The Minimum Spanning Tree (MST) problem in graphs with selfish edges

Recap

- VCG-mechanism: pair $M=\langle g,p\rangle$ where
  - $g(r) = \arg\max_{y\in X} \sum_i v_i(r_i,y)$
  - $p_i(g(r)) = -\sum_{j\neq i} v_j(r_j,g(r_{-i})) + \sum_{j\neq i} v_j(r_j,g(r))$
- VCG-mechanisms are truthful for utilitarian problems
- The classic shortest-path problem on (private-edge) graphs is utilitarian $\Rightarrow$ we showed an efficient $O(m+n \log n)$ time implementation of the corresponding VCG-mechanism:
  - $g(r)$ = compute a shortest-path
  - $p_e(g(r))$ = pays for the marginal utility of e (difference between the length of a replacement shortest path in $G-e$ and the length of a shortest path in G)

Another very well-known problem: the Minimum Spanning Tree problem

- INPUT: an undirected, weighted graph $G=(V,E,w)$, $w(e) \in R^+$ for any $e \in E$, with n nodes and m edges
- OUTPUT: a minimum spanning tree (MST) $T=(V,E_T)$ of G, namely a spanning tree of G having minimum total weight $w(T)=\sum_{e \in E_T} w(e)$
- Recall: T is a spanning tree of G if:
  1. T is a tree
  2. T is a subgraph of G
  3. T contains all the nodes of G
- Fastest centralized algorithm costs $O(m\,\alpha(m,n))$ time (B. Chazelle, A minimum spanning tree algorithm with Inverse-Ackermann type complexity. J. ACM 47(6): 1028-1047 (2000)), where α is the inverse of the Ackermann function

The Ackermann function $A(i,j)$ and its inverse $\alpha(m,n)$

Notation: By a^{b^c} we mean $a^{(b^c)}$, and not $(a^b)^c = a^{b \cdot c}$.

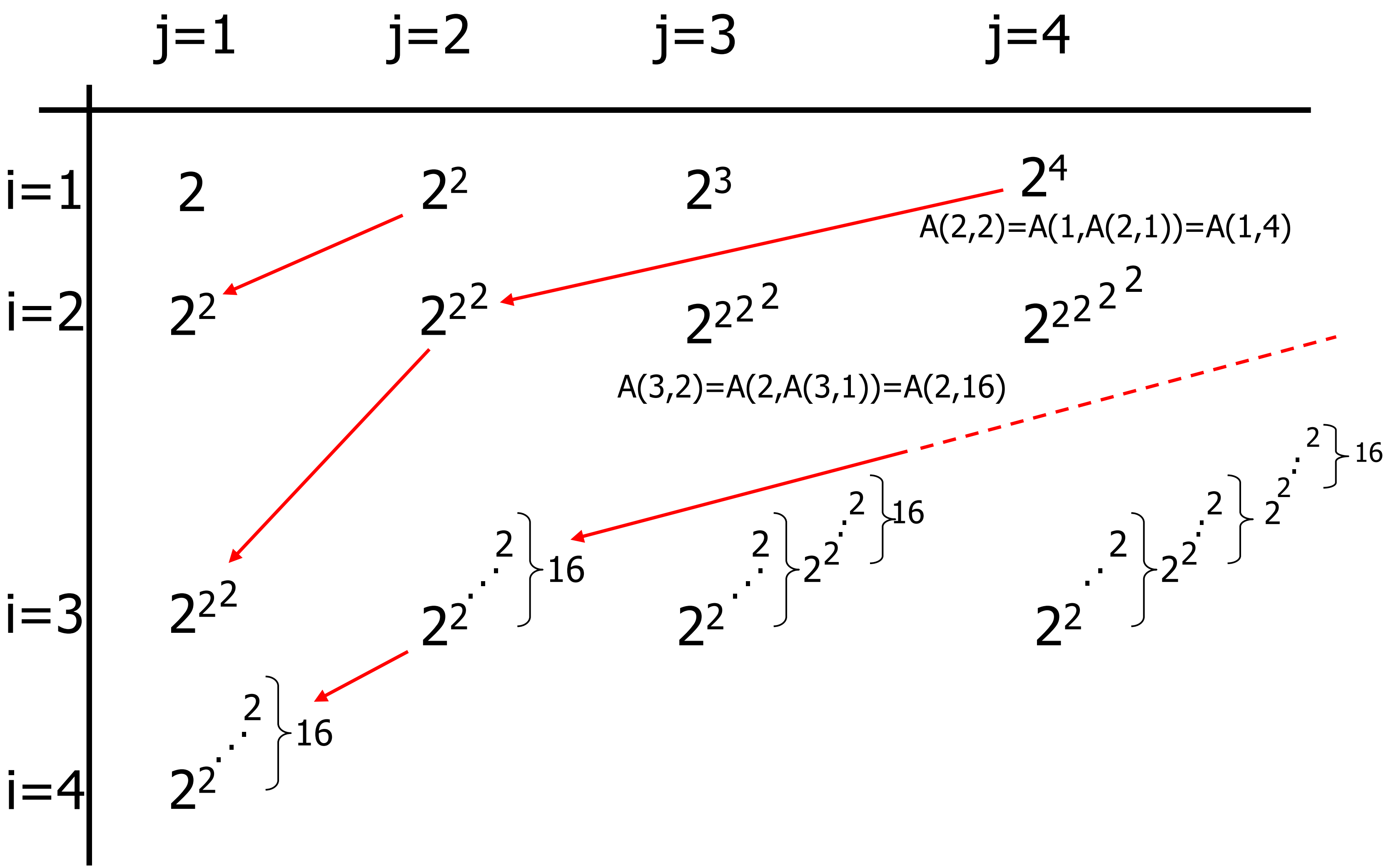
For integers $i,j \geq 1$, let us define $A(i,j)$ as:

