Graph Algorithms - Lecture 6

November 6, 2020

Table of contents

MST problem. Given $G = (V, E)$ a graph and $c : E \to \mathbb{R}$ ($c(e)$ is the cost of the edge e) find $T^* \in \mathcal{T}_G$ such that

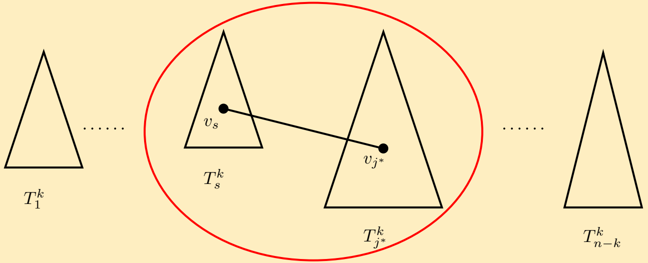
$$c(T^*) = \min_{T \in \mathcal{T}_G} c(T),$$

where $c(T) = \sum_{e \in E(T)} c(e)$.

- Start with the family $\mathcal{T}^0 = (T_1^0, T_2^0, \ldots, T_n^0)$ of disjoint trees: $T_i^0 = (\{i\}, \varnothing)$, $i = \overline{1, n}$.
- In each step k $(0 \leqslant k \leqslant n-2)$, from the family $\mathcal{T}^k = (T_1^k, T_2^k, \ldots, T_{n-k}^k)$ of $n-k$ disjoint trees such that $V = \bigcup_{i=1}^{n-k} V(T_i^k)$ and $\bigcup_{i=1}^{n-k} E(T_i^k) \subseteq E$, construct $\mathcal{T}^{k+1}$ as follows:
  - choose $T_s^k \in \mathcal{T}^k$;
  - find a minimum cost edge $e^* = v_s v_{j^*}$ from the set of edges of G with an extremity $v_s \in V(T_s^k)$ and the other in $V \setminus V(T_s^k)$ $(v_{j^*} \in V(T_{j^*}^k))$;
  - $\mathcal{T}^{k+1} = (\mathcal{T}^k \setminus \{T_s^k, T_{j^*}^k\}) \cup T$, where T is the tree obtained from T_s^k and $T_{j^*}^k$ by adding the edge e^*.

Remarks

- Note that, if, at some step, there is no edge with an extremity in $V(T_s^k)$ and the other in $V \setminus V(T_s^k)$, it follows that G is not connected and there is no MST in G.
- The above construction is suggested in the figure below:



- The family $\mathcal{T}^{n-1}$ has just one tree, T_1^{n-1}.

Theorem 1

If $G = (V, E)$ is a connected graph with $V = \{1, 2, \ldots, n\}$, then T_1^{n-1} constructed by the above algorithm is an MST of G.

Proof: We prove (by induction) that (*) $\forall k \in \{0, \ldots, n-1\}$ there exists a spanning tree T_k^*, MST of G, such that

$$E(\mathcal{T}^k) = \bigcup_{i=1}^{n-k} E(T_i^k) \subseteq E(T_k^*).$$

In particular, for $k = n-1$, $E(\mathcal{T}^{n-1}) = E(T_1^{n-1}) \subseteq E(T_{n-1}^*)$ implies $T_1^{n-1} = T_{n-1}^*$ and the theorem is proved.
For $k = 0$, we have $E(\mathcal{T}^0) = \varnothing$ and, since G is connected, there is a MST T_0^*; therefore the property (*) holds.

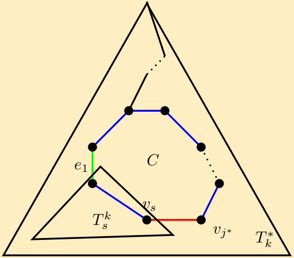
Proof cont'd. If the property (*) holds for $0 \leqslant k \leqslant n-2$, then there exists a MST of G, T_k^*, such that $E(\mathcal{T}^k) \subseteq E(T_k^*)$. By construction, $E(\mathcal{T}^{k+1}) = E(\mathcal{T}^k) \cup \{e^*\}$. If $e^* \in E(T_k^*)$, we then take $T_{k+1}^* = T_k^*$ and the property holds for $k+1$.

Suppose that $e^* \notin E(T_k^*)$. Then, $T_k^* + e^*$ has exactly one circuit C, containing $e^* = v_s v_{j^*}$. Since $v_{j^*} \notin V(T_s^k)$, it follows that there is an edge $e_1 \neq e^*$ in C with an extremity in $V(T_s^k)$ and the other in $V \setminus V(T_s^k)$. By the choosing of e^* we have $c(e^*) \leqslant c(e_1)$ and $e_1 \in E(T_k^*) \setminus E(\mathcal{T}^k)$.

Let $T^1 = T_k^* + e^* - e_1$; obviously, $T^1 \in \mathcal{T}_G$ (being connected with $n-1$ edges).

Since $e_1 \in E(T_k^*) \setminus E(\mathcal{T}^k)$, we have $E(\mathcal{T}^{k+1}) \subseteq E(T^1)$.

Proof cont'd.



