ON THE PROBLEM OF FINDING ALL MINIMUM SPANNING TREES

João Guilherme Martinez

Rosiane de Freitas, Altigran Silva

Programa de Pós-graduação em Informática
Instituto de Computação
Universidade Federal do Amazonas

VII Latin American Workshop on Cliques in Graphs
La Plata, Argentina
November 8-11th, 2016

Contents

Introduction

- Let G be an undirected weighted graph with n vertices and m edges, we say that T is a spanning tree of G if it's a graph that connects all vertices from G and does not contain a cycle.

Introduction

- Let G be an undirected weighted graph with n vertices and m edges, we say that T is a spanning tree of G if it's a graph that connects all vertices from G and does not contain a cycle.
- T is a minimum spanning tree (MST) only if it has the minimal total weighting for its edges.

Introduction

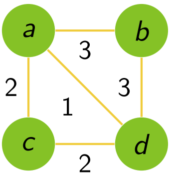
- The problem has optimal substructure and it can be solved by greedy algorithms in polynomial time.

Introduction

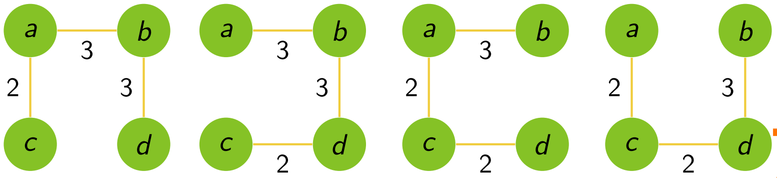
- The problem has optimal substructure and it can be solved by greedy algorithms in polynomial time.
- However such algorithms retrieve only one MST instance and therefore comes up the need to explore algorithms that can enumerate all possible MST's for a graph.

Example

Base graph:

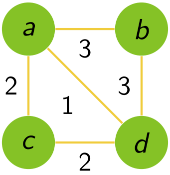


All spanning trees that are not minimum:

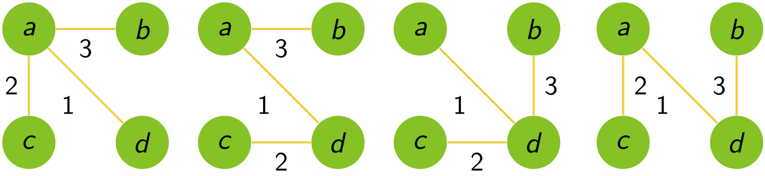


Example

Base graph:



All minimum spanning trees:



Contents

Related work

Algorithm	Year	Return	Time Complexity
Gabow and Myers	1978	All ST's	$O(n+m+N.n)$
Matsui	1997	All ST's	$O(n+m+N.n)$
Kapoor and Ramesh	1995	All ST's	$O(n+m+N)$
Shioura and Tamura	1995	All ST's	$O(n+m+N)$
Sörensen and Janssens	2005	All ST's (in order)	$O(N.m\log m+N^2)$
Perrin Wright	2000	All MST's	-
Yamada, Kataoka and Watanabe	2010	All MST's	$O(K.m\log n)$
Eppstein	1995	All MST's	$O(m+n\log n+K)$

Motivation

- On the development of an algorithm for information retrieval in relational databases, we choosed to work with the algorithm proposed by [Dourado et al., 2014] in *Algorithmic aspects of Steiner convexity and enumeration of Steiner trees*.

Motivation

- On the development of an algorithm for information retrieval in relational databases, we choosed to work with the algorithm proposed by [Dourado et al., 2014] in *Algorithmic aspects of Steiner convexity and enumeration of Steiner trees*.
- However one of the steps requires the generation of all MST's and this was a challenge.

Motivation

- On the development of an algorithm for information retrieval in relational databases, we choosed to work with the algorithm proposed by [Dourado et al., 2014] in *Algorithmic aspects of Steiner convexity and enumeration of Steiner trees*.
- However one of the steps requires the generation of all MST's and this was a challenge.
- Despite the existence of these algorithms, some of them where not implemented.

Our work

- Our proposal is to detail, implement and analyze the behavior in experiments of three of these algorithm's.

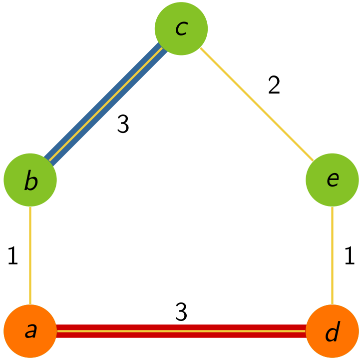
Related work

Algorithm	Year	Return	Time Complexity
Gabow and Myers	1978	All ST's	$O(n+m+N.n)$
Matsui	1997	All ST's	$O(n+m+N.n)$
Kapoor and Ramesh	1995	All ST's	$O(n+m+N)$
Shioura and Tamura	1995	All ST's	$O(n+m+N)$
Sörensen and Janssens	2005	All ST's (in order)	$O(N.m\log m+N^2)$
Perrin Wright	2000	All MST's	-
Yamada, Kataoka and Watanabe	2010	All MST's	$O(K.m\log n)$
Eppstein	1995	All MST's	$O(m+n\log n+K)$

Contents

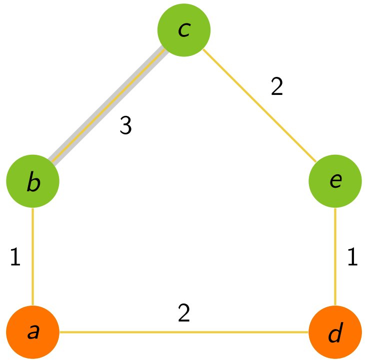
Property E and electability

- Consider $f = (a, d)$ and $g \in P = \{a, ..., d\}$.
- If $w(f) = w(g)$, f will be in at least one MST.



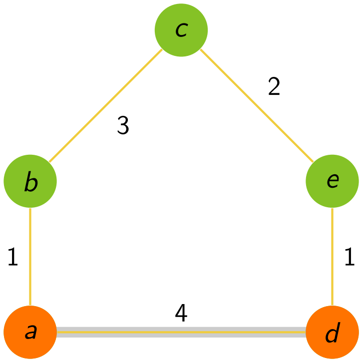
Property E and electability

- Consider $f = (a, d)$ and $g \in P = \{a, ..., d\}$.
- If $w(f) < w(g)$, T is not an MST, otherwise f would be choosen instead of g.



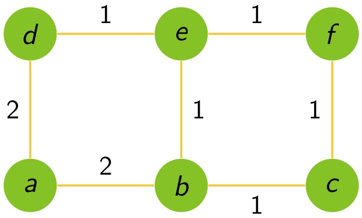
Property E and electability

- Consider $f = (a, d)$ and $g \in P = \{a, ..., d\}$.
- If $w(f) > w(g)$, f won't be in any MST.



