ASD (2° Mod)
Lesson n. 2 on MST

Time complexity of Prim's Algorithm

THM: $O(n^2)$ with an array; $O(m \log n)$ with a binary heap.

Proof: DO AS EXERCISE!

SUGGESTIONS: give answers to :

- How many times a node is explored ?
- How do you represent Q?
- Which operations on Q for every new explored node? How many? How can you implement them?

mplementation. Use a priority queue ala Dijkstra.

Maintain set of **explored** nodes **S**.

For each **unexplored** node **v**, maintain *attachment cost*

a[v] = cost of cheapest edge v to a node in S

```
Prim(G, c) {
    foreach (v ∈ V) a[v] ← ∞
    Initialize an empty priority queue Q
    foreach (v ∈ V) insert v onto Q
    Initialize set of explored nodes S ← ϕ

    while (Q is not empty) {
        u ← delete min element from Q
        S ← S ∪ { u }
        foreach (edge e = (u, v) incident to u)
            if ((v ∉ S) and (c_e < a[v]))
                decrease priority a[v] to c_e
}
```

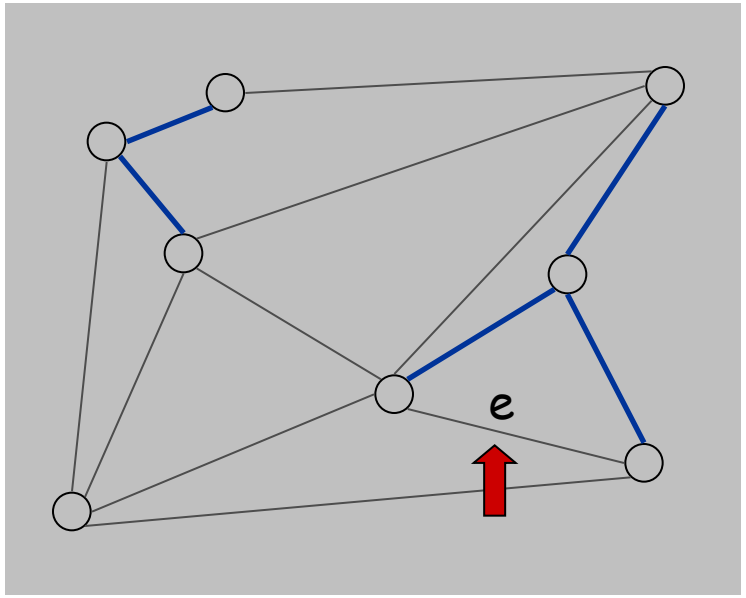
or any visited node **u ∈ V**, update $O(\deg(u))$ keys in **Q** → $\sum_u \deg(u) = O(m)$

ach update costs $O(\log n)$ (using Heap) → Total: $O(m \log n)$

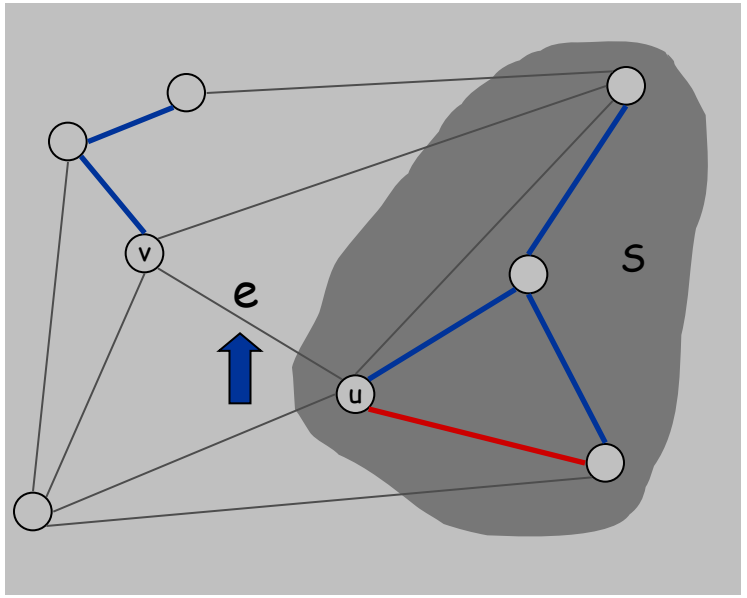
Kruskal's Algorithm: Proof of Correctness

Kruskal's algorithm. [Kruskal, 1956]

- Consider edges in ascending order of weight.
- Case 1: If adding **e** to **T** creates a cycle, discard **e** according to cycle property.
- Case 2: Otherwise, insert **e** = **(u, v)** into **T** according to cut property where **S** = set of nodes in **u's connected component**.



