LECTURE 8

Last Class: Dijkstra's algorithm for shortest paths

→ Algorithm

→ Proof of Correctness.

DIJKSTRA's ALGORITHM

Let S = Set of explored/discovered nodes. → $\boxed{d(s,u)}$

(For each vertex $u \in S$, we also store $d(u)$).

① $S = \{s\}$ and $d(s) = 0$. $d(v) = \infty \ \forall \ v \neq s$. } $O(|V|)$

Parent$(v) = \phi \ \forall \ v \in S$

→ $\underline{\underline{d'(v) = \infty \ \forall \ v \neq S}}$

② While $S \neq V$

|V| times

ⓐ For each vertex $v \notin S$ takes ← $O(\text{degree}(v))$

set $d'(v) = \min\{d(u) + l_{u,v} : u \in S \text{ and } (u,v) \text{ is an edge}\}$

ⓑ Find the vertex v with least $d'(v)$.

(i) ADD v to S.

(ii) $d(v) = d'(v)$.

→ Step (a) of the while loop takes
degree(v) time for each vertex $v \notin S$.

→ Total-time for step(a) would be

$$\sum_{v \notin S} \text{degree}(v) = O(|E|).$$

→ Step(b): We have to pick the vertex v with least d' among all vertices not in S.

→ Takes $O(|V|)$ time.

Time for each iteration of the while loop is

$$O(|V| + |E|).$$

$\Rightarrow$ **Total run-time is $O(|V| \cdot (|V|+|E|))$.**

---

① We don't have to recompute d' for every vertex after we update the set S.
↳ You only have to update for neighbors of the newly added vertex.

② We could use some "data structures" to compute min efficiently.

You can use Heaps / Priority Queues to get a run-time of

- $O(|V| + |E| \cdot \log |V|)$. (see [KT] if curious ...).

→ Greedy algorithms: You update solution gradually.

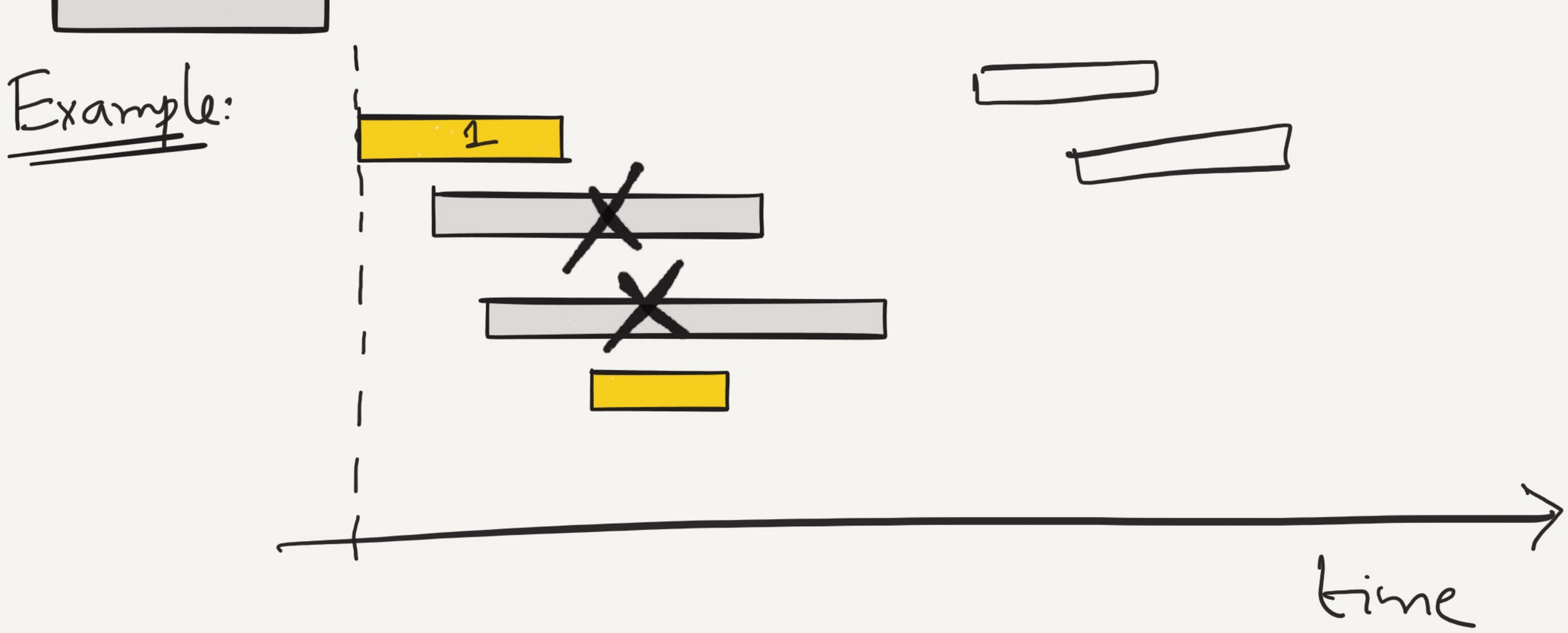
→ Analyzing greedy algorithms: "Always stays ahead".