On the other hand, since $c(e^*) \leqslant c(e_1)$, we have $c(T^1) = c(T_k^*) + c(e^*) - c(e_1) \leqslant c(T_k^*)$.
Because T_k^* is a MST of G, it follows that $c(T^1) = c(T_k^*)$, i. e., T^1 is a MST of G containing all adges in $E(\mathcal{T}^{k+1})$. By taking $T_{k+1}^* = T^1$ we finish the proof of the theorem. $\square$

Remarks

- The above proof remains true for tree-cost functions $c : \mathcal{T}_G \to \mathbb{R}$ satisfying: $\forall T \in \mathcal{T}_G, \forall e \in E(T), \forall e' \notin E(T)$

$$c(e') \leqslant c(e) \Rightarrow c(T + e' - e) \leqslant c(T).$$

- In the general method presented, the way of choosing the tree T_s^k is not detailed. We will discuss two famous different strategies.
- The first strategy chooses T_s^k as the maximum order tree in the family $\mathcal{T}^k$.
- In the second strategy T_s^k is one of the two trees in the family $\mathcal{T}^k$, connected by an edge of minimum cost over all edges with extremities on different trees of the family.

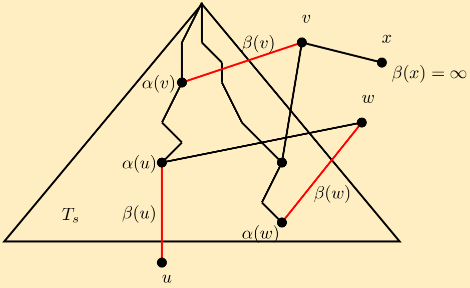
- In Prim's strategy T_s^k is the maximum order tree in the family $\mathcal{T}^k$.
- It follows that in each step $k > 0$ of the general method, $\mathcal{T}^k$ has a tree, $T_s^k = (V_s, E_s)$, with $k+1$ vertices and $n-k-1$ trees with just one vertex.
- Dijkstra's implementation: Let α and β be two vectors of size n; the elements of α are vertices from $V(G)$ and the elements of β are real numbers, with the following meaning:

(S) $$\forall j \in V \setminus V_s, \beta[j] = c(\alpha[j]j) = \min_{i \in V_s, ij \in E} c(ij)$$

Prim's algorithm

$V_s \leftarrow \{s\}$; $E_s \leftarrow \varnothing$; // for some $s \in V$.
for $(v \in V \setminus \{s\})$ do
  $\alpha[v] \leftarrow s$; $\beta[v] \leftarrow c(sv)$; // if $ij \notin E$, then $c(ij) = \infty$.
while $(V_s \neq V)$ do
  find $j^* \in V \setminus V_s$ s. t. $\beta[j^*] = \min_{j \in V \setminus V_s} \beta[j]$;
  $V_s \leftarrow V_s \cup \{j^*\}$; $E_s \leftarrow E_s \cup \{\alpha[j^*]j^*\}$;
  for $(j \in V \setminus V_s)$ do
    if $(\beta[j] > c[j^*j])$ then
      $\beta[j] \leftarrow c[j^*j]$; $\alpha[j] \leftarrow j^*$;

Remarks



- Note that (S) is satisfied after the initializations. In the while loop, the strategy of the general method is respected and also the meaning of (S) is maintained by the test in the for loop.
- Time complexity: $\mathcal{O}(n-1)+\mathcal{O}(n-2)+\cdots+\mathcal{O}(1)=\mathcal{O}(n^2)$ which is good for graphs with size $\mathcal{O}(n^2)$.

- In Kruskal's algorithm T_s^k is one of the two trees in the family $\mathcal{T}^k$, connected by an edge of minimum cost over all edges with extremities on different trees of the family.
- This choice can be done by performing firstly a non-decreasing sorting of the edges by their costs and, after that, parsing the obtained list. If we denote by T the tree T_s^k, the algorithm can be described as follows.

  sort $E = \{e_1, e_2, \ldots, e_m\}$ s. t. $c(e_1) \leqslant \ldots \leqslant c(e_m)$;
  $T \leftarrow \varnothing$; $i \leftarrow 1$;
  while $(i \leqslant m)$ do
    if $(\langle T \cup \{e_i\}\rangle_G$ has no circuits) then
      $T \leftarrow T \cup \{e_i\}$;
    $i++$;

- The sorting can be done in $\mathcal{O}(m \log m) = \mathcal{O}(m \log n)$ time.
- In order to efficiently implement the test from the while loop, it is necessary to represent the sets of vertices of the trees, $V(T_1^k), V(T_2^k), \ldots, V(T_{n-k}^k)$ (at each step k of the general method), and to test if the current edge has both extremities in the same set.
- These sets will be represented using trees (which are not, in general, subtrees of the graph G). Each such tree has a root which will be used to designate the set of vertices of the graph G stored in the tree.
- More precisely, we have a function *find*(v) which finds the set to which the vertex v belongs, that is, returns the root of the tree storing the set to which v belongs.

- In the general method it is necessary to make the (disjoint) union of the vertex sets of two trees (in order to obtain $\mathcal{T}^{k+1}$).
- We use a procedure $union(u, w)$ with the following meaning: it makes the union of the two sets of vertices, to which v and w belong.
- We can rewrite the while loop of the algorithm, using these two procedures, as follows:

  while ($i \leqslant m$) do
  let $e_i = vw$;
  if ($find(v) \neq find(w)$) then
  $union(v, w)$;
  $T \leftarrow T \cup \{e_i\}$;
  $i++$;

- The array $root[1..n]$ with entries from V has the meaning: $root[v]$ = root of the tree storing the set to which v belongs.
- Add to the initialization step (corresponding to the family $\mathcal{T}^0$):

  for ($v \in V$) do

  $\quad root[v] \leftarrow v$;
- The function *find* (having time complexity $\mathcal{O}(1)$):

  function $find(v : V)$;

  return $root[v]$;
- The procedure *union* (having time complexity $\mathcal{O}(n)$):

  procedure $union(v, w : V)$;

  for ($i \in V$) do

  $\quad$ if ($root[i] = root[v]$) then

  $\quad\quad root[i] = root[w]$;

Time complexity analysis:

- The are $\mathcal{O}(m)$ calls of the function *find during the while loop.*
- *In the sequence of find-calls, exactly $n-1$ union-calls are interleaved (one call for each edge of the final MST).*
- *Therefore, the time spent by the algorithm during the while loop is $\mathcal{O}(m\mathcal{O}(1)+(n-1)\mathcal{O}(n))=\mathcal{O}(n^2)$.*

Hence, the time complexity of the algorithm is $\mathcal{O}(max(m\log n, n^2))$. If G has many edges, $m=\mathcal{O}(n^2)$, then Prim's algorithm is more efficient.

