algorithms and data structures (spanning trees)

amir yehudayoff

spanning trees

a connected structure (graph)

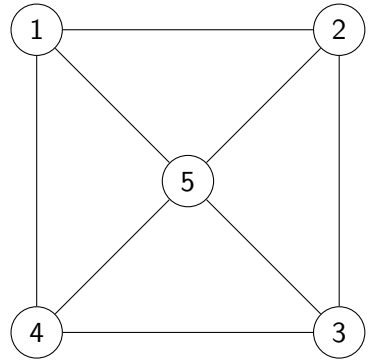
what is the minimum cost of keeping it connected?

graphs

(simple undirected) graph $G = (V, E)$

vertex set V

edge set E is pairs $\{u, v\}$ for $u \neq v$ in V



graphs

graph $G = (V, E)$

connected if there is a path between any two vertices
(better to have more edges)

no cycles if there are no non-trivial cycles
(better to have less edges)

definition a forest F is a graph with no cycles

forests

definition a forest F is a graph with no cycles

properties if F be a forest then

$|E(F)| \leq |V(F)| - 1$

if $|E(F)| < |V(F)| - 1$ then there are $e = \{u, v\}$ so that $F \cup \{e\}$ is a forest

if $|E(F)| = |V(F)| - 1$ then F is connected

definition a tree T is a connected forest
(between "connected" and "no cycles")
(analog of basis in linear algebra)

definition a tree T is a connected forest
(between "connected" and "no cycles")
(analog of basis in linear algebra)

properties
if T is a tree then $|E(T)| = |V(T)| - 1$
every tree has a leaf
every tree is planar

terminology

connected undirected graph G with weight $w : E \to \mathbb{R}$

H is a subgraph of G if $V(H) = V(G)$ and $E(H) \subseteq E(G)$

spanning tree T is a connected subgraph of G with no cycles

the weight of T is $w(T) = \sum_{e \in E(T)} w(e)$

MST

input: connected G and $w : E(G) \to \mathbb{R}$

output: minimum weight spanning tree

remark discrete optimization search problem

Kruskal: greedy

let $F_0 = \emptyset$

as long as F_k is not spanning

  let $e \in E$ be of minimum weight
    so that $F_k \cup \{e\}$ has no cycles
  set $F_{k+1} = F_k \cup \{e\}$

output the spanning F

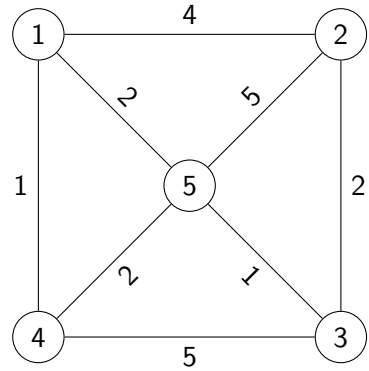
run

let $F_0 = \emptyset$

as long as F_k is not spanning

let $e \in E$ be of minimum weight
so that $F_k \cup \{e\}$ has no cycles
set $F_{k+1} = F_k \cup \{e\}$

output the spanning F



running time

as long as F_k is not spanning
  let $e \in E$ be of minimum weight ... $F_k \cup \{e\}$ has no cycles
  set $F_{k+1} = F_k \cup \{e\}$

how? later on...

claim

each F_k is a forest with k edges so $k \leq n-1$

if $k < n-1$ then there is $e \in E(G)$ so that $F_k \cup \{e\}$ has no cycles

F_{n-1} is a tree

F_{n-1} is MST

claim

each F_k is a forest with k edges so $k \leq n-1$

if $k < n-1$ then there is $e \in E(G)$ so that $F_k \cup \{e\}$ has no cycles

F_{n-1} is a tree

F_{n-1} is MST

how can we prove MST?
(it is in not unique)

the algorithm generates

$$F_0 \subset F_1 \subset F_2 \subset \ldots \subset F_{n-1}$$

the algorithm generates

$$F_0 \subset F_1 \subset F_2 \subset \ldots \subset F_{n-1}$$

claim

for every k there is MST T_k so that $F_k \subseteq T_k$

"witness MST is dynamic"

there is MST T_k so that $F_k \subseteq T_k$

proof (induction)

there is MST T_k so that $F_k \subseteq T_k$

proof (induction)