2nd Example of Greedy algorithms

Interval Scheduling:

Server ← Requests for processing jobs $(s(i), f(i))$

Example:



Input: Sequence of jobs:

$(s(1), f(1))$, $(s(2), f(2))$, . . . , $(s(n), f(n))$

Output: Pick the maximum number of "non-conflicting" jobs/requests.

How to solve?

→ Use a **simple rule** to select the first job

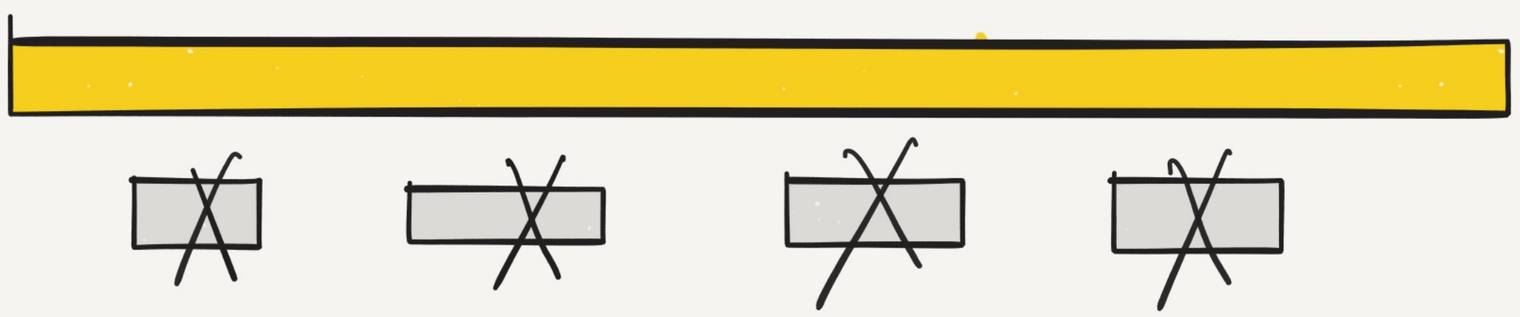
→ Remove conflicting jobs

→ Repeat until no more jobs left.

Which rule can we use??