- The array $pred[1..n]$ with entries from $V \cup \{0\}$ has the meaning $pred[v]$ =the vertex before v on the unique path to v from the root of the tree storing the set to which v belongs.
- Add to the initialization step (corresponding to the family $\mathcal{T}^0$):

  for $(v \in V)$ do
      $pred[v] \leftarrow 0$;

- The function $find(v)$ runs in $\mathcal{O}(h(v))$ time, where $h(v)$ is the length of the tree-path from v to the root (of its tree):

  function $find(v : V)$;
  $i \leftarrow v$;// a local variable.
  while $(pred[i] > 0)$ do
      $i \leftarrow pred[i]$;
  return i;

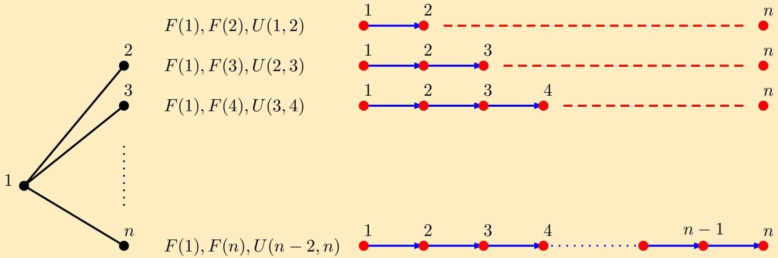
- The procedure *union* (having time complexity $\mathcal{O}(1)$) is called only for root vertices:

  procedure $union(root_1, root_2 : V)$
  $pred[root_1] \leftarrow root_2$;

- The while loop of the algorithm is changed to support this modification:

  while $(i \leqslant m)$ do
    let $e_i = vw$; $x \leftarrow find(v)$; $y \leftarrow find(w)$;
    if $(x \neq y)$ then
      $union(x, y)$;
      $T \leftarrow T \cup \{e_i\}$;
    $i++$;

If we execute this form of the while loop to the graph $G = K_{1,n-1}$ with the sorted edge list, $E = \{12, 13, \dots, 1n\}$, then the sequence of calls of the two procedures is (F and U abbreviates find and *union*):



- Hence this form of the algorithm has time complexity $\Omega(n^2)$ (even if the graph is sparse).
- The weakness of this implementation is given by the fact that, in the union procedure, we make the root of the new tree that of the tree storing a smaller number of vertices, implying an augmenting of $h(v)$ to $\mathcal{O}(n)$ during the algorithm.
- We can avoid this by keeping in the root of each tree the cardinality of the set stored. More precisely, the meaning of $pred[v]$, when v is a root, is:

$$pred[v] < 0 \Leftrightarrow v \text{ is the root of the tree}$$

$$\text{storing a set with } -pred[v] \text{vertices}$$

- The initialization step is

  for $(v \in V)$ do

  $\quad pred[v] \leftarrow -1;$

- The procedure *union* takes also $\mathcal{O}(1)$ time to maintain the new meaning:

  procedure $union(root_1, root_2 : V)$

  $t \leftarrow pred[root_1] + pred[root_2];$

  if $(-pred[root_1] \geqslant -pred[root_2])$ then

  $\quad pred[root_2] \leftarrow root_1; \; pred[root_1] \leftarrow t;$

  else

  $\quad pred[root_1] \leftarrow root_2; \; pred[root_2] \leftarrow t;$

Fact. With this implementation of the procedures find and union the algorithm has as invariant:

$$(*)\ \forall v \in V, -pred[find(v)] \geqslant 2^{h(v)}$$

In other words, the number of vertices stored in the tree to which v belongs is at least 2 to the power "distance of v to the root".

Proof of the fact. After the initialization step we have $h(v) = 0$, $find(v) = v$, and $-pred[v] = 1$, $\forall v \in V$, therefore $(*)$ holds with equality.

Suppose that $(*)$ holds before an iteration in the while loop. We have two possible cases:

- In this while iteration union is not called. The array pred is not updated, so $(*)$ remains fulfilled after this iteration.

- In this while iteration we have a call of the procedure union. Let $union(x, y)$ be this call, and suppose that in the union procedure the assignement $pred[y] \leftarrow x$ is executed. This means that before this iteration we have $-pred[x] \geqslant -pred[y]$.
  The vertices v for which $h(v)$ changes in this iteration are those for which, before the iteration we had $find(v) = y$ and $-pred[y] \geqslant 2^{h(v)}$.
  After the while iteration, we have $h'(v) = h(v)+1$ and $find'(v) = x$. Hence, we must verify if $-pred'[x] \geqslant 2^{h'v)}$. Indeed, $-pred'[x] = -pred[x] - pred[y] \geqslant 2 \cdot (-pred[y]) \geqslant 2 \cdot 2^{h(v)} = 2^{h(v)+1} = 2^{h'(v)}$.