(base) $k = 0$ is true (G is connected)

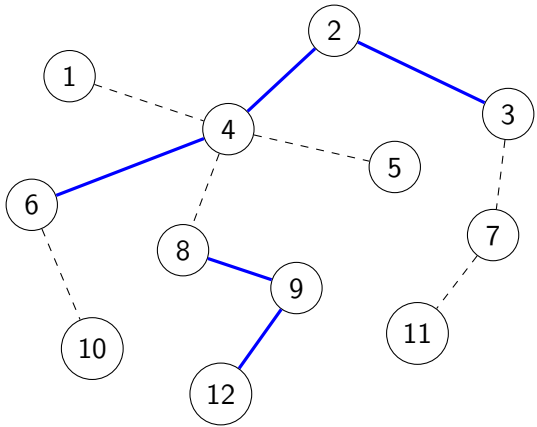
there is MST T_k so that $F_k \subseteq T_k$

proof (induction)

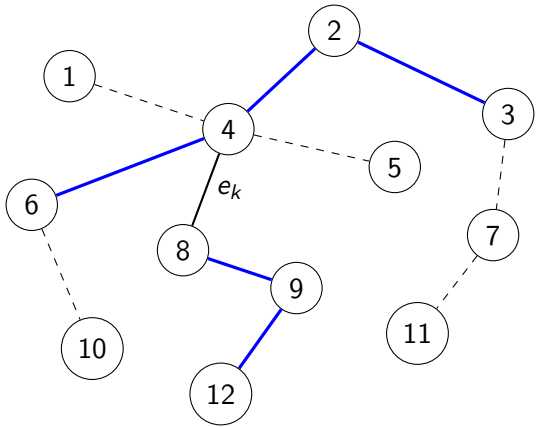
(base) $k = 0$ is true (G is connected)

(step) assuming $F_k \subseteq T_k$ what is T_{k+1}?

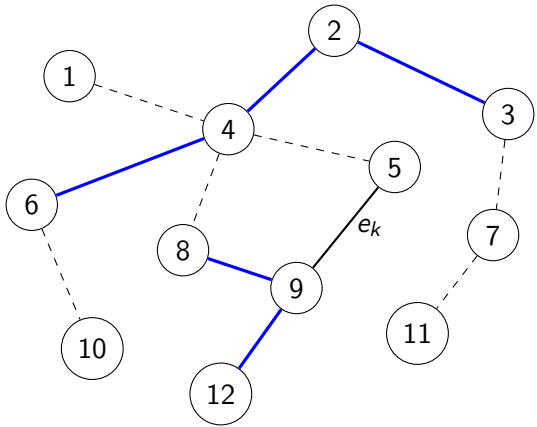
$F_k \subset T_k$



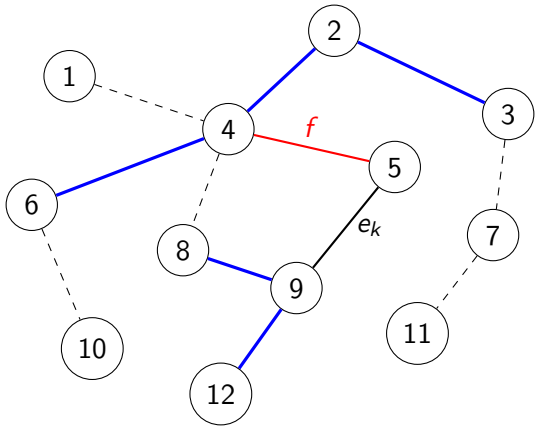
easy e_k



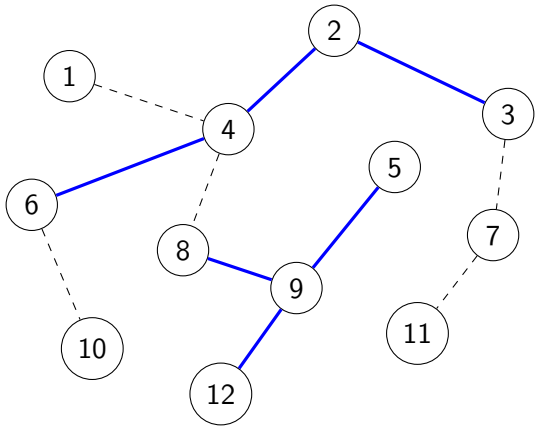
harder e_k



find f



$F_{k+1} \subset T_{k+1}$



assuming $F_k \subseteq T_k$ what is T_{k+1}?