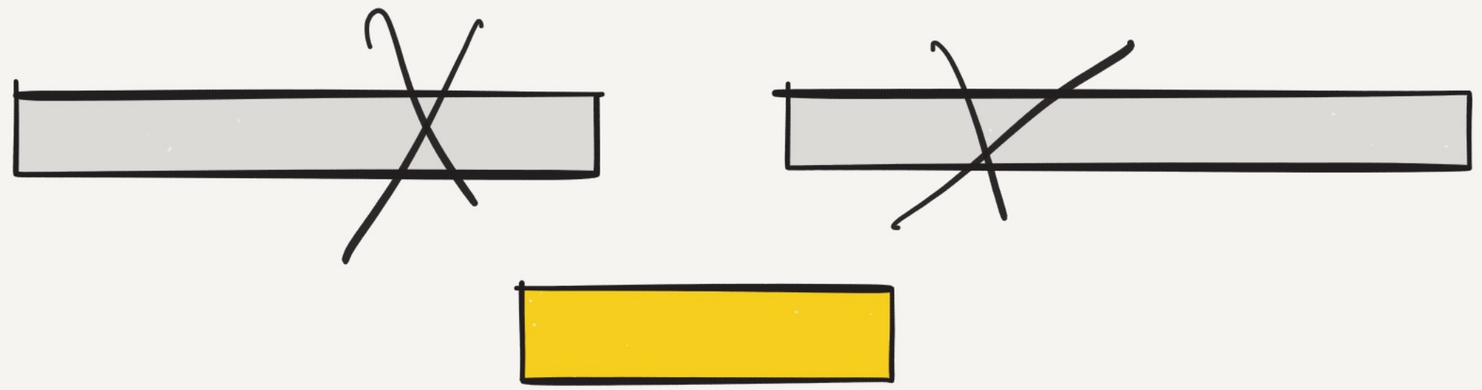
1. Pick the job with least finish time
2. Pick " " that conflicts with the least number of ones.
3. Pick the job with "least time". → (f(i) - s(i)).
4. Pick the job with least start time.

× (4) Pick the one with least start time.



} total # assigned = 1.

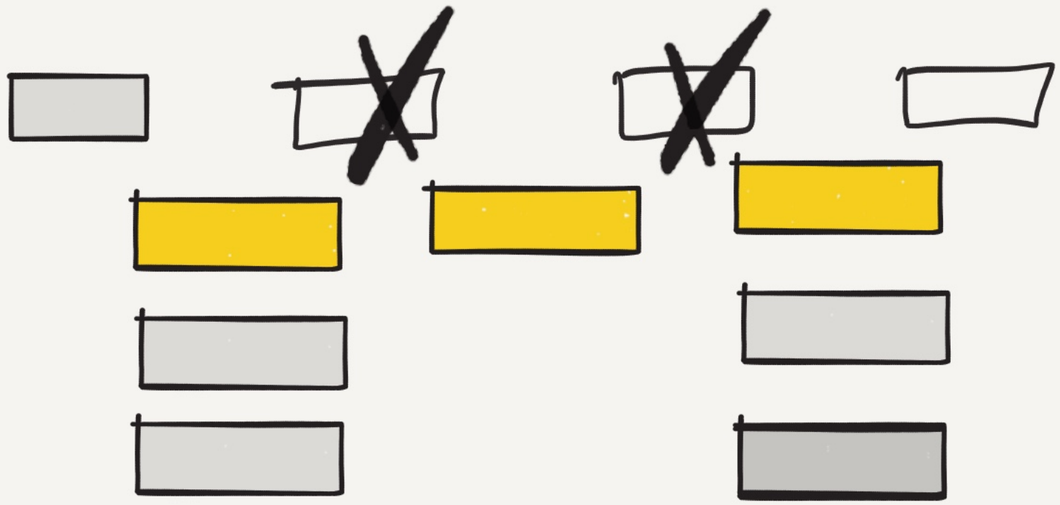
× (3)



} total # assigned = 1.

Pick the one with least time.

× (2) Pick the one with least # of conflicts, solves the previous example.



total # assigned = 3.

Best is 4.

Earliest finish time works!!