It follows that $(*)$ is an invariant of the algorithm. □

By taking the logarithm in (*) we get

$$h(v) \leqslant \log\left(-pred[find(v)]\right) \leqslant \log n, \forall v \in V.$$

Therefore, the time complexity of the while loop is

$$\mathcal{O}(n - 1 + 2m \log n) = \mathcal{O}(m \log n).$$

Hence, this second implementation of the procedures union–find gives an $\mathcal{O}(m \log n)$ time complexity of the Kruskal's algorithm.

The time complexity of the while loop in the above solution is due to the sequence of find calls. Tarjan (1976) observed that a call with $h(v) > 1$ can, without changing the $\mathcal{O}(h(v))$ time, "collapse" the tree path from v to the root, making $h(x) = 1$ for all vertices x on the path. In this way, the future find calls for these vertices will take smaller time. More precisely, the function find is:

function $find(v : V)$; // i, j, aux are local variables.
  $i \leftarrow v$;
  while $(pred[i] > 0)$ do
    $i \leftarrow pred[i]$;
  $j \leftarrow v$;
  while $(pred[j] > 0)$ do
    $aux \leftarrow pred[j]$; $pred[j] \leftarrow i$; $j \leftarrow aux$;
  return i;

If $A : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ is the Ackermann function given by:

$$(1) \qquad A(m,n) = \begin{cases} n+1, & \text{if } m = 0 \\ A(m-1,1), & \text{if } m > 0 \text{ and } n = 0 \\ A(m-1, A(m, n-1)), & \text{if } m > 0 \text{ and } n > 0 \end{cases}$$

and if we denote, $\forall m \geqslant n > 0$,

$$\alpha(m,n) = \min\{z \;:\; A(z, 4\lceil m/n \rceil) \geqslant \log n, z \geqslant 1\}$$

we get that the time complexity of the while loop using union from the second solution and the above find, is $\mathcal{O}(m \cdot \alpha(m,n))$.
Note that $\alpha(m,n)$ is an extremely slow increasing function, and for practical values of n $\alpha(m,n) \leqslant 3$, hence, the third solution is practically a linear implementation ($\mathcal{O}(m)$) of Kruskal's algorithm.

Let $G = (V, E)$ be a (multi)graph. If $A \subseteq E$ and $v \in V$, we denote $d_A(v) = |\{e \;:\; e \in A, e \text{ incident with } v\}|$, i. e., the degree of v in the subgraph spanned by A, $\langle A \rangle_G$.

Definition

A matching (independent set of edges) in G is any set of edges $M \subseteq E$ such that

$$d_M(v) \leqslant 1, \forall v \in V.$$

The family of all matchings in the graph G is denoted $\mathcal{M}_G$:

$$\mathcal{M}_G = \{M \;:\; M \subseteq E, M \text{ matching in } G\}.$$

Note that $\mathcal{M}_G$ satisfies:

(i) $\varnothing \in \mathcal{M}_G$;

(ii) $M \in \mathcal{M}_G, M' \subseteq M \Rightarrow M' \in \mathcal{M}_G$.

Let $M \in \mathcal{M}_G$ be a matching.

- A vertex $v \in V$ with $d_M(v) = 1$ is called saturated by M, and the set of all vertices of G saturated by M is denoted $S(M)$. Obviously,

$$S(M) = \bigcup_{e \in M} e, \text{ and } |S(M)| = 2 \cdot |M|.$$

- A vertex $v \in V$ with $d_M(v) = 0$ is called exposed (with respect) to M, and the set of all vertices of G exposed to M is denoted $E(M)$. Clearly, $E(M) = V \setminus S(M)$, and $|E(M)| = |V| - 2 \cdot |M|$.

Maximum matching problem:

P$_1$ given a graph $G = (V, E)$, find $M^* \in \mathcal{M}_G$ such that

$$|M^*| = \max_{M \in \mathcal{M}_G} |M|.$$

We denote $\nu(G) = \max_{M \in \mathcal{M}_G} |M|$.

The maximum matching problem is closely related to the minimum edge-cover problem.

Definition

An edge-cover in G is any set of edges $F \subseteq E$ such that

$$d_F(v) \geqslant 1, \forall v \in V(G).$$

The family of all edge-covers in the graph G is denoted $\mathcal{F}_G$:

$$\mathcal{F}_G = \{F \ : \ F \subseteq E, F \text{ edge-cover in } G\}.$$

Note that $\mathcal{F}_G$ has the following properties

(i) $\mathcal{F}_G \neq \varnothing \Leftrightarrow G$ has no isolated vertices (then $E \in \mathcal{F}_G$);

(ii) $F \in \mathcal{F}_G, F' \supseteq F \Rightarrow F' \in \mathcal{F}_G$.

Minimum edge-cover problem:

P_2 given a graph $G = (V, E)$, find $F^* \in \mathcal{F}_G$ such that

$$|F^*| = \min_{F \in \mathcal{F}_G} |F|.$$

Theorem 2

(Norman-Rabin, 1959)Let $G = (V, E)$ be a graph of order n, without isolated vertices. If M^* is a maximum cardinality matching in G and F is a minimum cardinality edge-cover in G, then

$$|M^*| + |F^*| = n.$$

Proof: "$\leqslant$" Let M^* be maximum cardinality matching in G and consider the following algorithm:

$F \leftarrow M^*$;
for $(v \in E(M^*))$ do
  find $v' \in S(M^*)$ s. t. $vv' \in E$;
  $F \leftarrow F \cup \{vv'\}$;

Proof cont'd.

Note that, $\forall v \in E(M^*)$, since G has no isolated vertices, there is an edge incident with v, and, since M^* is maximal w. r. t. inclusion, this edge has the other end in $S(M^*)$.