let e_k be

$$F_{k+1} = F_k \cup \{e_k\}$$

if $e_k \in T_k$ then $T_{k+1} = T_k$

otherwise...

assuming $F_k \subseteq T_k$ what is T_{k+1}?

e_k is $F_{k+1} = F_k \cup \{e_k\}$ and assume $e_k \notin T_k$

assuming $F_k \subseteq T_k$ what is T_{k+1}?

e_k is $F_{k+1} = F_k \cup \{e_k\}$ and assume $e_k \notin T_k$

by algorithm for every $e \in T_k \setminus F_k$

$$w(e_k) \leq w(e)$$

assuming $F_k \subseteq T_k$ what is T_{k+1}?

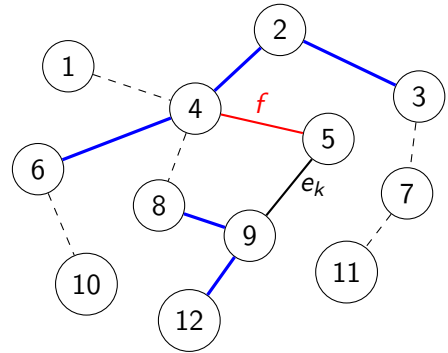
e_k is $F_{k+1} = F_k \cup \{e_k\}$ and assume $e_k \notin T_k$

by algorithm for every $e \in T_k \setminus F_k$

$$w(e_k) \leq w(e)$$

in $T_k \cup \{e_k\}$ there is one cycle C

because no cycle in F_{k+1} on C there is $f \notin F_{k+1}$



assuming $F_k \subseteq T_k$ what is T_{k+1}?

e_k is $F_{k+1} = F_k \cup \{e_k\}$ and assume $e_k \notin T_k$

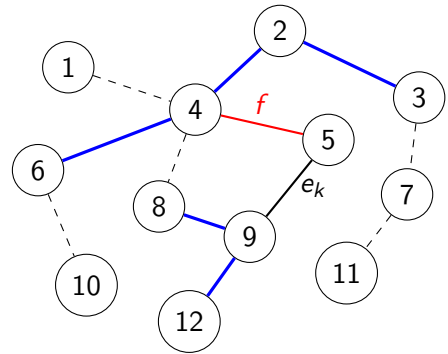
by algorithm for every $e \in T_k \setminus F_k$

$$w(e_k) \leq w(e)$$

in $T_k \cup \{e_k\}$ there is one cycle C

because no cycle in F_{k+1} on C there is $f \notin F_{k+1}$

set $T_{k+1} = (T_k \setminus f) \cup \{e_k\}$



T_{k+1}'s properties

e_k is $F_{k+1} = F_k \cup \{e_k\}$ and assume $e_k \notin T_k$

found f in $F_{k+1} \setminus T_k$ in a unique cycle

set $T_{k+1} = (T_k \setminus f) \cup \{e_k\}$

T_{k+1} is a tree because $T_k \cup \{e_k\}$ has unique cycle and removed f

$F_{k+1} = F_k \cup \{e_k\} \subseteq (T_k \setminus f) \cup \{e_k\} = T_{k+1}$ because $f \notin F_{k+1}$

$w(T_{k+1}) = w(T_k) - w(f) + w(e_k) \leq w(T_k)$

claim

for every k there is MST T_k so that $F_k \subseteq T_k$

for $k = n - 1$ we conclude that output is MST

remarks

could be many MSTs

algorithm founds one of them

if weights are distinct, there is a unique MST (exercise)

other algorithms

there are many MST algorithms (Prim, Botuvka, ...)

all of them are “greedy”

Prim e.g. grows a tree instead of a forest

a "promise" problem: assumed G is connected

for general graphs

Kruskal's algorithm outputs a "minimum weight maximally connected forest"

Prim's algorithm gets stuck in a "local tree"

running time

as long as F_k is not spanning
  let $e \in E$ be of minimum weight ... $F_k \cup \{e\}$ has no cycles
  set $F_{k+1} = F_k \cup \{e\}$

order E according to w in time $O(|E| \log |E|)$
go over ordered list once

how to check cycles? (data structure...)