Earliest Finish Time Algorithm:

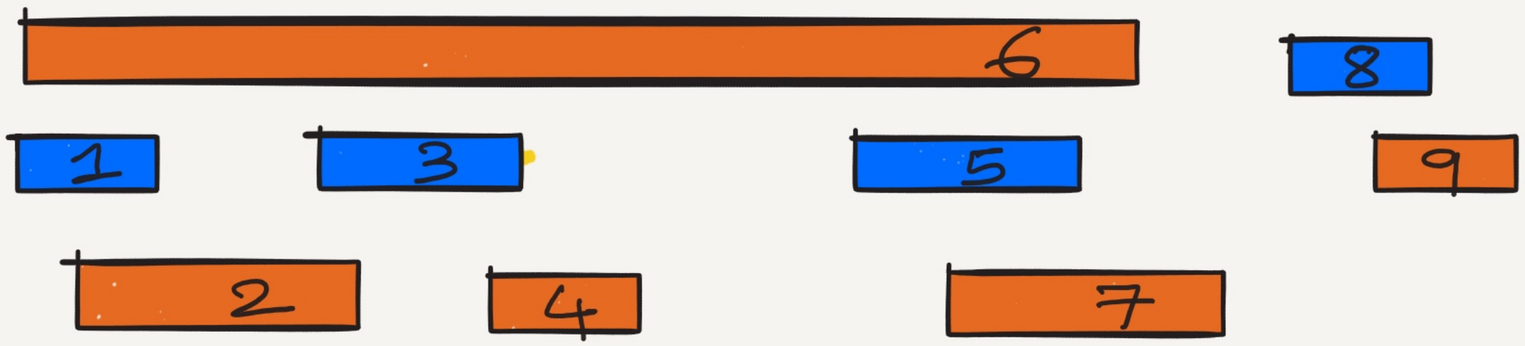
(1) R = all jobs. $A = \phi$.

(2) While R is not empty

→ Pick request i with the least finish time from R.

→ Add i to A.

→ Remove all requests that conflict with i.



Output = jobs numbered 1, 3, 5, 8 in the figure.

How to analyze??

"Greedy stays ahead"

"At any point in time, our algorithm's performance is no worse than that of the optimal one"

Need a way to quantitatively compare our algorithm to the optimal set of jobs.

"Lemma:" Our algorithm frees up sooner than the optimal "one".

Suppose that $A = \{ i_1, i_2, \ldots, i_k \}$

$f(i_1) < f(i_2) < \ldots < f(i_k)$.

Suppose an optimal solution is

$O = \{ j_1, j_2, \ldots, j_m \}$

$f(j_1) < f(j_2) < \ldots < f(j_m)$.

the jobs in O are ordered in increasing finish time.

Lemma: $\forall \quad l \leq k, \quad f(i_l) \leq f(j_l)$.

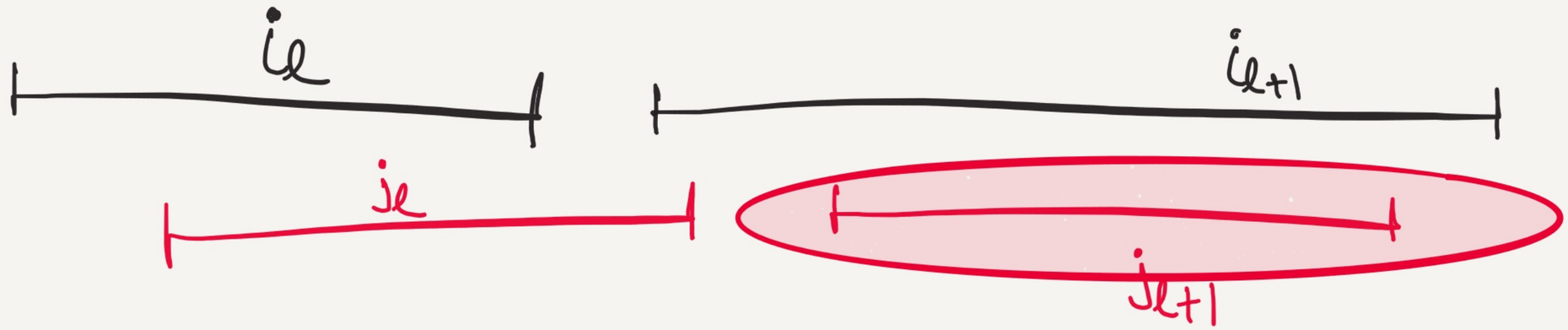
$f(i_l)$: finish time of the l^{th} job under A.