The set F of edges constructed is an edge-cover and $|F| = |M^*| + |E(M^*)| = |M^*| + n - 2 \cdot |M^*| = n - |M^*|$. Hence

(2) $$|F^*| \leqslant |F| = n - |M^*|.$$

"$\geqslant$" Let F^* be minimum cardinality edge-cover in G and consider the following algorithm:

$M \leftarrow F^*$;
for $(\exists v \in V : d_M(v) > 1)$ do
  find $e \in M$ incident with v;
  $M \leftarrow M \setminus \{e\}$;

Proof cont'd.

Obviously, the algorithm constructs a matching M in G. If the edge e incident with v (and removed from M in a while iteration) is $e = vv'$, then $d_M(v') = 1$ and in the next iteration we will have $d_M(v') = 0$, hence at each while iteration an exposed vertex w.r.t. the final matching M is created (if there would be another edge e' in the current set M incident with v', then, since $e \in F^*$, it follows that $F^* \setminus \{e\}$ would be an edge-cover, contradicting the choice of F^*).

Hence, if M is the matching constructed by the algorithm, then $|F^*| - |M| = |E(M)| = n - 2 \cdot |M|$, i. e.,

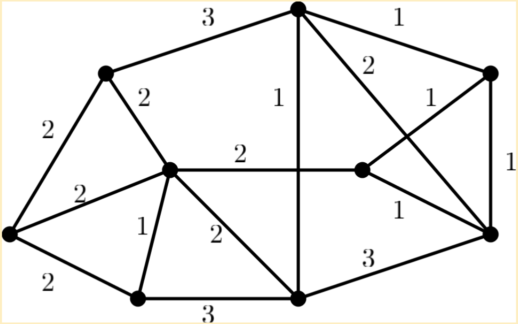
$$|F^*| = n - |M| \geqslant n - |M^*|. \tag{3}$$

From (2) and (3) the theorem follows. □

Remark

Note that we have proved that the two problems P_1 and P_2 are polynomially equivalent since the matching M and the edge-cover F constructed are optimal solutions, respectively.

Exercise 1'. Find a minimum cost spanning tree in the following graph.



Exercise 1. Let $G = (V, E)$ be a connected graph and $c : E \to \mathbb{R}$ a weight function on its edges. A subset $A \subseteq E$ is called *cut* if exists a bipartion (S, T) of V such that $A = \{uv \in E : u \in S, v \in T\}$ ($G \setminus A$ is no more connected).

(a) If in every *cut* it exists only one edge of minimum cost, then G contains only one minimum cost spanning tree.

(b) Show that, if c is an injective function, then G contains only one minimum cost spanning tree.

(c) Are true the reciprocal assertions?

Exercise 2. Let $G = (V, E)$ be a connected graph of order n, $c : E \to \mathbb{R}$, and $\mathcal{T}_G^{min}$ the family of its (c related) minimum cost spanning trees. Define $H = (\mathcal{T}_G^{min}, E(H))$ where $T_1T_2 \in E(H) \iff |E(T_1) \Delta E(T_2)| = 2$. Prove that H is connected and its diameter is at most $n - 1$.

Exercises for the next week seminar

Exercise 3. Let $G = (V, E)$ be a connected graph and $c : R \to \mathbb{R}$. For any spanning tree $T = (V, E') \in \mathcal{T}_G$, and $v \neq w \in V$ we denote by P_{vw}^T the unique vw-path in T. Show that a spanning tree $T^* = (V, E^*)$ has a minimum cost if and only if

$$\forall\, e = vw \in E \setminus E^*, \forall\, e' \in E(P_{vw}^{T^*}), \text{ we have } c(e) \geqslant c(e').$$

Exercise 4. Let $G = (V, E)$ be a 2-edge-connected graph and $c : E \to \mathbb{R}$. If $T = (V, E')$ is minimum cost spanning tree of G and $e \in E'$, $T - e$ has exactly two connected components T_1' and T_2' respectively. We denote by $e_T \neq e$ a minimum weight edge in the cut generated by $(V(T_1'), V(T_2'))$ in $G - e$. Show that, if T^* is a minimum cost spanning tree of G, and $e \in E(T^*)$, then $T^* - e + e_{T^*}$ is a minimum cost spanning tree of $G - e$.

Exercise 5. Let $G = (V, E)$ be a connected graph and $c : E \to \mathbb{R}$ an injective weight on its edges. Let us consider the following algorithm

for $(e \in E)$ do
  $\gamma(e) \leftarrow r$; // all the edges are colored red; during the execution they will be red, blue or green
while $((\exists A \subseteq E$, a cut, s. t. $\gamma(e') \neq g$, where $c(e') = \min_{e \in A} c(e))$ or
    $(\exists C$, a cycle, s. t. $\gamma(e') \neq b$, where $c(e') = \max_{e \in C} c(e)))$ do
  for a cut, A, $\gamma(e') \leftarrow g$;
  for a cycle, C, $\gamma(e') \leftarrow b$;
return $H = (V, \{e \in E : \gamma(e) = g\})$;

Prove that

(a) an edge belongs to a minimum cost spanning tree if and only if it is of minumum cost in a certain cut;

(b) an edge doesn't belong to any minimum cost spanning tree of G if and only if it is of maximum cost on a certain cycle;

Exercise 5 (cont'd).

(c) the algorithm doesn't end as long as there are remaining red edges;

(d) the algorithm ends for any choice of the edges e' and H is the only minimum cost spanning tree in G.