closing a cycle?

need to quickly decide if e closes a cycle in the forest F

the "disjoint set" data structure

partitions

a partition of V is a collection of pairwise disjoint sets $S_1, S_2, \ldots, S_m$ such that

$$V = \bigcup_{i=1}^{m} S_i$$

(e.g. connected component in F)

disjoint sets

maintain a partition with operations

are u, v in the same part?

merge parts of v, u

maintain a partition with operations

are u, v in the same part?

merge parts of v, u

remarks

can check if $e = \{u, v\}$ close a cycle by checking if parts of u, v are the same

partition of $F \cup \{e\}$ obtained by merging

what is part of v?

we shall have a representative map

$$r : V \to V$$

such that for every v

$$v, r(v) \text{ belong to same part}$$

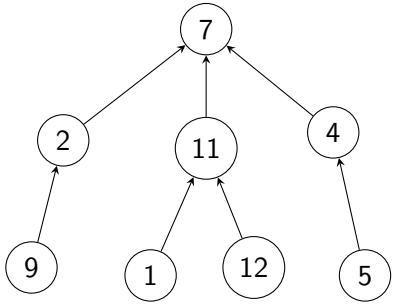
so: u, v are in same part iff $r(u) = r(v)$

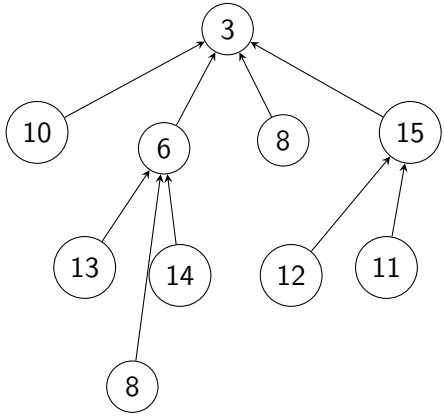
pointers

maintain pointers $p : V \to V$

invariant: p has no cycles

example





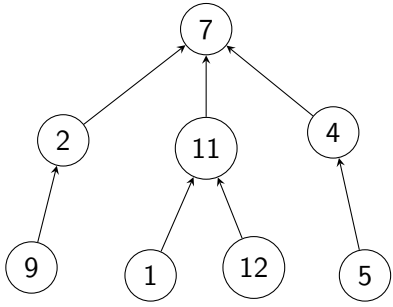
representatives

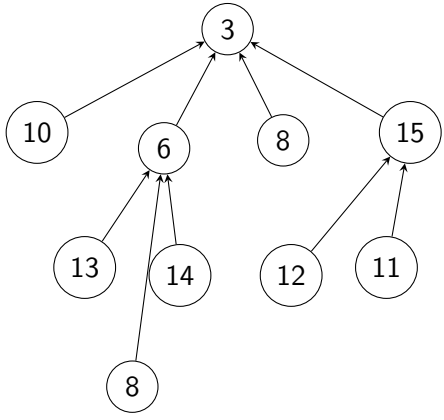
$r(v)$ is "follow p to a root"

if $p(v) = v$ output v
else output $r(p(v))$

the representatives are roots in the directed forest

example





cost of $r(\cdot)$ is depth of p-tree

if depth is large then $r(\cdot)$ is expensive

small depth

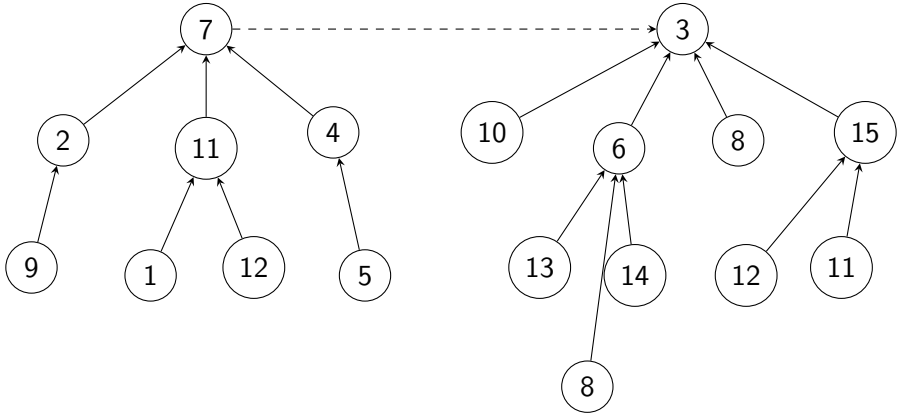
at each root v write the size $s(v)$ of the part of v