$f(j_l)$: finish time of the l^{th} job under O.

Proof: Induction.

Base case: $l = 1$. True because $f(i_1)$ was the least finish time.

Induction step: Suppose claim is true for l. Want to show for $l+1$.



① When we picked $i_{\ell+1}$, $j_{\ell+1}$ also belongs to the set R.

Why, $f(i_\ell) \leq f(j_\ell) \leq s(j_{\ell+1})$

$\Rightarrow$ $j_{\ell+1}$ does not conflict with i_ℓ.

$\Rightarrow$ that the situation in figure cannot occur.

$\Rightarrow$ Lemma is true by induction.

Theorem: Earliest Finish First$^{(EFF)}$ finds an optimal set of jobs.

Proof: Suppose there exists $O = \{j_1, j_2, \ldots j_m\}$ such that $m > k$.

$(A = \{i_1, i_2, i_3 \ldots i_k\})$

(Here as before $f(i_1) < f(i_2) < \ldots < f(i_k)$

$f(j_1) < f(j_2) < \ldots < f(j_k) < \ldots < f(j_m)$

By the lemma, $f(i_k) \leq f(j_k)$.

$\Rightarrow$ j_{k+1} does not conflict with the jobs in A.

$\Rightarrow$ j_{k+1} is not removed $\Rightarrow$ EFF algorithm would consider it & assign at least one more

Theorem: Earliest Finish First$^{(EFF)}$ finds an optimal set of jobs.

Proof: Suppose there exists $O = \{j_1, j_2, \ldots j_m\}$ such that $m > k$.

$(A = \{i_1, i_2, i_3 \ldots i_k\})$

(Here as before $f(i_1) < f(i_2) < \ldots < f(i_k)$

$f(j_1) < f(j_2) < \ldots < f(j_k) < \ldots < f(j_m)$

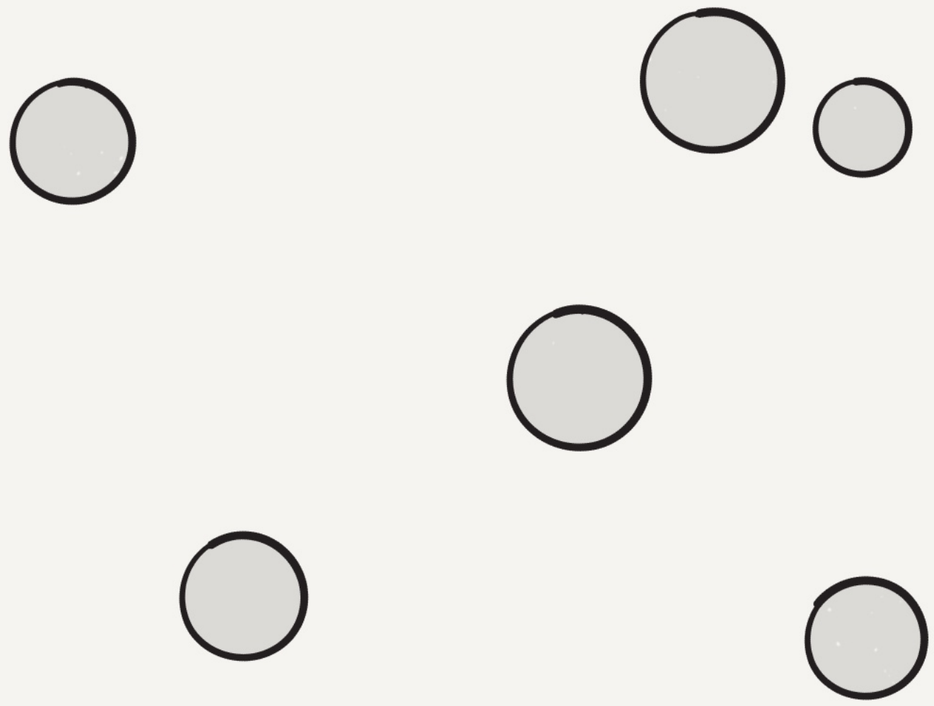
By the lemma, $f(i_k) \leq f(j_k)$.