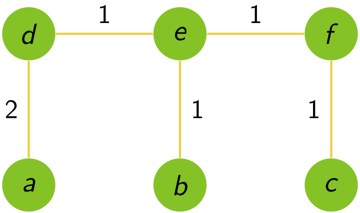
Example

Graph G



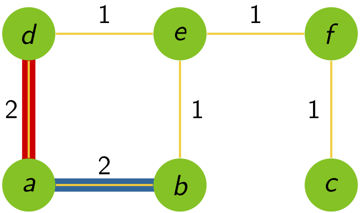
Example

MST T_0



Example

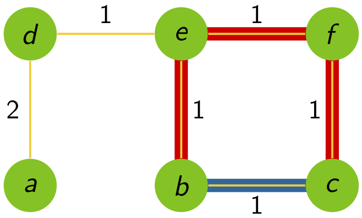
All candidates to be replaced by (a, b)



- $X(2) = \{(a, d)\}$

Example

All candidates to be replaced by (b, c)



- $X(1) = \{(b,e),(e,f),(f,c)\}$

Perrin Wright's algorithm

Weight	**Edges originally in tree**	**Edges candidates**
$w = 1$	$(b,e),(e,f),(f,c)$	$(b,c),(b,e),(e,f),(f,c)$
$w = 2$	(a,d)	$(a,b),(a,d)$

- Now we form the subsets.

Perrin Wright's algorithm

Weight	Edges originally in tree	Edges candidates
$w = 1$	$(b,e),(e,f),(f,c)$	$(b,c),(b,e),(e,f),(f,c)$
$w = 2$	(a,d)	$(a,b),(a,d)$

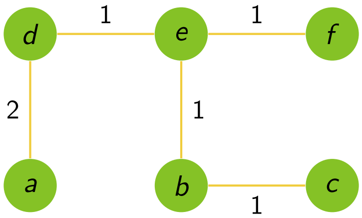
- Now we form the subsets.
- For the edges with $w = 1$, we can form $\binom{4}{3} = 4$ subsets of size 3.
- For $w = 2$, we can form $\binom{2}{1} = 2$ subsets of size 1.
- Lets call S the number of subsets formed.

Perrin Wright's algorithm

Weight	**Subsets**
$w = 1$	$\{[(b,c),(b,e),(e,f)],[(b,c),(b,e),(f,c)],[(b,c),(e,f),(f,c)],[(b,e),(e,f),(f,c)]\}$
$w = 2$	$\{[(a,b)],[(a,d)]\}$

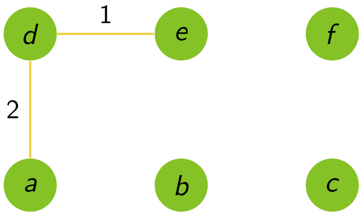
- Number of combinations $2 * 4 = 8$. Lets call it C.
- However, not all combinations are valid.
- Check if the tree is connected, or if it has a cycle.
- One of this checks should be enough.

Example



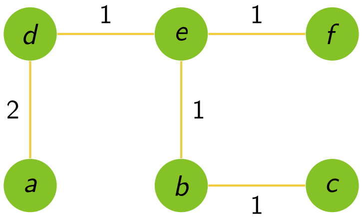
- T_0

Example



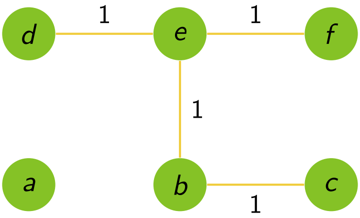
- Remove all edges in $X(1)$

Example



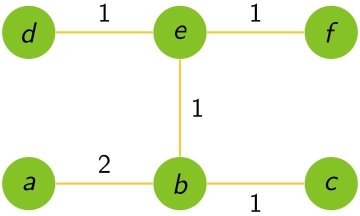
- Apply subset with $w = 1$ -> $(b, c), (b, e), (e, f)$

Example



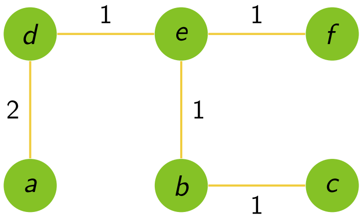
- Apply subset with $w = 1$ -> $(b, c), (b, e), (e, f)$
- Remove all edges in $X(2)$

Example



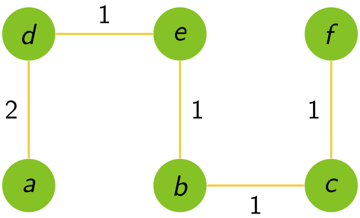
- Apply subset with $w = 1$ -> $(b, c), (b, e), (e, f)$
- Apply subset with $w = 2$ -> (a, b)
- MST's $= 1$

Example



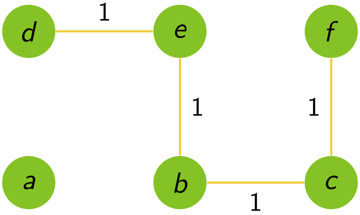
- Apply subset with $w = 1$ -> $(b, c), (b, e), (e, f)$
- Apply subset with $w = 2$ -> (a, d)
- MST's $= 2$

Example



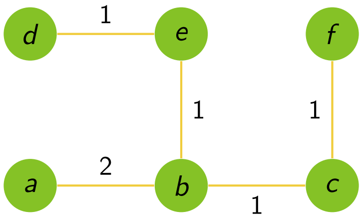
- Apply subset with $w = 1$ -> $(b, c), (b, e), (f, c)$

Example



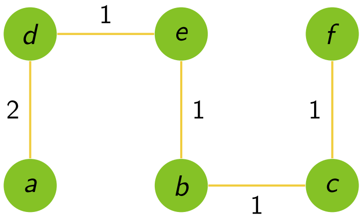
- Apply subset with $w = 1$ -> $(b, c), (b, e), (f, c)$
- Remove all edges in $X(2)$

Example



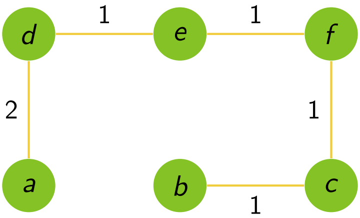
- Apply subset with $w = 1$ -> $(b, c), (b, e), (f, c)$
- Apply subset with $w = 2$ -> (a, b)
- MST's = 3

Example



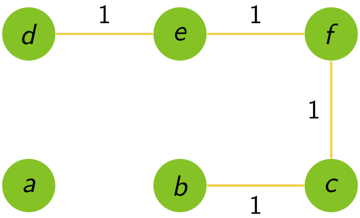
- Apply subset with $w = 1$ -> $(b, c), (b, e), (f, c)$
- Apply subset with $w = 2$ -> (a, d)
- MST's $= 4$

Example



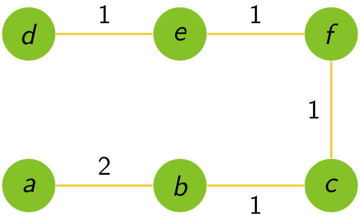
- Apply subset with $w = 1$ -> $(b, c), (e, f), (f, c)$

Example



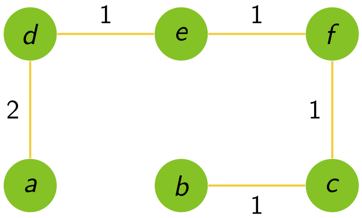
- Apply subset with $w = 1$ -> $(b, c), (e, f), (f, c)$
- Remove all edges in $X(2)$

Example



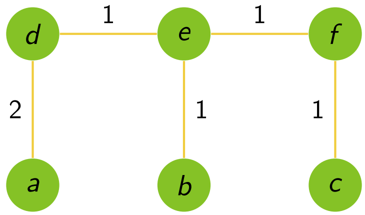
- Apply subset with $w = 1$ -> $(b, c), (e, f), (f, c)$
- Apply subset with $w = 2$ -> (a, b)
- MST's $= 5$