Exercise 6. Let H be a connected graph, $\varnothing \neq A \subseteq V(H)$, and $w : E(H) \to \mathbb{R}_+$. A Steiner tree for (H, A, w) is a tree $T(H, A, w) = (V_T, E_T) \subseteq H$ with $A \subseteq V_T$ which has the minimum weight between all the trees containing A, and which are subgraphs of H:

$$s[T(H, A, w)] = \sum_{e \in E_T} w(e) =$$

$$= \min \left\{ \sum_{e \in E_{T'}} w(e) \; : \; T' = (V_{T'}, E_{T'}) \text{ tree in } H, A \subseteq V_{T'} \right\}$$

(a) Prove that a Steiner tree can be determined in polynomial time complexity if $A = V(H)$ or $|A| = 2$.

Exercise 6 (cont'd).

(b) Let $G = (V, E)$ be a connected graph with $V = \{1, 2, \ldots, n\}$, and $A \subseteq V$; we also have a cost function $c : E \to \mathbb{R}_+$. Consider the complete graph K_n (with $V(K_n) = V$) and define $\overline{c} : E(K_n) \to \mathbb{R}_+$:

$$\overline{c}(ij) = \min \left\{ c(P) = \sum_{e \in E(P)} c(e) \;:\; P \text{ is } ij\text{-path in } G \right\}$$

Prove that $s[T(G, A, c)] = s[T(K_n, A, \overline{c})]$ and show how to build a Steiner tree $T(K_n, A, \overline{c})$ starting from a Steiner tree $T(G, A, c)$.

(c) Show that it exists a Steiner tree $T(K_n, A, \overline{c})$ such that all its vertices from outside A has degree at least 3. Using this property prove that there exists a Steiner tree $T(K_n, A, \overline{c})$ having at most $2|A| - 2$ vertices.

Exercise 7. Consider an ordering $E = \{e_1, e_2, \ldots, e_m\}$ of the edges of a connected graph $G = (V, E)$ of order n. For every subset $A \subseteq E$ we define $\mathrm{x}^A \in GF^m$ its m-dimensional characteristic vector: $\mathrm{x}_i^A = 1 \Leftrightarrow e_i \in A$. GF^m is the m-dimensional vector space over $\mathbb{Z}_2$.

(a) Show that the subset of the characteristic vectors corresponding to all the cuts in G completed with zero vector is a subspace X of GF^m.

(b) Show that the subset of the characteristic vectors corresponding to all the circuits in G spans a subspace U of GF^m which is orthogonal on X.

(c) Prove that $dim(X) \geqslant n - 1$.

(d) Show that $dim(U) \geqslant m - n + 1$.

(e) Finally, prove that the above inequalities are in fact equalities.

Exercises for the next week seminar

Exercise 8. Let $G = (V, E)$ be a connected graph and $c : E \to \mathbb{R}$ a cost function on its edges.

a) Let T^* be a minimum c-cost spanning tree of G and $\epsilon > 0$. Prove that T^* is the only minimum $\overline{c}$-cost spanning tree of G, where

$$\overline{c}(e) = \begin{cases} c(e) - \epsilon, & \text{if } e \in E(T^*) \\ c(e), & \text{otherwise} \end{cases}$$

b) Deduce from here that for every minimum spanning tree, T^*, of G there exists an ordering of the edges in G such that Kruskal's algorithm returns T^*.

Exercise 9. Let $G = (V, E)$ be a connected graph and $c : E \to \mathbb{R}$ an injective cost function on its edges. Let T^* be the minimum cost spanning tree of G and T_0 be a second-best minimum cost spanning tree in G.

(a) Is T_0 the only second-best minimum cost spanning tree in G?

(b) Prove that $|E(T^*) \Delta E(T_0)| = 2$.

(c) Devise an algorithm to find the second-best minimum cost spanning tree in G.

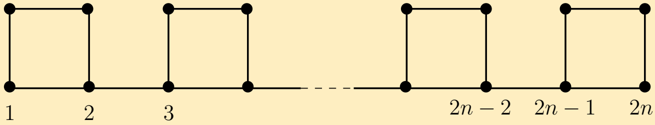
Exercise 10. Let $G = (V, E)$ be a connected graph and $c : E \to \mathbb{R}$ be a cost function on its edges. True or false? (Justify your answers!)

(a) Any edge of minimum cost in G is contained in a certain minimum cost spanning tree (MST) of G.

(b) If G has a cycle, C, whose minimum cost edge is unique on C, then that edge must be contained in every MST of G.

(c) If an edge is contained in a certain MST of G, then that edge must be of minimum cost in a certain cut of G.

Exercise 11. Find the number of maximum matchings in the following graph:

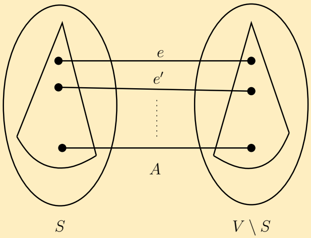


Exercise 12. Two kids play the following game on a given graph G: they alternatively pick a new vertex $v_0, v_1, \ldots$ such that, for every $i > 0$, v_i is adjacent with v_{i-1}. The player which can not choose another vertex will loose the game. Prove that the player which starts the game has always a winning strategy if and only if G has not a perfect matching.

Exercise 13. Let S be a non-empty, finite set, $k \in \mathbb{N}^*$, and $\mathcal{A} = (A_i)_{1 \leqslant i \leqslant k}$ and $\mathcal{B} = (B_i)_{1 \leqslant i \leqslant k}$ two partitions of S. Prove that $\mathcal{A}$ and $\mathcal{B}$ admits a common set of representatives, i. e., there exist $r_{\mathcal{A}}, r_{\mathcal{B}} : \{1, 2, \ldots, k\} \to S$ such that for every $1 \leqslant i \leqslant k$, $r_{\mathcal{A}}(i) \in A_i$ and $r_{\mathcal{B}}(i) \in B_i$, and the two functions have the same image.

Exercise 1. Solution.

