kostra grafu (Spanning Tree)
neorientovaný graf!

$G = (V, E)$ undirected connected graph and W weight function.

$H = (V_H, E_H)$ with $V_H \subseteq V$ and $E_H \subseteq E$ subgraph of G.

- The **weight** of H is the number

$W(H) = X$
e2EH
$W(e)$.

- H is a **spanning subgraph** of G if $V_H = V$.

Observation 10.2
A connected spanning subgraph of minimum weight is a tree.

Minimum Spanning Trees

(G,W) undirected connected weighted graph

Definition 10.3
A **minimum spanning tree (MST)** of G is a connected spanning
subgraph T of G of minimum weight.

The **minimum spanning tree problem**:
Input: Undirected connected weighted graph (G,W)
Output: An MST of G

Prim's Algorithm

Idea
Grow an MST out of a single vertex by always adding edges of
minimum weight.

A **fringe edge** for a subtree T of a graph is an edge with exactly one
endpoint in T (so $e = (u, v)$ with $u \in T$ and v 6$\in$ T).

Algorithm PRIM(G,W)

1. $T \leftarrow$ one vertex tree with arbitrary vertex of G
2. **while** there is a fringe edge **do**
3. add fringe edge of minimum weight to T
4. **return** T

Implementation of Prim's Algorithm

Algorithm PRIM(G,W)

1. Initialise parent array _:
_[v] $\leftarrow$ NIL for all vertices v
2. Initialise weight array:
weight[v] $\leftarrow \infty$ for all vertices v
3. Initialise priority queue Q
4. $v \leftarrow$ arbitrary vertex of G
5. Q.INSERT$(v, 0)$
6. weight[v] = 0
7. **while not**(Q.IS-EMPTY()) **do**
8. $y \leftarrow$ Q.EXTRACT-MIN()
9. **for all** z adjacent to y **do**
10. RELAX(y, z)
11. **return** _

Algorithm RELAX(y, z)

1. $w \leftarrow$ W`(y, z)´
2. **if** weight[z] = ∞ **then**
3. weight[z] $\leftarrow w$
4. _[z] $\leftarrow y$
5. Q.INSERT(z,w)
6. **else if** $w <$ weight[z] **then**
7. weight[z] $\leftarrow w$
8. _[z] $\leftarrow y$
9. Q.DECREASE KEY(z,w)

Kruskal’s Algorithm

A different approach to computing MSTs.
A **forest** is a graph whose connected components are trees.

Idea

Starting from the spanning forest without any edges, repeatedly
add edges of minimum weight until the forest becomes a tree.

Algorithm KRUSKAL(G,W)

1. $F \leftarrow \emptyset$
2. **for all** $e \in E$ in the order of increasing weight **do**
3. **if** the endpoints of e belong to different connected components of (V, F) **then**
4. $F \leftarrow F \cup \{e\}$
5. **return** tree with edge set F

Implementation of Kruskal’s Algorithm

Algorithm KRUSKAL(G,W)

1. $F \leftarrow 0$
2. **for all** vertices v of G **do**
3. MAKE-SET(v)
4. sort edges of G into non-decreasing order by weight
5. **for all** edges (u, v) of G in non-decreasing order by weight **do**
6. **if** FIND-SET(u) /= FIND-SET(v) **then**
7. $F \leftarrow F \cup \{(u, v)\}$
8. UNION(u, v)
9. **return** F

Data Structures for Disjoint Sets

- A **disjoint set** data structure maintains a collection
$S = \{S_1, \ldots, S_k\}$ of **disjoint sets**.
- The sets are **dynamic**, i.e., they may change over time.
- Each set S_i is identified by some **representative**, which is some
member of that set.

Operations:

- MAKE-SET(x): Creates new set whose only member is x. The
representative is x.
- UNION(x, y): Unites set S_x containing x and set S_y containing y
into a new set S and removes S_x and S_y from the collection.
- FIND-SET(x): Returns representative of the set holding x.