Example



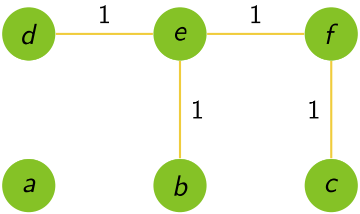
- Apply subset with $w = 1$ -> $(b, c), (e, f), (f, c)$
- Apply subset with $w = 2$ -> (a, d)
- MST's $= 6$

Example



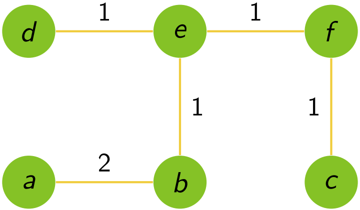
- Apply subset with $w = 1$ -> $(b, e), (e, f), (f, c)$

Example



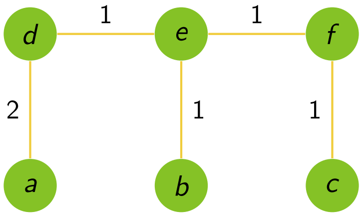
- Apply subset with $w = 1$ -> $(b, e), (e, f), (f, c)$
- Remove all edges in $X(2)$

Example



- Apply subset with $w = 1$ -> $(b, e), (e, f), (f, c)$
- Apply subset with $w = 2$ -> (a, b)
- MST's $= 7$

Example



- Apply subset with $w = 1$ -> $(b, e), (e, f), (f, c)$
- Apply subset with $w = 2$ -> (a, d)
- MST's $= 8$
- End of algorithm

Expected Time Complexity

- Get all non-tree edges, $O(m)$
- Get candidate edges, $O(mn)$
- Form subsets, $O(S)$
- Generate and check trees, $O(Cn)$
- TOTAL, $O(m + mn + S + Cn)$
- We know that $C \gg S$ and that $C \geq K$
- **TOTAL,** $O(mn + Cn)$

Contents

Description

- Three proposed algorithm's:
  - All_MST $= O(K(nm + n^2 \log n))$
  - All_MST$_1$ $= O(Kmn)$
  - All_MST$_2$ $= O(Km \log n)$

Description

- Three proposed algorithm's:
  - All_MST $= O(K(nm + n^2 \log n))$
  - All_MST$_1 = O(Kmn)$
  - All_MST$_2 = O(Km \log n)$

Description

- It uses a combinatorial optimization framework for solving enumerating problems.

Description

- It uses a combinatorial optimization framework for solving enumerating problems.
- Consider two edge sets: F and R.
- A spanning tree T is $(F,R)-$*admissible* if $F \subseteq T$ and $R \cap T = \emptyset$.

Description

- Let T be an MST of G, and e an arbitraty edge of T.

Description

- Let T be an MST of G, and e an arbitraty edge of T.
- Deleting e from T divides the tree into two connected components V_1 and V_2.

Description

- Let T be an MST of G, and e an arbitraty edge of T.
- Deleting e from T divides the tree into two connected components V_1 and V_2.
- $\text{Cut}(e)$ is the set of edges that can substitute e and reconnect V_1 and V_2.

Description

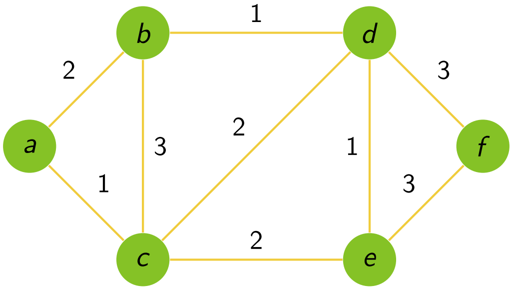
- Let T be an MST of G, and e an arbitraty edge of T.
- Deleting e from T divides the tree into two connected components V_1 and V_2.
- $\mathrm{Cut}(e)$ is the set of edges that can substitute e and reconnect V_1 and V_2.
- A substitute $S(e) := \{\tilde{e} \in \mathrm{Cut}(e) \mid \tilde{e} \neq e, w(\tilde{e}) = w(e)\}$.

Pseudo-code

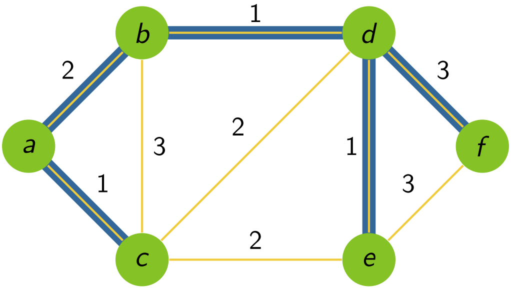
Algorithm 1: All_MST$_1$(F,R,T)

$k \leftarrow$ size of F
for $i = k+1, ..., n-1$ **do**
  find Cut(e^i)
  find, if one exists, a substitute $\tilde{e}^i \in S(e^i)$
for $i = k+1, ..., n-1$ **do**
  if $\tilde{e}^i$ *exists* **then**
    $T_i := T \cup \tilde{e}^i \setminus e^i$
    output T_i
    $F_i := F \cup \{e^{(k+1)}, ..., e^{(i-1)}\}$
    $R_i := R \cup \{e^i\}$
    All_MST$_1$(F_i,R_i,T_i)

Example

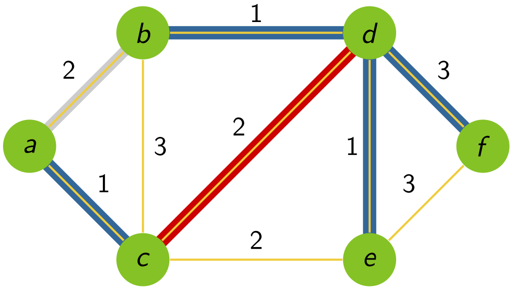


Example



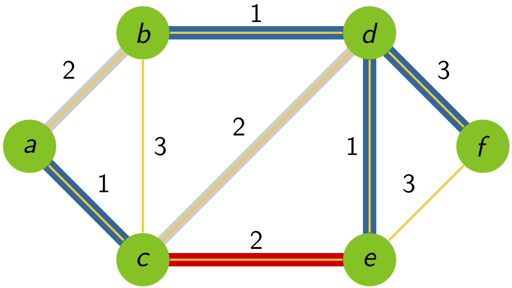
- $F = [\text{ - }]$
- $R = [\text{ - }]$

Example



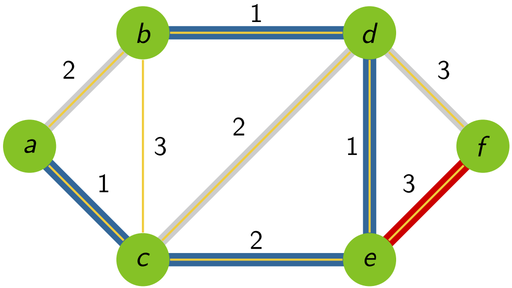
- $F = [\ (\mathrm{a,c})\ ]$
- $R = [\ (\mathrm{a,b})\ ]$