Case 1



Case 2

Implementation: Kruskal's Algorithm

Implementation. Use the union-find data structure.

- Build set T of edges in the MST.
- Maintain set for each connected component.
- $O(m \log n)$ for sorting and $O(m \alpha(m, n))$ for union-find.

$m \le n^2 \Rightarrow \log m$ is $O(\log n)$

essentially a constant

```
Kruskal(G, c) {
   Sort edges weights so that c1 ≤ c2 ≤ ... ≤ cm.
   T ← φ

   foreach (u ∈ V) make a set containing singleton u

   for i = 1 to m                    are u and v in different connected components?
      (u,v) = ei                     ↙
      if (u and v are in different sets) {
         T ← T ∪ {ei}
         merge the sets containing u and v
      }                              ↖ merge two components
   return T
}
```

NEW DATA STRUCTURE!

WE NOW NEED A NEW DATA STRUCTURE TO MANAGE
SUBSET OPERATIONS EFFICIENTLY!

THIS PART FOLLOWS THE BOOK and IT WILL BE GIVEN IN FEW LESSONS:

Demetrescu et Al,
Algoritmi e Strutture Dati

ARBITRARY EDGE COSTS

To remove the assumption that all edge costs are distinct: perturb all edge costs by tiny amounts to break any ties.

Impact. Kruskal and Prim only interact with costs via pairwise comparisons. If perturbations are sufficiently small, MST with perturbed costs is MST with original costs.

e.g., if all edge costs are integers, perturbing cost of edge e_i by i / n^2

Implementation. Can handle arbitrarily small perturbations implicitly by breaking ties lexicographically, according to index.

Small perturbations of edge costs have no impacts!

Assume $c(e) \geq 1$ for all $e \in E$ and fix $b = 1/n^2$. Now, consider the new instance $I_b = \langle G(V,E),$ $c_b: E \rightarrow R+ \rangle$ such that:

$c_b(e) = c(e) \pm b$ and $c(e) \neq c(e')$ for any $e \neq e'$ (all distinct!)

THM. Let T_b be any MST for I_b then T_b is also a MST for the original instance

$I = \langle G(V,E),\ c: E \rightarrow R+ \rangle$

Proof. By contradiction. Absurd hyp: exists T^* better than T_b for I,

We use the following facts:

I) $C(T^*, I_b) < C(T^*, I) + 1/n$ (since $b=1/n^2$)

II) $C(T^*, I) \leq C(T_b, I) - 1$ (absurd hyp.)

III) $C(T_b, I) < C(T_b, I_b) + 1/n$ (since $b=1/n^2$)

(I) ← (II) → IV) $C(T^*, I_b) < C(T_b, I) - 1 + 1/n$

(IV)←(III) → $C(T^*, I_b) < C(T_b, I_b) + 1/n - 1 + 1/n$ (for $n > 2$)

$< C(T_b, I_b)$ (absurd!)

⊙

EXCERCISE N.1 (MST Properties)

Input: Connected Graph $G(V,E)$; $e \in E$.

Output: Decide whether an MST T exists s.t. $e \in T$

Provide an algorithm working in $O(m+n)$ time.

Hint: **Combine the CUT Property and the CYCLE one to decide whether** e is a MINIMAL BRIDGE.

How detect which is the case for input $G(V,E);c:E\rightarrow R^{+}$; **e=(u,w)** in E ?

Simple Idea:

