$ : set of subsets of  $X$

Greedy algorithm on subset in  $F$  with most elements in  $X$ . At each step select the subset and remove its elements from  $X$  and repeat

## 9 – Randomized algorithms

- Las Vegas: always correct (randomized quicksort)
- Monte Carlo: may return wrong values, though high probability of correct result
  - One sided: decision problems give only false positives/negatives

- Two sided: decision problems may fail in any case

## # Lecture 19

**High probability** algorithm  $A_\Pi$  for problem  $\Pi$

has complexity  $f(n)$  / is correct

with high probability if

$$\exists c, d > 0. \Pr (A \text{ has complexity} > cf(n)) / \Pr (A \text{ is not correct}) < \frac{1}{n^d}$$

## 10 – Minimum cut – Karger

**Complexity**  $O(n^4 \log n)$

Minimum number of edges to remove, in order to disconnect the (multi)graph

### 10.1 – Algorithm

Repeat *Full Contraction*  $k$  times, to reduce error

Karger returns the minimum with *high probability* ( $\Pr (\text{fail}) < \frac{1}{n^d}$ ) with

$$k = \frac{dn^2 \ln n}{2} = \Theta(n^2 \log n)$$

### 10.2 – Definitions

#### 10.2.1 – Multigraphs

**Multiplicity**  $m : \mathbb{S} \rightarrow \mathbb{N}, m(e) = \text{occurrences of an element } e \in \text{multiset } \mathbb{S}$

$\mathcal{G} = (\mathcal{V}, \mathcal{E})$  is a multigraph, where  $\mathcal{E}$  is a multiset

#### 10.2.2 – Full Contraction

**Complexity**  $O(n^2)$

Choose a random edge and contract on it, until two vertices remain

**Contraction** contract a graph  $\mathcal{G}$  on edge  $(u, v) \in \mathcal{E}$  (join vertices of the edge):

- Delete  $u$
- Delete all edges between  $u$  and  $v$
- Move all edges of  $u$  to  $v$

### 10.3 – Karger-Stein

**Complexity**  $O(n^2 \log^3 n)$

Avoids first  $\frac{n}{\sqrt{2}}$  iterations

### 10.4 – 2020 version



**Complexity**  $O(m \log n)$

# Lecture 21

## 11 – Chernoff bounds

Upper bounds on probability of the value of a variable  $X = \sum_{i=1}^n X_i$

$$\Pr (X > (1 + \delta)\mu) < \left( \frac{e^\delta}{(1 + \delta)^{1+\delta}} \right)^\mu$$

$\forall \delta > 0, \mu = E[X]$

### 11.1 – Variants

- $\Pr (X < (1 - \delta)\mu) < e^{\frac{-\mu\delta^2}{2}}$ , when  $0 < \delta \leq 1$
- $\Pr (X > (1 + \delta)\mu) < e^{\frac{-\mu\delta^2}{2}}$ , when  $0 < \delta \leq 2e - 1$