Example



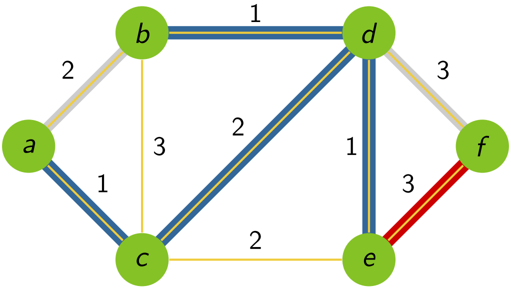
- $F = [$ (a,c) $]$
- $R = [$ (a,b),(c,d) $]$

Example



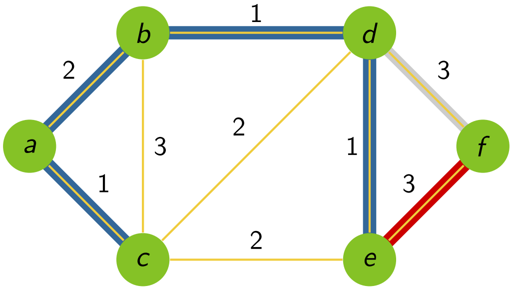
- $F =$ [(a,c),(c,e),(b,d),(d,e)]
- $R =$ [(a,b),(c,d),(d,f)]

Example



- $F = [\ (\text{a,c}),(\text{c,d}),(\text{b,d}),(\text{d,e})\ ]$
- $R = [\ (\text{a,b}),(\text{d,f})\ ]$

Example



- $F = [\ (\text{a,c}),(\text{a,b}),(\text{b,d}),(\text{d,e})\ ]$
- $R = [\ (\text{d,f})\ ]$

Time Complexity

- To find Cut(e), $O(m)$.

Time Complexity

- To find Cut(e), $O(m)$.
- At each subproblem, this is repeated at most $O(n)$.

Time Complexity

- To find Cut(e), $O(m)$.
- At each subproblem, this is repeated at most $O(n)$.
- **TOTAL,** $O(Kmn)$.

Contents

Eppstein's algorithm

- The main idea is to generate an equivalent graph (EG) from G by performing *sliding operations*.
- Each ST of $EG \Leftrightarrow$ MST of G.

The sliding operation

- Let edges $e = (u, v)$ and $f = (v, w)$ share a common vertex v, and suppose that $w(e) < w(f)$.

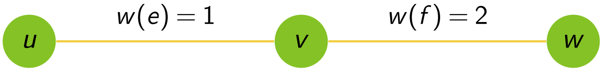
The sliding operation

- Let edges $e = (u, v)$ and $f = (v, w)$ share a common vertex v, and suppose that $w(e) < w(f)$.
- To perform a sliding operation:
  - Remove $f(v, w)$.
  - Insert $f'(u, w)$ with the same weight.

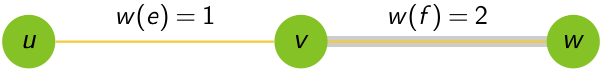
The sliding operation

- Let edges $e = (u, v)$ and $f = (v, w)$ share a common vertex v, and suppose that $w(e) < w(f)$.
- To perform a sliding operation:
  - Remove $f(v, w)$.
  - Insert $f'(u, w)$ with the same weight.
- Only perform if $w(e) < w(f)$.

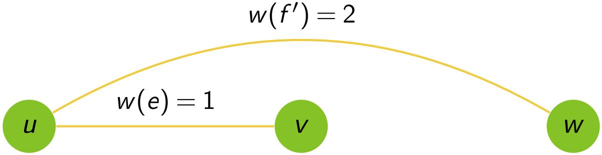
Example



Example



Example



Generating the Equivalent Graph

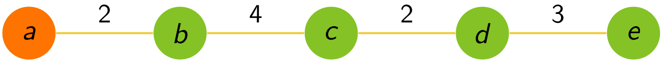
- Let T_0 be a MST in G.
- Choose a vertex to be root of T_0.

Generating the Equivalent Graph

- Let T_0 be a MST in G.
- Choose a vertex to be root of T_0.
- Form EG by repeatedly performing sliding operations through edges e and f as long as $e \in T_0$ and u is closer to the root of T_0 than is v.

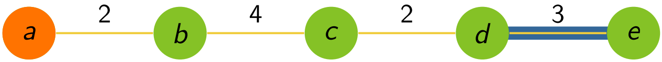
Example

a is the choosen root

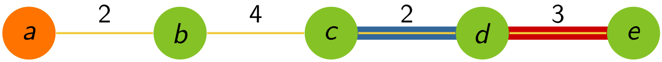


Example

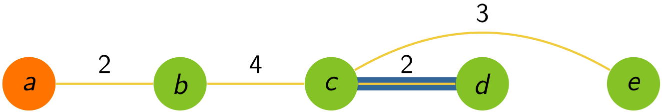
Now we perform sliding operations beginning from the edge fartest from the root



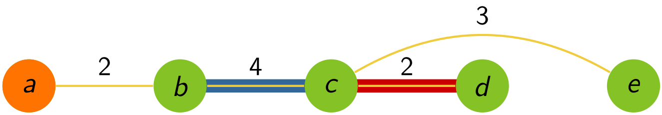
Example



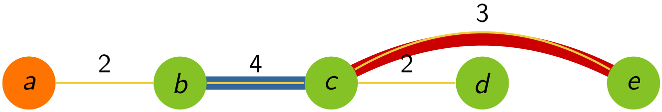
Example



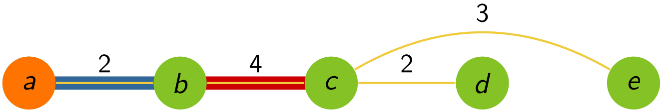
Example



Example

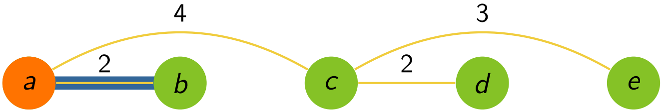


Example



Example

EG complete. Note that some "path compression" operations were made.



Optimizing the construction of EG

- Implemented directly, $O(mn)$.

Optimizing the construction of EG

- Implemented directly, $O(mn)$.
- The following simple technique suffices to achieve $O(m \log n)$ time.

Optimizing the construction of EG

- For any edge e in T_0, define the *heavy ancestor* of e to be the first edge on the path from e to the root having weight greater than that of e.

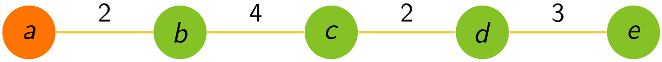
Pseudo-code

Algorithm 2: Sliding(G,u,ℓ,$root$)

```
mark u as visited
for each vertex v adjacent to u do
    if v is unvisited then
        if u = root then
            A[0] = Edge(u, v, cost(u, v))
            SLIDING(G,v,0,root)
            SLIDECHILDREN(u,v,0)
        else
            ℓ' ← BINSEARCH(A,cost(u, v),0,ℓ)
            Edge e = A[ℓ']
            A[ℓ'] = Edge(u, v, cost(u, v))
            SLIDING(G,v,ℓ',root)
            SLIDECHILDREN(u,v,ℓ')
            A[ℓ'] = e
```

Example

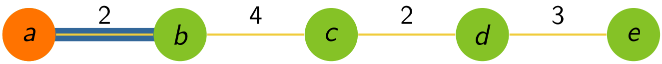
a is the root of this tree.