(a) By deleting an edge e from a spanning tree T of G, we get exactly two connected components (subtrees of T - why?): one with vertex set S and other with $V \setminus S$; e belongs to the cut generated by the bipartition $(S, V \setminus S)$:



- If T is a minimum cost spanning tree, then e is the edge with minimum cost in the corresponding cut A (why?).
- Suppose on the contrary that there exist two minimum cost spanning trees $T_1 = (V, E_1) \neq T_2 = (V, E_2)$. Let $e_1 \in E_1 \setminus E_2$.
- $T_1 - e_1$ has two connected components with vertex sets S and $V \setminus S$.
- In the cut A generated by $(S, V \setminus S)$, e_1 is the minimum cost edge.
- On the other hand, $T_2 + e_1$ contains only one cycle C (why?).
- C meets the cut in at least one another edge (why?), say $e_2 \in E_2 \setminus E_1$.
- $T_2 + e_1 - e_2$ is a spanning tree (why?) with $c(T_2 + e_1 - e_2) < c(T_2)$ - contradiction.

(b) and (c) are consequences of (a). Why?

Exercise 4. Solution. Let T_0 be a minimum cost spanning tree in $G - e$.

- $c(T_0) \leqslant c(T^* - e + e_{T^*})$ (why?).
- $T_0 + e$ contains exactly one cycle C which meets the cut generated by $(V(T_1'), V(T_2'))$ in, at least, another edge $e_0 \neq e$ (why?).
- $c(e_0) \geqslant c(e_{T^*})$ (why?).
- $T_0 + e - e_0 \in \mathcal{T}_G$, hence $c(T_0 + e - e_0) \geqslant c(T^*)$ and

$$c(T^* - e + e_{T^*}) \geqslant c(T_0) \geqslant c(T^*) - c(e) + c(e_0) \geqslant c(T^*) - c(e) + c(e_{T^*})$$

- Thus, the above inequalities have to be equalities and $T^* - e + e_{T^*}$ must be a minimum cost spanning tree in $G - e$.

Exercise 5. Solution. Let $T^* = (V, E^*)$ be a minimum cost spanning tree of G.

(a) "$\Rightarrow$" Let $e \in E^*$; $T^* - e$ is a forest with two subtrees T_1^* and T_2^*.

- In the cut generated by $V(T_1^*)$ and $V(T_2^*)$ we consider a minimum cost edge e'.
- Suppose, on the contrary, that $e' \neq e$, then $c(e) > c(e')$, but $T = T^* - e + e' \in \mathcal{T}_G$ and $c(T) < C(T^*)$ (why?) - contradiction.

"$\Leftarrow$" Let A be a cut and $e' \in A$ with $c(e') = \min\limits_{e \in A} c(e)$.

- Suppose, on the contrary, that $e' \notin E^*$.
- Then $T^* + e'$ contains a cycle C and let $e'' \in E(C) \cap A$ be an edge different from e'.
- $c(e') < c(e'')$ and $T^* + e' - e'' \in \mathcal{T}_G$ with $c(T^* + e' - e'') < c(T^*)$ (why?) - contradiction.

(b) "$\Rightarrow$" If $e \notin E^*$, then on the cycle C of $T^* + e'$, e' is of maximum cost. Why?

"$\Leftarrow$" If e' is of maximum cost on a cycle C, and $e' \in E^*$, then $T^* - e'$ is a forest with two subtrees T_1^* and T_2^*.

- In the cut generated by $V(T_1^*)$ and $V(T_2^*)$ C must contain another edge e'' (why?).
- Since $c(e') > c(e'')$ we get $T = T^* - e' + e'' \in \mathcal{T}_G$ and $c(T) < c(T^*)$ - contradiction.

(c) Suppose that we have a red edge $e = uv$; define

$$P = \{x \in V \ : \ \text{there exists an all-green path from } u \text{ to } x \text{ in } G\}.$$

- If $v \in P$, then $C = P_{uv} + e$ is a cycle on which e is of maximum cost (why?) and will be blue colored by the algorithm.
- If $v \notin P$, then the cut generated by P and $V \setminus P$ doesn't contain green edges (why?), hence the algorithm can green color an edge from it.

(d) Edges cannot be recolored (why?), hence the while loop ends after at most $|E|$ iterations when we have no more red edges.

- At the end of the algorithm we have only blue and green edges, the blue ones doesn't belong to T^*, hence all the blue edges are in T^*.
- As a by-product: T^* is the only minimum cost spanning tree in G (why?).

Exercise 7. Solution.

(a) We have to prove that, for every to cuts A_1, A_2 and $\alpha_1, \alpha_2 \in \{0, 1\}$, we have $\alpha_1 \mathrm{x}^{A_1} + \alpha_2 \mathrm{x}^{A_2} = \mathrm{x}^A$, where A is a cut or the empty set.

- Obviously $A = \varnothing \Leftrightarrow A_1 = A_2$ (why?).
- Since $\alpha_1, \alpha_2 \in \mathbb{Z}_2$, it will be sufficient to show that, if $A_i = \{uv \in E : u \in S_i, v \in T_i\}$, $i = \overline{1, 2}$, are distinct cuts, then $\mathrm{x}^{A_1} + \mathrm{x}^{A_2} = \mathrm{x}^A$, where A is a cut in G ((S_i, T_i) are bipartitions of V).
- It is easy to see that $A = A_1 \Delta A_2$ (why?).
- One can prove that A is equal with the cut $\{uv \in E : u \in S, v \in T\}$, where $S = S_1 \Delta S_2$, $T = V \setminus S$. How?