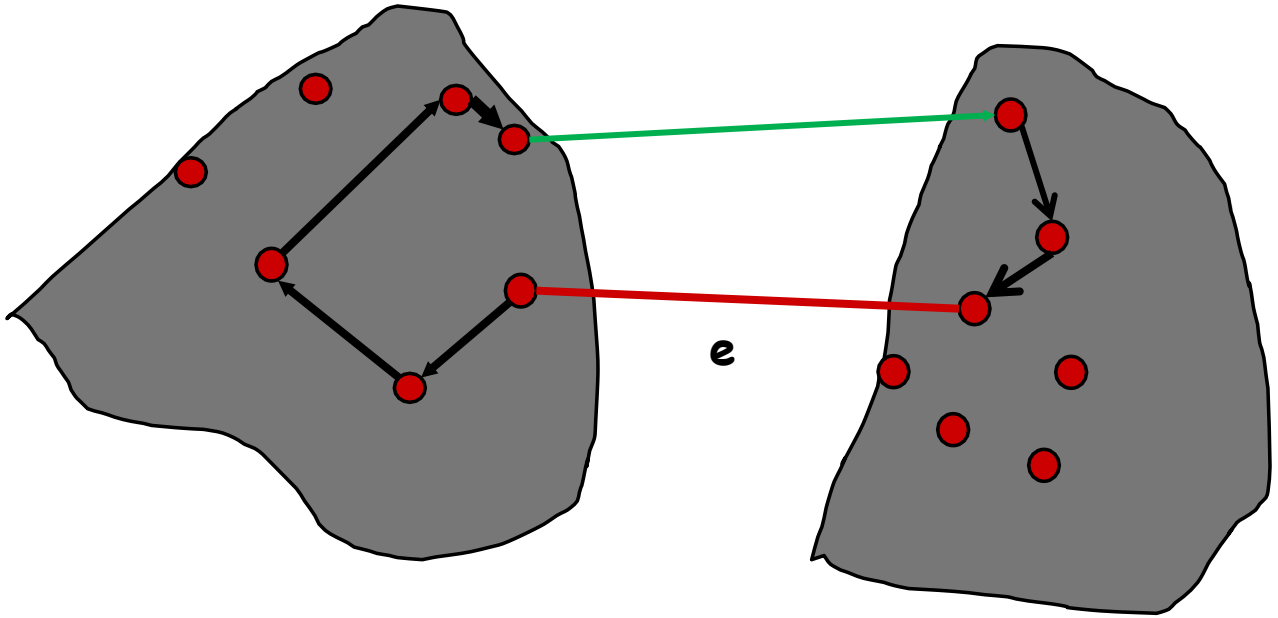
1) **Remove** all edges from E which are more expensive than **e** and remove also **e**. **Set** ***G'(V,E')*** as this new graph.

2) **Start** a *BFS* (or a *DFS*) from u (or from w) over ***G'(V,E')***

3) **If** the computed Tree **T(u)** contains **w** then **return**: e does not belong to any MST, otherwise **return:** it does!

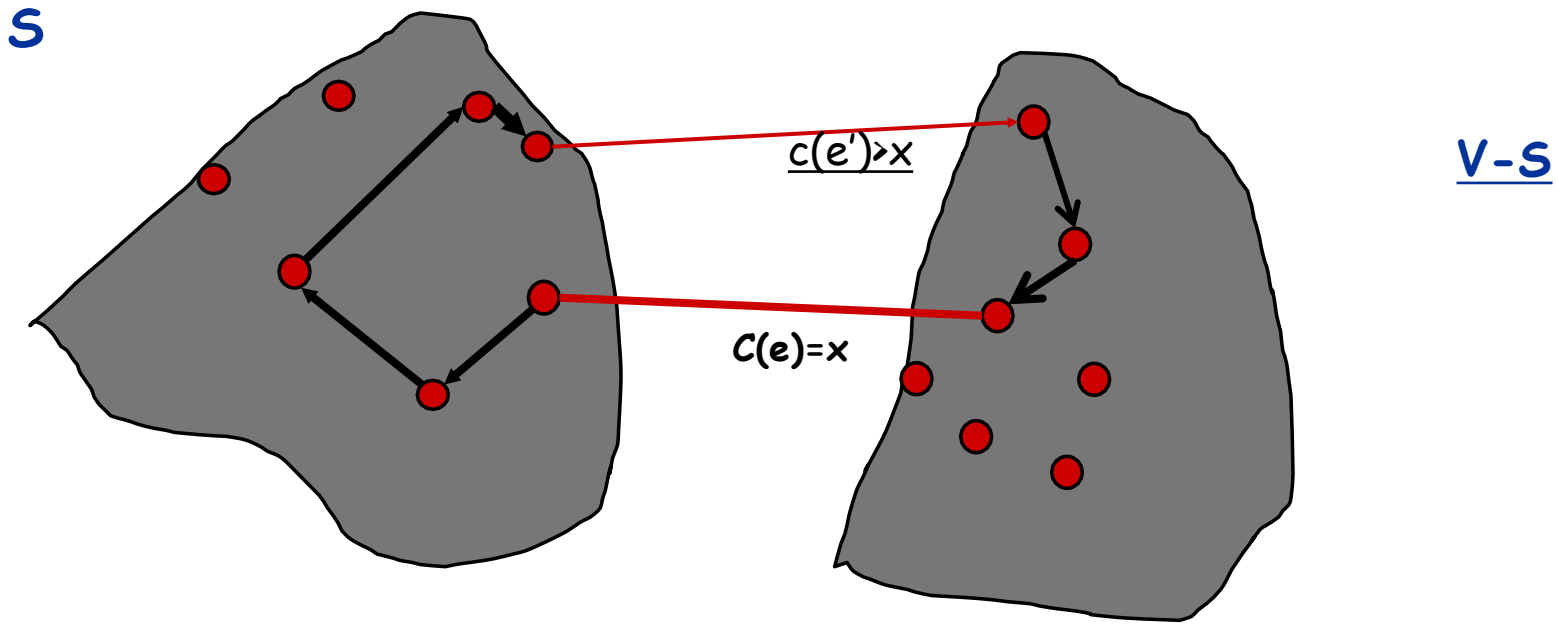
Let **e** := **(u,w)** and **c(e)** := **c**

I) case: there is a Path (forming a cycle with **e**) where **e** is the most expensive!