- $\ell =$
- $A =$

Example

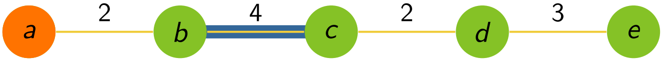
Don't have *heavy ancestor*



- $\ell = 0$
- $A = [(a, b)]$

Example

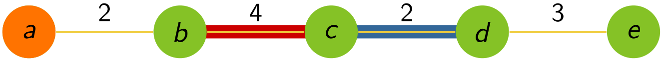
Don't have *heavy ancestor*



- $\ell = 0$
- $A = [(b, c)]$

Example

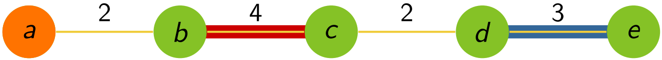
Heavy ancestor marked in red



- $\ell = 1$
- $A = [(b, c), (c, d)]$

Example

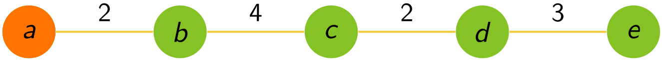
Heavy ancestor marked in red



- $\ell = 1$
- $A = [(b, c), (d, e)]$

Example

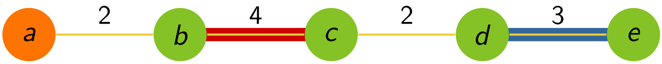
Now we perform the sliding operations



- $\ell = 1$
- $A = [(b, c), (d, e)]$

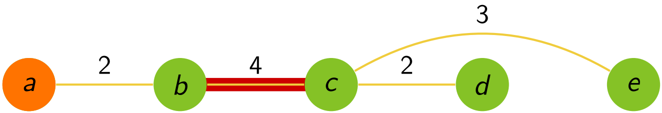
Example

Heavy ancestor marked in red



- $\ell = 1$
- $A = [(b, c), (d, e)]$

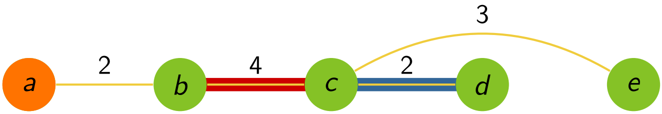
Example



- $\ell = 1$
- $A = [(b, c), (d, e)]$

Example

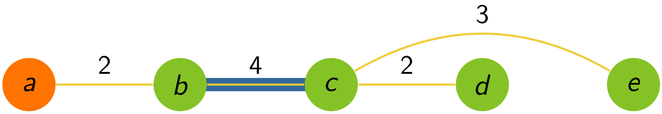
In this case, *heavy ancestor* is the previous edge, so nothing is done



- $\ell = 1$
- $A = [(b, c), (c, d)]$

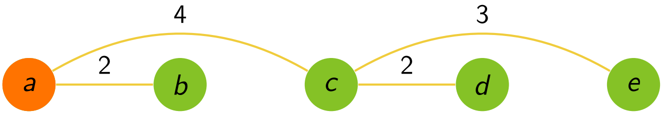
Example

In this case, as the edge does not have a *heavy ancestor*, we slide with the root vertex



- $\ell = 0$
- $A = [(b, c)]$

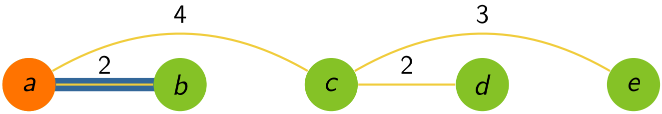
Example



- $\ell = 0$
- $A = [(b, c)]$

Example

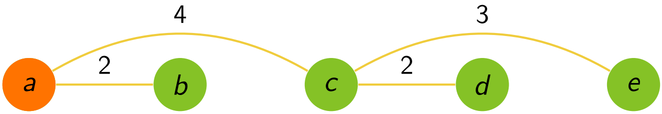
In this case, as the edge does not have a *heavy ancestor*, we slide with the root vertex



- $\ell = 0$
- $A = [(a, b)]$

Example

EG built



- $\ell = 0$
- $A = [(a, b)]$

Optimizing the construction of EG

- $O(\log n)$ per binary search

Optimizing the construction of EG

- $O(\log n)$ per binary search
- Total, $O(m \log n)$.

Optimizing the construction of EG

- $O(\log n)$ per binary search
- Total, $O(m \log n)$.
- By property E [Wright, 2000], we can delete from EG all edges not part of some MST.

Optimizing the construction of EG

- *Verifying Minimum Spanning Trees in Linear Time* - [Bazlamacci and Hindi, 1997]

Optimizing the construction of EG

- *Verifying Minimum Spanning Trees in Linear Time* - [Bazlamacci and Hindi, 1997]
- Retrieve the heaviest edge for each non-tree edge in time $O(m)$.

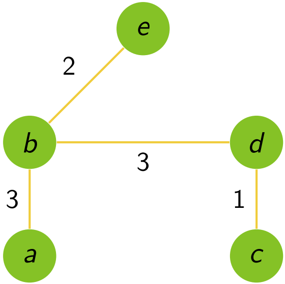
Optimizing the construction of EG

- First we make a Full Branching-Tree B_T corresponding to T_0 to make use of Property 1.

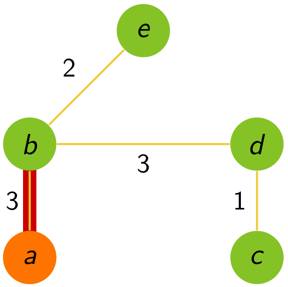
Property 1

The largest edge weight from x to y in T_0 is the same largest edge weight from x to y in B_T.