$$A(1,j) = 2^j \qquad j \geq 1;$$

$$A(i,1) = A(i-1,2) \qquad i \geq 2;$$

$$A(i,j) = A(i-1, A(i,j-1)) \qquad i,j \geq 2.$$

A(i,j) for small values of i and j



The $\alpha(m,n)$ function

For integers $m \geq n \geq 0$, let us define $\alpha(m,n)$ as:

$$\alpha(m,n) = \min\{i \geq 1 | \ A(i, \lfloor m/n \rfloor) > \log_2 n\}.$$

Properties of $\alpha(m,n)$

1. For fixed n, $\alpha(m,n)$ is monotonically decreasing for increasing m

$$\alpha(m,n) = \min\{i>0 : A(i, \underbrace{\lfloor m/n \rfloor}_{\text{growing in } m}) > \log_2 n\}$$

2. $\alpha(n,n) \to \infty$ for $n \to \infty$

$$\alpha(n,n) = \min\{i>0 : A(i, \lfloor n/n \rfloor) > \log_2 n\}$$

$$= \min\{i>0 : A(i, 1) > \underbrace{\log_2 n}_{\to \infty}\}$$

Remark

$\alpha(m,n) \leq 4$ for any practical purposes (i.e., for reasonable values of n)

$\alpha(m,n) = \min\{i>0 : A(i, \lfloor m/n \rfloor) > \log_2 n\}$

$A(4, \lfloor m/n \rfloor) \geq A(4,1) = A(3,2)$

$= 2^{2^{\cdot^{\cdot^{\cdot^{2}}}}} \Big\} 16 \quad >> 10^{80}$ $\cong$ estimated number of atoms in the universe!