union two roots $u \neq v$

if $s(u) > s(v)$ set $p(v) = u$ and $s(u) = s(u) + s(v)$

else set $p(u) = v$ and $s(v) = s(u) + s(v)$

union(7,3)



cost of $r(\cdot)$ is depth of p-tree

cost of $r(\cdot)$ is depth of p-tree

claim
at all times and for all v, depth of tree with root v is $\leq \log_2 s(v)$

cost of $r(\cdot)$ is depth of p-tree

claim
at all times and for all v, depth of tree with root v is $\leq \log_2 s(v)$

proof (induction)
(base) initially true
(step) when merge $u \neq v$ with $s(u) \geq s(v)$ new depth is

$$\begin{aligned}
&\leq \max\{\log_2(s(u)), 1 + \log_2(s(v))\} \\
&\leq \max\{\log_2(s(u)), 1 + \log_2((s(v) + s(u))/2)\} \\
&\leq \log_2(s(u) + s(v))
\end{aligned}$$

maintain a partition with operations r(v), merge(u,v)

$r(v)$ takes time $O(\log |V|)$

merge takes time $O(1)$

maintain a partition with operations r(v), merge(u,v)

$r(v)$ takes time $O(\log |V|)$

merge takes time $O(1)$

Kruskal's running time is $O(|E| \log(|E|) + |V| \log(|V|))$

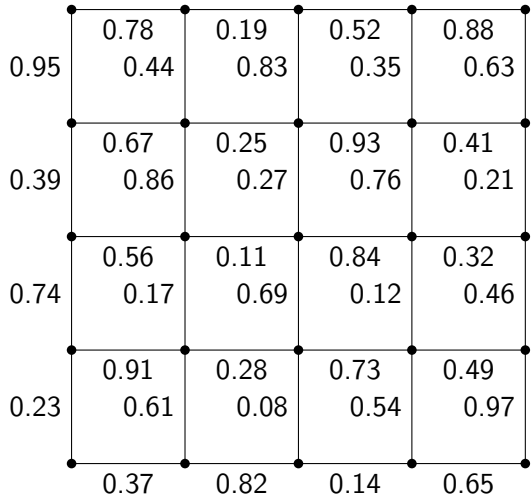
better

there are better data structure supporting *rep* and *merge*

for example there are data structures with running time $O(n \log^* n)$

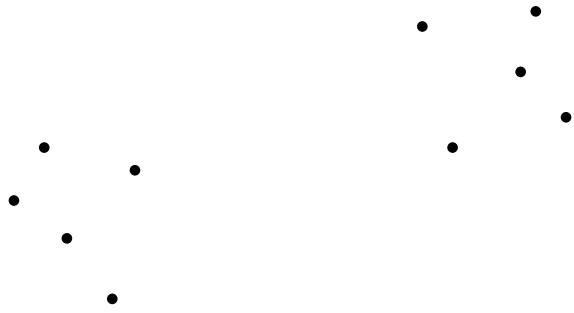
but these are more complicated

application: resistors

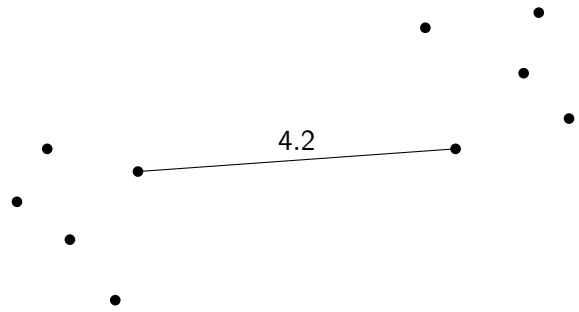


clustering

clustering points is important e.g. in ML



clustering via MST



define w via distance
build MST
for k-clustering remove $(k-1)$ heaviest edge

summary

spanning tree

MST

greedy

disjoint set data structure

applications