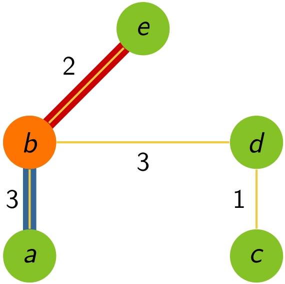
Example



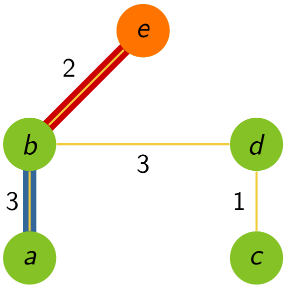
Example



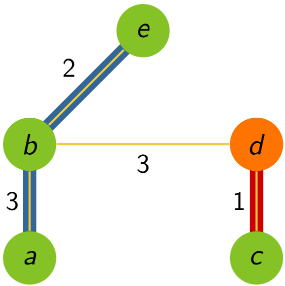
Example



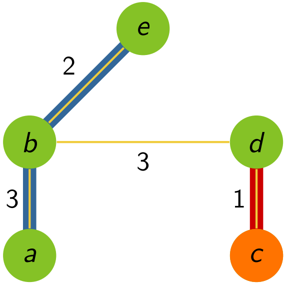
Example



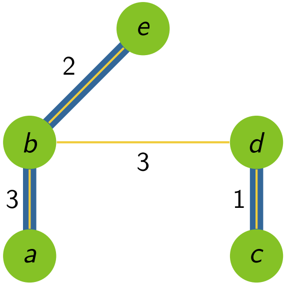
Example

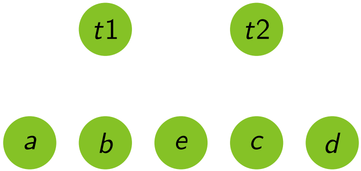


Example

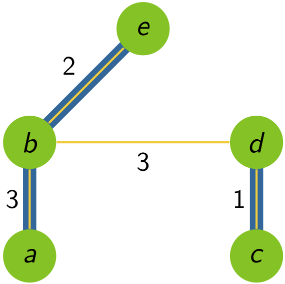


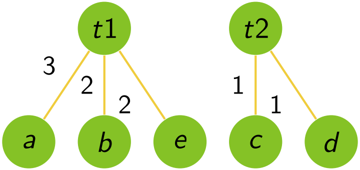
Example



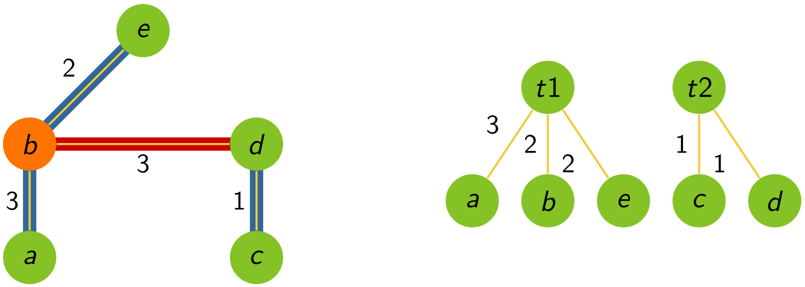


Example

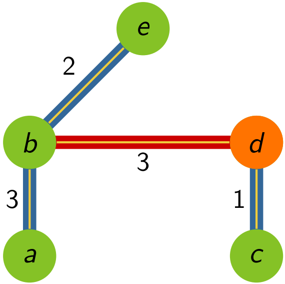


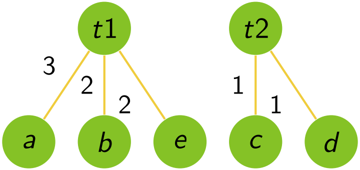


Example

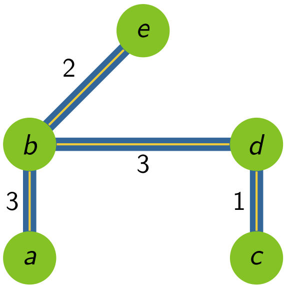


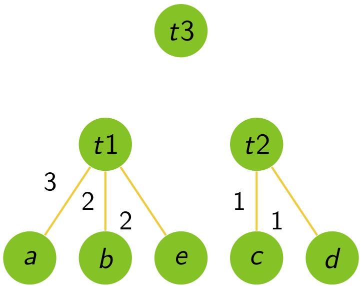
Example



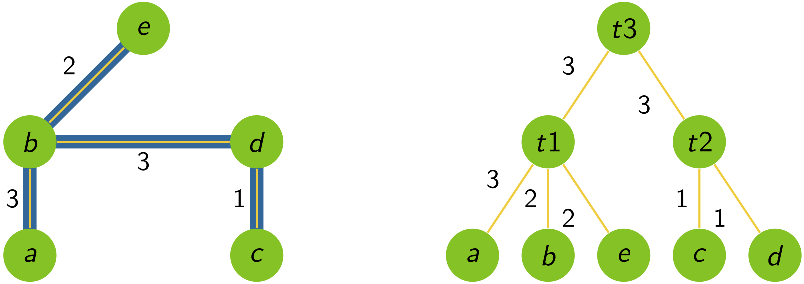


Example





Example



Optimizing the construction of EG

- [Schieber and Vishkin, 1988] algorithm ($O(m)$) to detect the lowest common ancestor (**lca**) between two vertices.

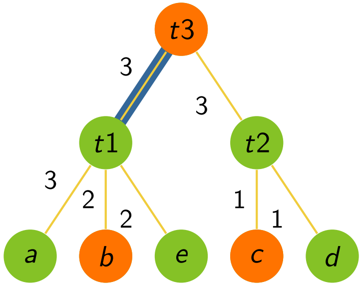
Optimizing the construction of EG

- [Schieber and Vishkin, 1988] algorithm ($O(m)$) to detect the lowest common ancestor (**lca**) between two vertices.
- [Bazlamacci and Hindi, 1997] algorithm ($O(m)$) to detect the heaviest edge between vertices.

Optimizing the construction of EG

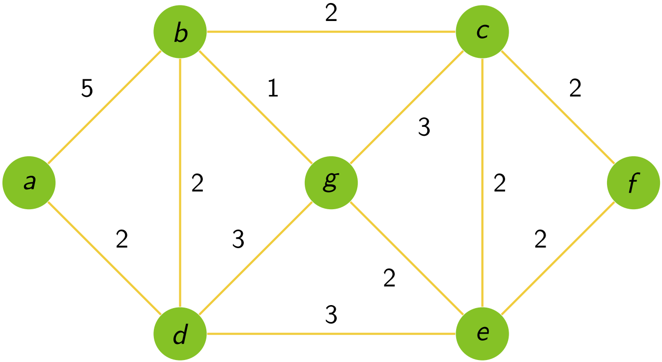
- [Schieber and Vishkin, 1988] algorithm ($O(m)$) to detect the lowest common ancestor (**lca**) between two vertices.
- [Bazlamacci and Hindi, 1997] algorithm ($O(m)$) to detect the heaviest edge between vertices.
- $(u, lca(u, v))$ and $(v, lca(u, v))$, which is the heaviest edge between (u, v).

Example



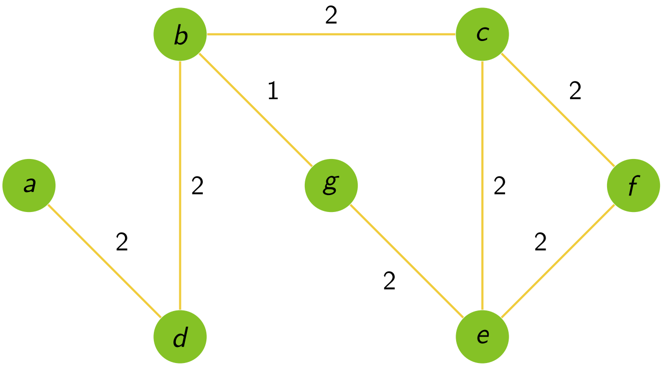
Example

Graph G



Example

ET_0 built



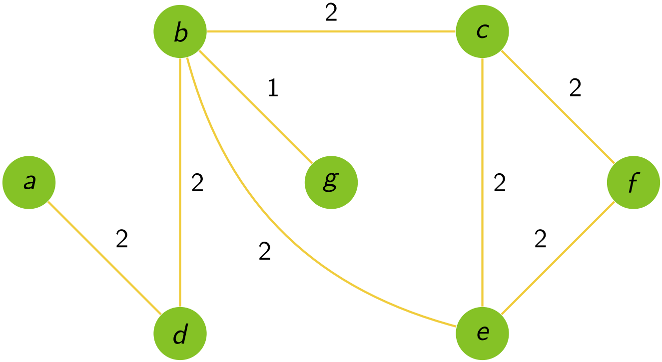
Optimizing the construction of EG

- With ET_0 built, we can now perform sliding operations in $O(n \log n)$

LAWCG
LA PLATA, ARGENTINA 2016

Example

EG built



Generation of MST's

- Apply [Kapoor and Ramesh, 1995] algorithm to retrieve all spanning trees from EG

Generation of MST's

- Apply [Kapoor and Ramesh, 1995] algorithm to retrieve all spanning trees from EG
- However, it has a complex description.