$\Rightarrow$ hence, $\alpha(m,n) \leq 4$ for any $n < 2^{10^{80}}$

The private-edge MST problem

- **Input**: a 2-edge-connected, undirected graph G=(V,E) such that each edge is owned by a distinct selfish agent; we assume that agent's private type t(e) is the positive cost of the edge e she owns, and her valuation function is equal to her type if the edge is selected in the solution, and 0 otherwise.
- **Question**: design an efficient (in terms of time complexity) truthful mechanism in order to find a MST of G_t=(V,E,t)

VCG mechanism

The problem is utilitarian (indeed, the cost of a solution is given by the sum of the valuations of the selected edges) ⇒ **VCG-mechanism** $M=\langle g,p\rangle$:

- g: computes a MST $T=(V,E_T)$ of $G=(V,E,r)$; let $r(T)$ denote its weight;
- p_e: For any edge $e\in E$, $p_e=-\sum_{i\neq e} v_i(r_i,g(r_{-e}))+\sum_{i\neq e} v_i(r_i,g(r))$, namely

$p_e=r(T_{G-e}) - [r(T)-r(e)]$ if $e\in E_T$

$p_e=0$ otherwise.

Remark: $u_e = p_e+v_e= p_e- t_e = p_e- r(e) =$
$r(T_{G-e})-r(T)+ r(e) - r(e)$, and since $r(T_{G-e}) \geq r(T) \Rightarrow u_e\geq 0$

⇒ For any $e\in T$ we have to compute T_{G-e}, namely the replacement MST for e (MST in $G-e=(V,E\setminus\{e\},r_{-e})$)

Remark: G is 2-edge-connected since otherwise a bridge edge e would imply that T_{G-e} does not exist, and so $r(T_{G-e})$ is undefined ⇒ according to the payment scheme, agent owning e would get an unbounded payment!

A trivial solution

1. First, we compute a MST of G
2. Then, $\forall e \in T$ we compute a MST of $G-e$

Time complexity: we pay $O(m\,\alpha(m,n))$ for step 1, and $O(m\,\alpha(m,n))$ for each of the $n-1$ edges of the MST in step 2

$\Rightarrow O(nm\,\alpha(m,n))$ total time

We will show an efficient solution costing $O(m\,\alpha(m,n))$ time!!!

A related problem: MST sensitivity analysis

- Input
  - $G=(V,E,w)$ weighted and undirected
  - $T=(V,E_T)$ MST of G
- Question
  - For any $e \in E_T$, how much $w(e)$ can be increased until the minimality of T is affected?
  - For any $f \notin T$, how much $w(f)$ can be decreased until the minimality of T is affected? (we will not be concerned with this aspect)
- The first question is exactly what we are looking for to compute the marginal utility (i.e., the payment) of an edge selected in a solution!

Computing the sensitivity of a tree edge

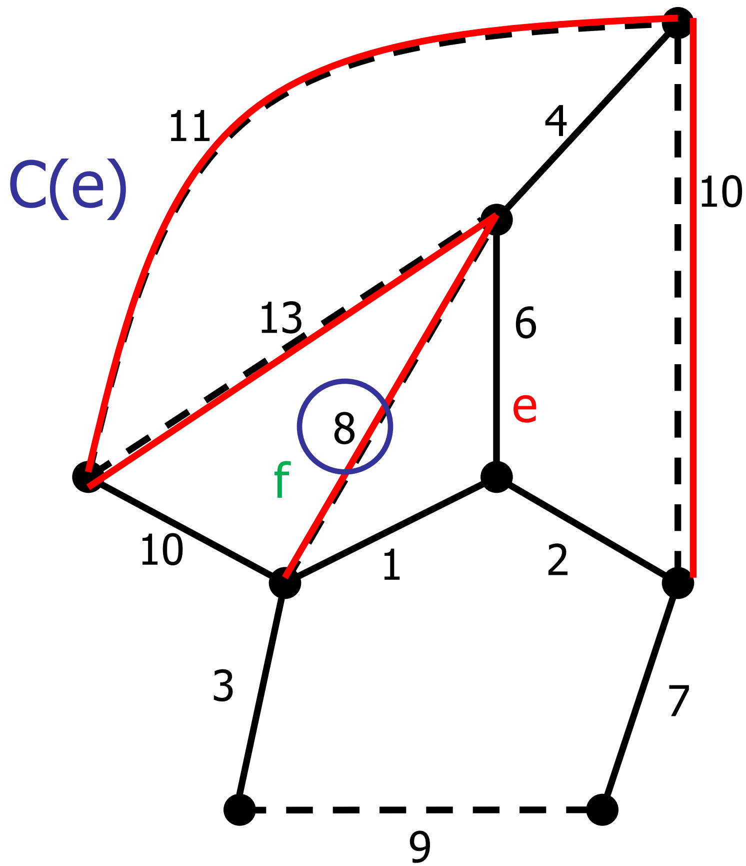
$G=(V,E)$, T any spanning tree of G. We define:

- For any non-tree edge $f=(x,y)\in E\setminus E(T)$
  - $T(f)$: (unique) simple path in T joining x and y (a.k.a. the fundamental cycle of f w.r.t. T)
- For any tree-edge $e\in E(T)$
  - $C(e)=\{f\in E\setminus E(T): e\in T(f)\}$; notice that $C(e)$ contains all the non-tree edges that cross the cut induced by the removal of e from T; we will call them crossing edges (w.r.t. the tree edge e)

From the classic **blue rule** in a MST…

- If e is an edge of the MST T, then T remains minimal until $w(e) \leq w(f)$, where f is the cheapest crossing edge w.r.t. e (f is called a swap edge for e); let us call this value up(e)
- More formally, for any $e \in E(T)$
  - $up(e) = \min_{f \in C(e) = \{f \in E \setminus E(T):\ e \in T(f)\}} \{w(f)\}$
  - $swap(e) = \arg\min_{f \in C(e)} \{w(f)\}$

MST sensitivity analysis



Edge e can increase its cost up to 8 before being replaced by edge f

up(e)=8
swap(e)=f

Remark

- Computing all the values up(e) is equivalent to compute a MST of G-e for any edge e in the MST T of G; indeed

$$w(T_{G-e})=w(T)-w(e)+up(e)$$

⇒ In the VCG-mechanism, the payment p_e of an edge e in the solution is exactly up(e), where now the graph is weighted w.r.t. r

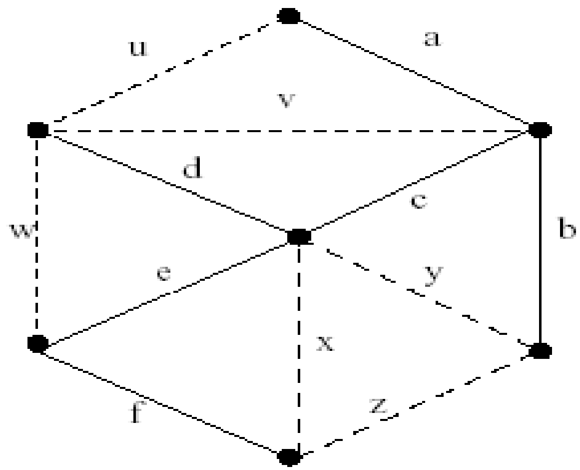
Idea of the efficient algorithm

- From the above observations, it is easy to devise an $O(mn)$ time implementation for the VCG-mechanism: just compute a MST T of $G=(V,E,r)$ in $O(m\,\alpha(m,n))$ time, and then $\forall e \in T$ compute $C(e)$ and $up(e)$ in $O(m)$ time (can you see the details of this step?)
- In the following, we sketch how to boil down the overall complexity to $O(m\alpha(m,n))$ time by checking efficiently all the non-tree edges which form a cycle in T with e