(Partially) solved exercises

(b) Since $U = span\left(\{\mathrm{x}^{E(C)} \ : \ C \text{ cycle in } G\}\right)$, it suffices to prove that for every cycle C and every cut A, $\mathrm{x}^A \perp \mathrm{x}^{E(C)}$.

- If we choose a cut A and a cycle C, it easy to see that $|A \cap E(C)| \equiv 0 \pmod 2$ (why?).
- Hence

$$\langle \mathrm{x}^A, \mathrm{x}^{E(C)} \rangle \equiv \sum_{i=1}^{m} \mathrm{x}_i^A \mathrm{x}_i^{E(C)} \overset{\text{why?}}{=} |A \cap E(C)| \equiv 0 \pmod 2.$$

(c) We will produce $(n-1)$ linearly independent vectors from X.

- For every $v \in V$, let $A_v = \{vw \in E \ : \ w \neq v\} \neq \varnothing$.
- If $V = \{v_1, v_2, \ldots, v_n\}$, then $\mathrm{x}^{A_1}, \mathrm{x}^{A_2}, \ldots, \mathrm{x}^{A_{n-1}}$ are independent. Let $\{i_1, i_2, \ldots, i_k\} \subseteq \{1, 2, \ldots, n-1\}$.
- Since G is connected, there exists an edge $e_h = v_{i_l} v_q \in E$, where $q \notin \{i_1, i_2, \ldots, i_k\}$ (why?).
- We have

$$\left(\sum_{j=1}^{k} \mathrm{x}^{A_{i_j}} \right)_h = 1, \text{ thus } \sum_{j=1}^{k} \mathrm{x}^{A_{i_j}} \neq 0 \text{ (why?)}.$$

- A base in X will have at least $n-1$ vectors, hence $dim(X) \geqslant n-1$.

(d) We will now produce $(m-n+1)$ linearly independent vectors from U.

- Take T to be a spanning tree of G; for any $e \in E \setminus E(T)$, $T+e$ contains exactly one cycle denoted by C_e.
- Let $\{e_{i_1}, e_{i_2}, \ldots, e_{i_p}\} \subseteq E \setminus E(T)$.
- Obviously $e_{i_1} \notin E(C_{e_{i_j}})$, for $2 \leqslant j \leqslant p$ so

$$\left(\sum_{j=1}^{p} \mathrm{x}^{E(C_{e_{i_j}})}\right)_{i_1} = 1, \text{ thus } \sum_{j=1}^{p} \mathrm{x}^{E(C_{e_{i_j}})} \neq 0 \text{ (why?)}.$$

- In this way $dim(U) \geqslant m-n+1$.

(e) We have proved that

$$dim(X) + dim(U) \geqslant m = dim(GF^m)$$

- We further know that $U \subseteq X^\perp$ (why?).
- Hence $dim(U) \leqslant dim(X^\perp) = dim(GF^m) - dim(X) \geqslant dim(U)$.
- This means $dim(X) + dim(U) = m$, and, from here, we can get the desired equalities.

Exercise 9. Solution. If c is injective, then there exists only one minimum cost spanning tree in G (see ex. 1).

(a) No. $G \equiv K_4$, $V(G) = \{x, y, z, t\}$; $c(xy) = 2$, $c(yz) = 4$, $c(zt) = 3$, $c(tx) = 1$, $c(xz) = 5$, and $c(yt) = 6$ ...

(b) Obviously, $p = |E(T^*) \setminus E(T_0)| = |E(T_0) \setminus E(T^*)| = |E(T^*) \Delta E(T_0)|/2$.

- Suppose, on the contrary, that $k \geqslant 2$ and let e_1 be an edge of minimum cost from $E(T^*) \setminus E(T_0)$.
- $T_0 + e_1$ contains a cycle C and there exists an edge $e_0 \in E(C) \cap (E(T_0) \setminus E(T^*))$.
- We will prove that $c(e_0) > c(e_1)$. If not, $T^* + e_0$ contains a cycle C' and there exists an edge $e_2 \in E(C') \cap (E(T^*) \setminus E(T_0))$.
- $T' = T^* + e_0 - e_2 \in \mathcal{T}_G$, hence $c(T') < c(T^*)$ and $c(e_2) < c(e_0) < c(e_1)$ - contradiction (why?).

- Thus, $c(e_0) > c(e_1)$. The tree $T'' = T_0 + e_1 - e_0 \in \mathcal{T}_G$ has the cost $c(T'') < c(T_0)$.
- It will follow that T'' is minimum cost spanning tree, but $T'' \neq T^*$ - contradiction (why?).

(c) Apply an algorithm for finding a minimum cost spanning tree for any graphs $G - e$, $\forall e \in E(T^*)$ and retain the minimum cost spanning tree obtained.

- The time complexity is?
- Remark: some of the above graphs may not be connected.

Exercise 10. Solution.

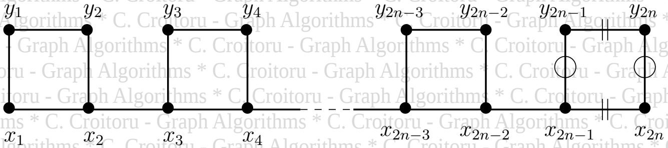
(a) True, by Kruskal's algorithm (how?). True (why?).

(b) False (why?).

(c) True (why?).

Exercise 11. Solution.

- Let $\#_{max}(G_n)$ be the number of maximum matchings in G_n.
- One can use induction on n to prove that the number of perfect matchings in the above graph is 2^n.



Exercise 13. Solution. Let $G = (S, T; E)$ be the following bipartite graph: $T = \mathcal{A}$, $S = \mathcal{B}$, and $A_iB_j \in E$ if $A_i \cap B_j \neq \varnothing$.

- Use the Hall's theorem (how?).