Generation of MST's

- Apply [Kapoor and Ramesh, 1995] algorithm to retrieve all spanning trees from EG
- However, it has a complex description.
- We decided to use [Shioura and Tamura, 1995] algorithm, that achieves the same result in the same time complexity: $O(n+m+N)$.

Generation of MST's

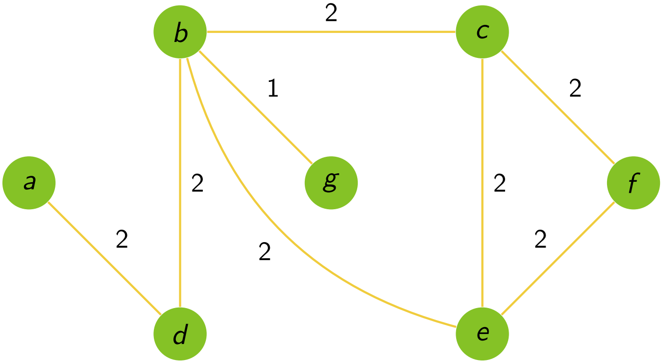
- Get first MST by a DFS on EG.

Generation of MST's

- Get first MST by a DFS on EG.
- Next, generates all ST's making all edge-replacement combinations.

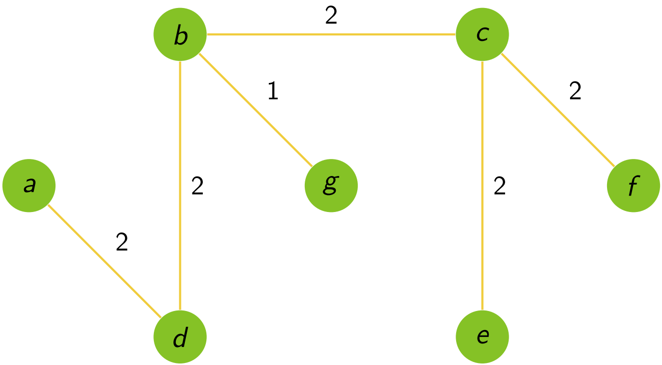
Example

EG graph



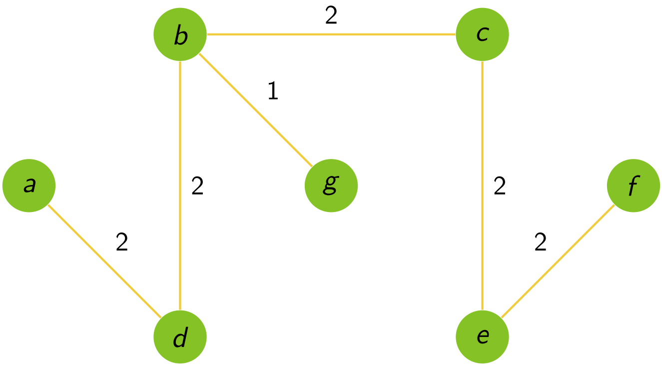
Example

DFS tree = MST 1



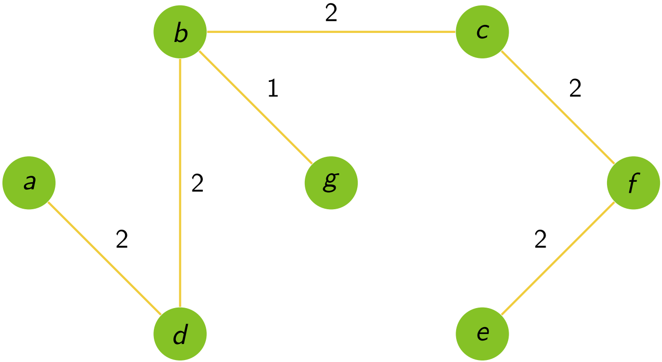
Example

MST 2



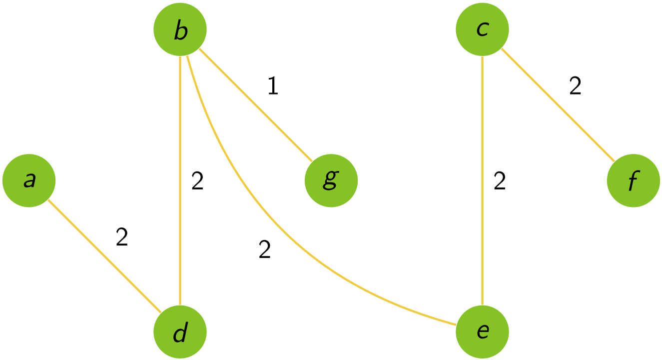
Example

MST 3



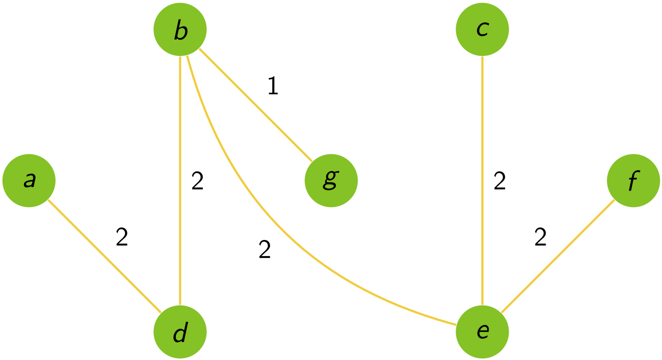
Example

MST 4



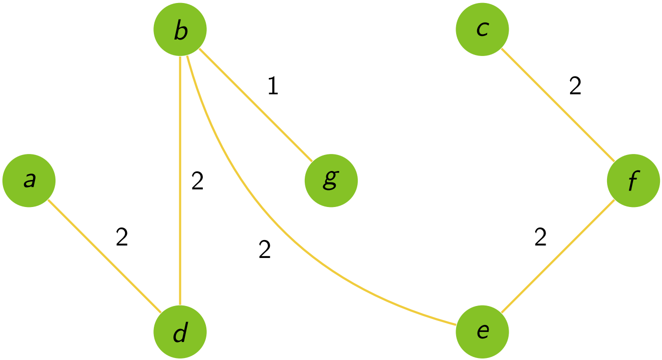
Example

MST 5



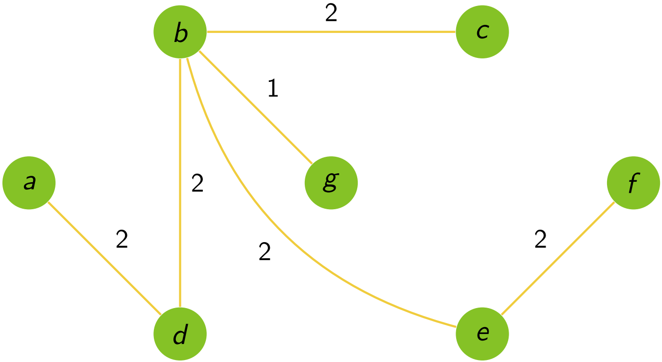
Example

MST 6



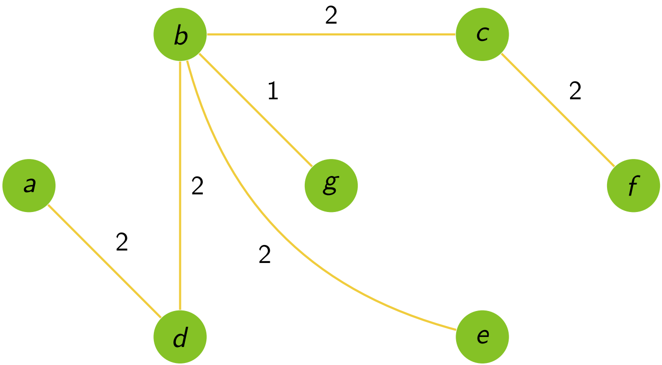
Example

MST 7



Example

MST 8



Time Complexity

- Generating first MST T_0, $O(m + n \log n)$

Time Complexity

- Generating first MST T_0, $O(m + n \log n)$
- Eliminating non-mst edges, $O(m)$

Time Complexity

- Generating first MST T_0, $O(m + n \log n)$
- Eliminating non-mst edges, $O(m)$
- Constructing EG, $O(n \log n)$

Time Complexity

- Generating first MST T_0, $O(m + n \log n)$
- Eliminating non-mst edges, $O(m)$
- Constructing EG, $O(n \log n)$
- Generation of all MST's, $O(m + n + K)$

Time Complexity

- Generating first MST T_0, $O(m + n \log n)$
- Eliminating non-mst edges, $O(m)$
- Constructing EG, $O(n \log n)$
- Generation of all MST's, $O(m + n + K)$
- **TOTAL,** $O(m + n \log n + K)$

Contents

Experiments

- The performance of the algorithms were analyzed in:
  - Complete Graphs
  - Random Graphs ($m = 3n$)
  - With equal edge weights
  - With random weights in $[1, 10^L]$ for $L = 2$ and $L = 3$.
  - Experiments based on [Yamada et al., 2010]

Experiments

Graph	MST's	Perrin	Yamada	Eppstein
K5	125	0,003	0,002	0,000
K6	1269	0,054	0,029	0,006
K7	16807	1,188	0,434	0,082
K8	262144	31,002	7,5482	1,335
K9	4782969	960,126	152,972	23,957
K10	100000000	*	3541,48	502,631

Cayley's formula $= n^{n-2}$

Experiments

L	Graph	MST's	Perrin	Yamada	Eppstein
2	K20	4	0,004	0,001	0,001
	K40	120	0,046	0,056	0,006
	K60	176580	69,633	209,633	0,077
	K80	5971968	*	4218,73	0,167
3	K20	1	0,003	0,000	0,001
	K40	1	0,030	0,003	0,006
	K60	12	0,102	0,039	0,014
	K80	2	0,234	0,016	0,024
	K100	6	0,473	0,050	0,039
	K120	8	0,816	0,169	0,057
	K140	256	1,432	3,992	0,081
	K160	1152	2,681	8,282	0,104
	K180	208	2,902	1,940	0,133
	K200	1152	4,830	16,314	0,167

Experiments in Random Graphs

L	Vertices	MST's	Perrin	Yamada	Eppstein
2	200	16	0,078	0,087	0,011
	400	48	0,350	1,720	0,026
	600	2048	6,467	76,091	0,042
	800	6144	23,276	230,59	0,06
3	200	1	0,065	0,051	0,011
	400	1	0,269	0,051	0,011
	600	2	0,614	0,755	0,042
	800	2	1,116	1,429	0,061

7 LAWCG
LA PLATA, ARGENTINA 2016

Contents

Concluding remarks

- Eppstein's algorithm has the best time complexity.

Concluding remarks

- Eppstein's algorithm has the best time complexity.