Then, by the cycle property → **e** cannot belong to any MST

II case: the Set **S** of all nodes reachable from **u** with edges cheaper than **x** does not contain **w** and, so, the set **V-S** contains **w** and all nodes reachable from **w** with edges cheaper than **x**



Then, by the cut property → **e** belongs to any MST

12

Excersise n.2

Prove or Confute the following Statements:

a) Let $\langle G(V,E)\ w\rangle$ be s.t. G is connected and all edges have distinct weights. Let e^* be the edge of minimal weight. Does e^* always belong to an MST ?

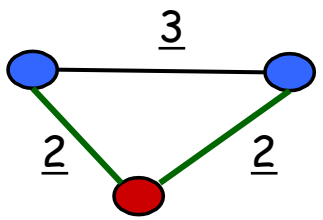
b) Let T be an MST for $\langle G(V,E),w\rangle$ and consider the NEW instance $\langle G(V,E),w^2\rangle$ where

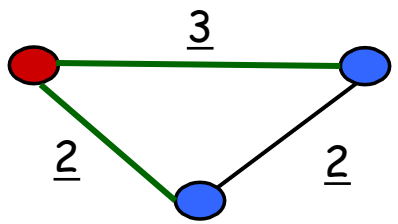
$$\text{for any } e \in E : w^2(e) = (w(e))^2$$

Is T an MST for $\langle G(V,E),w^2\rangle$ as well?

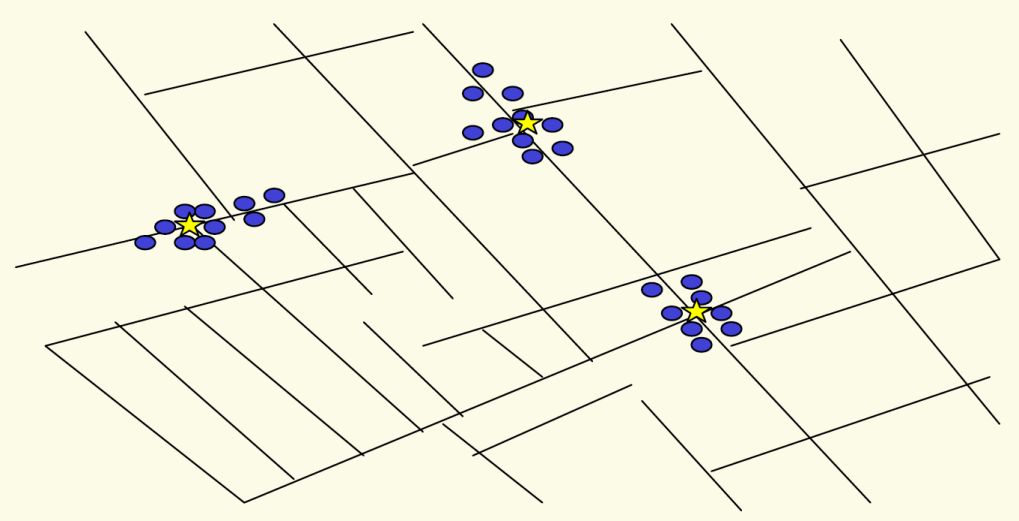
Shortest Path Tree vs Minimum Spanning Tree

- Consider an **instance** of the **MST** problem: **<G(V,E);c: E → R+>**
- **FACT 1:** The MST ***T(s)*** computed by Prim's Algorithm may depend by the source node **s**. But **T(s)** is a feasible and optimal solution for instance <G(V,E);c: E → R+>, **for any choice of s' in V.**
- Consider an **instance** of the **SPT** problem: **<G(V,E);c: E → R+;s>**
- **FACT 2: The SPT T(s) (and its cost) computed by *any* correct algorithm may depend on the choice of s**





4.7 Clustering



Outbreak of cholera deaths in London in 1850s.
Reference: Nina Mishra, HP Labs

Clustering

Clustering. Given a set U of n objects labeled $p_1, \ldots, p_n$, classify into coherent groups.