$\Rightarrow$ j_{k+1} does not conflict with the jobs in A.

$\Rightarrow$ j_{k+1} is not removed $\Rightarrow$ the set R is not empty.

Minimum Spanning Trees

(3rd example of Greedy algorithms).



fiber optic cable.
Connections

Each pair (i,j) has a cost C(i,j).

Goal: Connect the cities with least total cost.

$G = (V, E)$ Each edge has an associated weight.

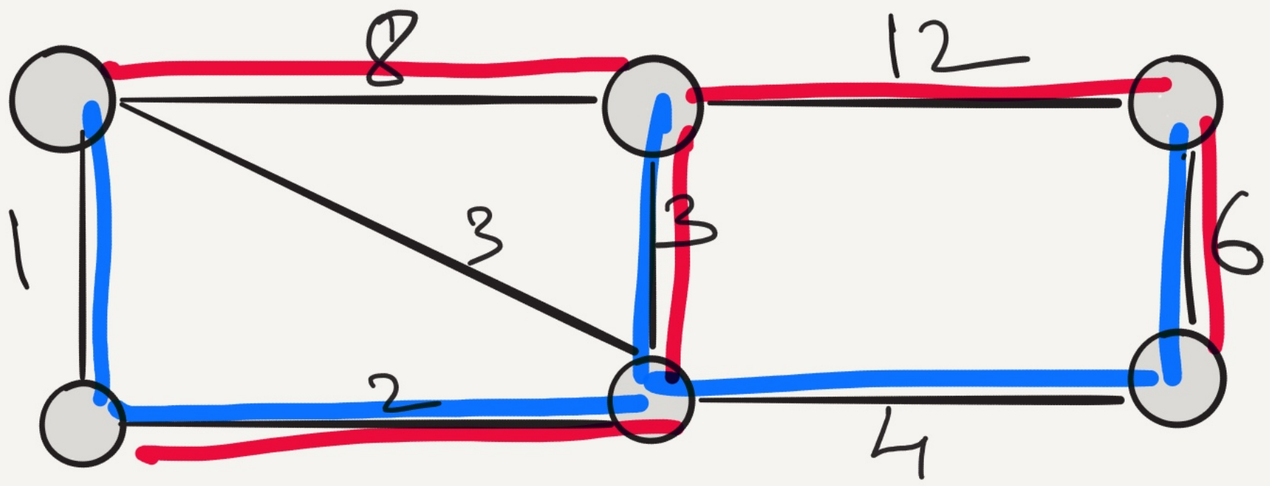
↓ Vertices ↓ Edges

Undirected weighted graphs

Spanning Tree: A spanning tree is a sub-graph of G so that

(a) the edges form a tree

(b) they covers/hit all the vertices of V.



wt(blue) = 16

wt(red) = 31

Minimum Spanning Tree (MST):

→ Spanning tree with least total weight.

INPUT: $G = (V, E)$ undirected weighted graph

OUTPUT: Compute a minimum spanning tree (MST).

Applications: → fiber optic cables / electricity grids / hydraulic
→ Ethernet protocols
→ Important primitive for approximation algorithms

Greedy algorithms for computing a MST.

① Find the "closest" city/vertex that is not "covered" and "add" it. → PRIM's Algorithm

② Find the edge with least weight that does not create a cycle and add that edge.

→ Kruskal's algorithm.