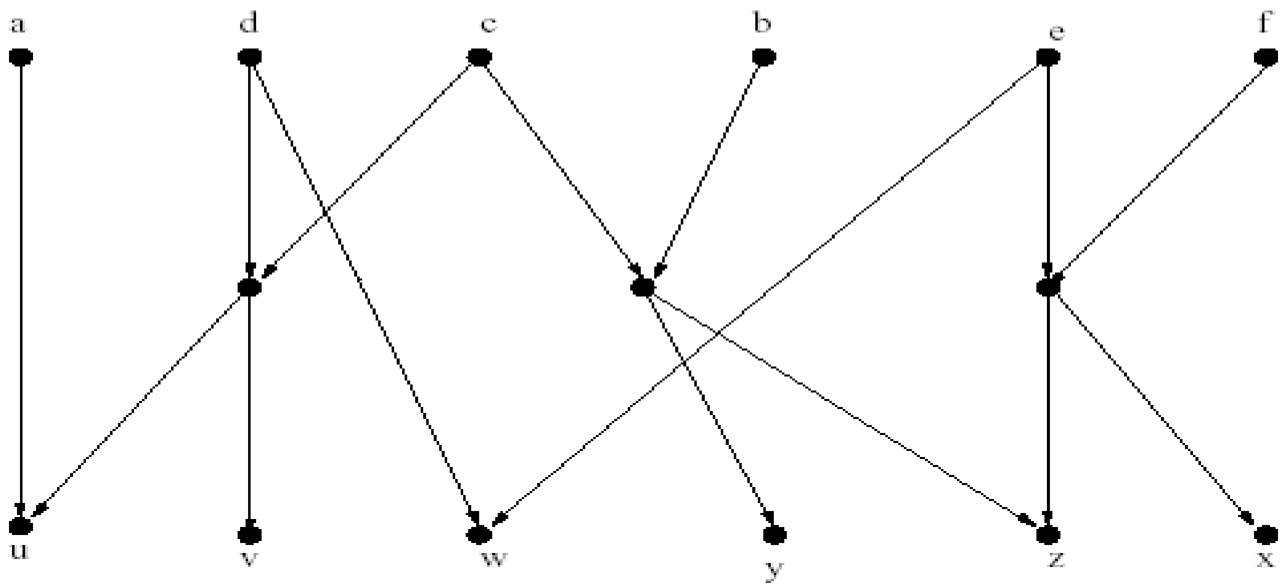
The Transmuter

- Given a graph G=(V,E,w) and a spanning tree T of G, a transmuter D(G,T) is a *directed acyclic graph* (DAG) representing in a compact way the set of all fundamental cycles of T w.r.t. G, namely {T(f) : f is not in T}
- D will contain:
  1. A source node (in-degree=0) s(e) for any edge e in T
  2. A sink node (out-degree=0) t(f) for any edge f not in T
  3. A certain number of auxiliary nodes of in-degree=2 and out-degree not equal to zero.
- **Fundamental property**: there is a path in D from s(e) to t(f) iff $e \in T(f)$

An example



(a)



How to build a transmuter

- It has been shown that for a graph of n nodes and m edges, a transmuter contains $O(m\,\alpha(m,n))$ nodes and edges, and can be computed in $O(m\,\alpha(m,n))$ time:

R. E. Tarjan, Application of path compression on balanced trees, J. ACM 26 (1979) pp 690-715

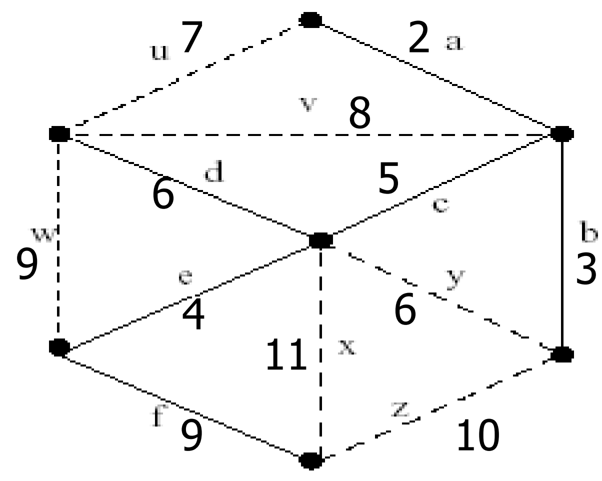
Topological sorting

- Let D=(V,A) be a directed graph. Then, a topological sorting of D is a numbering $v_1, v_2, \ldots, v_{n=|V|}$ of the vertices of D s.t. if there exists a directed path from v_i to v_j in D, then we have $i<j$.
- D has a topological sorting iff is a DAG
- A topological sorting, if any, can be computed in $O(|V|+|A|)$ time (homework!).