↑ photos, documents. micro-organisms

Distance function. Numeric value specifying "closeness" of two objects:

distance(p_i, pj)

↑ number of corresponding pixels whose intensities differ by some threshold

Fundamental problem. Divide into clusters so that points in different clusters are far apart.

- Routing in mobile ad hoc networks.
- Identify patterns in gene expression.
- Document categorization for web search.
- Similarity searching in medical image databases
- Skycat: cluster 10^9 sky objects into stars, quasars, galaxies.

Clustering of Maximum Spacing

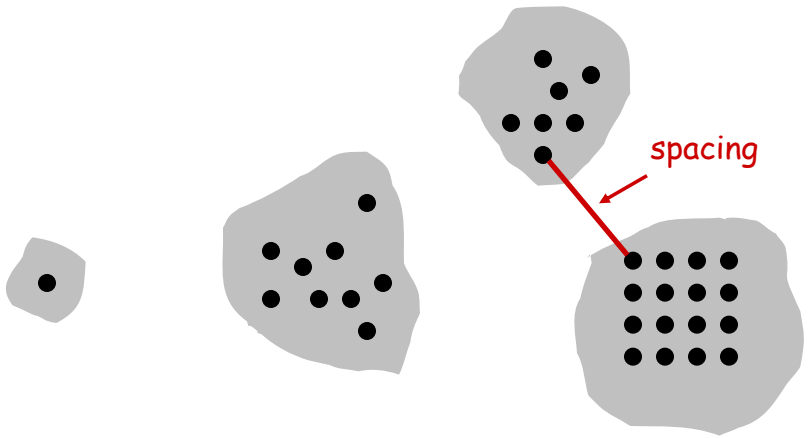
k-Clustering. Divide objects into k non-empty groups.

Distance function. Assume it satisfies several natural properties.

- $d(p_i, p_j) = 0$ iff $p_i = p_j$ (identity of indiscernibles)
- $d(p_i, p_j) \geq 0$ (nonnegativity)
- $d(p_i, p_j) = d(p_j, p_i)$ (symmetry)

Spacing. Min distance between any pair of points in different clusters.

Clustering of maximum spacing. Given an integer k, find a k-clustering of maximum spacing.



k = 4

K-Clustering

EXCERCISE: Provide formal definition of the K-Clustering Problem:

- INSTANCE: I = <….> ??????
- FEASIBLE SOLUTIONS: Y = ….. ????
- COST OF A FEASIBLE SOLUTION: c(Y)
- GOAL: MINIMIZE/MAXIMIZE ???

- Determine the main parameters of the size of the Instance

PROBLEM MODEL

MAIN IDEA:

USE WEIGHTED GRAPHS TO REPRESENT THE INSTANCE !

$G(V,E)$, $d: E \rightarrow R^+$, where

- $V = U = \{ p_1, \ldots, p_n \}$
- $E = \{$ALL non-ordered pairs in V$\}$ (i.e. complete graph)
- $Cost(e) = d(p_i, p_j)$ for any $e = \{p_i, p_j\}$

Greedy Clustering Algorithm

Single-link k-Clustering algorithm.

- Form a graph on the vertex set **U**, corresponding to **n** clusters.
- Find the **closest** pair (edge) of objects (p,p′) such that p & p′ are not in the same cluster, and add an edge between them: so merging 2 clusters.
- Repeat **n-k** times until there are exactly **k** clusters.

Key Obs. 1. This procedure is precisely Kruskal's algorithm (except we stop when there are **k** connected components).

Key Obs. 2. Equivalent to finding an MST **T** and deleting the **k-1** most expensive edges from **T** (thus forming **k** connected components).

Proofs of 1 and 2: Excercises

Greedy Clustering Algorithm: Analysis

Theorem. Let C^* denote the clustering $C^*_1, \ldots, C^*_k$ formed by deleting the **k-1** most expensive edges of an MST. Then, C^* is a k-clustering of **maximal spacing**.

Pf. Let C denote some other clustering $C_1, \ldots, C_k$.

- The spacing of C^* is the length **d*** of the **(k-1)st most expensive edge**.
- Let p, p′ be in the same cluster in C^*, say C^*_r, but different clusters in C, say C_s and C_t.
- Some edge (q, q′) on p-->p′ path in C^*_r spans **two diff.** clusters in C.
- **All** edges on p-->p′ path have length ≤ **d*** since Kruskal chooses them.
- Spacing of C is ≤ **d*** since q and q′ are in **different** clusters of C. ▪