- But it is complex and produces equivalent graphs with path compressions.

Concluding remarks

- Eppstein's algorithm has the best time complexity.
- But it is complex and produces equivalent graphs with path compressions.
- Perrin's and Yamada's algorithm have much simpler steps, but worse time complexity.

Concluding remarks

- Eppstein's algorithm has the best time complexity.
- But it is complex and produces equivalent graphs with path compressions.
- Perrin's and Yamada's algorithm have much simpler steps, but worse time complexity.
- Perrin's algorithm is better than Yamada's for random weight edges, but much worse for equal weight edges.

References

C. F. Bazlamacci and K. S. Hindi. Verifying minimum spanning trees in linear time. In *Symposium on Operations Research*, pages 139–144, Heidelberg, September 1997. Springer-Verlag.

Mitre Dourado, Rodolfo Oliveira, and Fábio Protti. Algorithmic aspects of steiner convexity and enumeration of steiner trees. *Annals of Operations Research*, 223(1): 155–171, 2014.

David Eppstein. Representing all minimum spanning trees with applications to counting and generation. Technical Report 95-50, Univ. of California, Irvine, Dept. of Information and Computer Science, Irvine, CA, 92697-3425, USA, 1995. URL `http://www.ics.uci.edu/~eppstein/pubs/Epp-TR-95-50.pdf`.

Harold N. Gabow and Eugene W. Myers. Finding All Spanning Trees of Directed and Undirected Graphs. *SIAM Journal on Computing*, 7(3):280–287, August 1978. ISSN 0097-5397. doi: 10.1137/0207024. URL `http://epubs.siam.org/doi/abs/10.1137/0207024`.

References (cont.)

Sanjiv Kapoor and H. Ramesh. Algorithms for enumerating all spanning trees ofundirected and weighted graphs. *SIAM J. Comput.*, 24(2):247–265, April 1995. ISSN 0097-5397. doi: 10.1137/S009753979225030X. URL http://dx.doi.org/10.1137/S009753979225030X.

T. Matsui. A Flexible Algorithm for Generating All the Spanning Trees in Undirected Graphs. *Algorithmica*, 18(4):530–543, August 1997. ISSN 0178-4617. doi: 10.1007/PL00009171. URL http://link.springer.com/10.1007/PL00009171.

Baruch Schieber and Uzi Vishkin. On finding lowest common ancestors: Simplification and parallelization. In *AWOC*, volume 319 of *Lecture Notes in Computer Science*, pages 111–123. Springer, 1988.

Akiyoshi Shioura and Akihisa Tamura. Efficiently scanning all spanning trees of an undirected graph. *J. Operation Research Society Japan*, 38:331–344, 1995.

Kenneth Sorensen and Gerrit K. Janssens. An algorithm to generate all spanning trees of a graph in order of increasing cost. *Pesquisa Operacional*, 25:219 – 229, 08 2005. ISSN 0101-7438.

Perrin Wright. Counting and constructing minimal spanning trees, 2000.

References (cont.)

Takeo Yamada, Seiji Kataoka, and Kohtaro Watanabe. Listing all the minimum spanning trees in an undirected graph. *Int. J. Comput. Math.*, 87(14):3175–3185, 2010. doi: 10.1080/00207160903329699. URL http://dx.doi.org/10.1080/00207160903329699.

Muchas gracias!

joaogam@icomp.ufam.edu.br
rosiane@icomp.ufam.edu.br
alti@icomp.ufam.edu.br

7 LAWCG
LA PLATA, ARGENTINA 2016