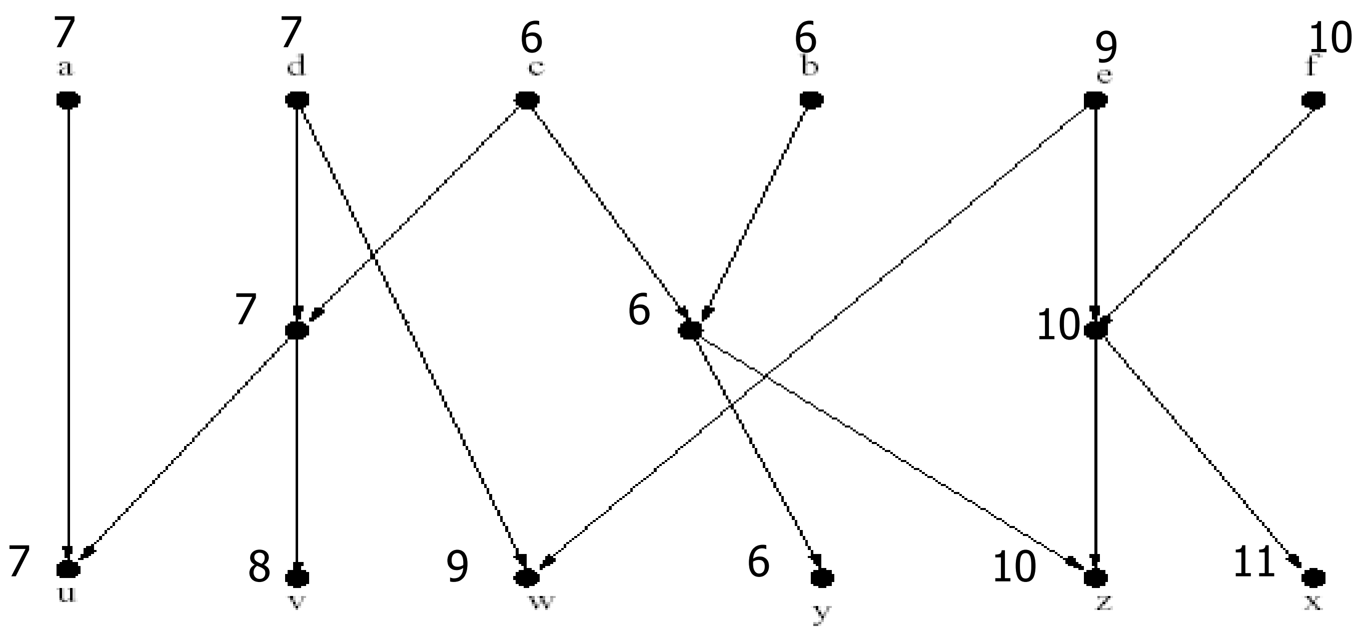
Computing up(e)

- We start by topologically sorting the transmuter (which is a DAG)
- We label each node in the transmuter with a weight, obtained by processing the transmuter in reverse topological order:
  - We label a sink node t(f) with r(f)
  - We label a non-sink node v with the minimum weight out of all its adjacent (already labeled) successors
- When all the nodes have been labeled, a source node s(e) is labeled with up(e) (and the corresponding swap edge)

An example


u 7
2 a
v 8
d
6
5
c
w
9
b
3
e
y
4
6
11
x
f
9
z
10

(a)


7
a
7
d
6
c
6
b
9
e
10
f
7
6
10
7
u
8
v
9
w
6
y
10
z
11
x

Time complexity for computing up(e)

1. Transmuter build-up: $O(m\,\alpha(m,n))$ time
2. Computing up(e) values:
   - Topological sorting: $O(m\,\alpha(m,n))$ time
   - Processing the transmuter: $O(m\,\alpha(m,n))$ time

Time complexity of the VCG-mechanism

Theorem

There exists a VCG-mechanism for the private-edge MST problem running in $O(m\,\alpha(m,n))$ time.

Proof.

Time complexity of g: $O(m\,\alpha(m,n))$

Time complexity of p: we compute all the values up(e) in $O(m\,\alpha(m,n))$ time.

■